Audit Report

Router Integration

v1.0

May 29, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Kailaasa Infotech Pte Ltd to perform a security audit of Router Integration.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

During this audit phase, we conducted a thorough review of the entire lifecycle of requests going through the Router protocol. This lifecycle begins when a user initiates a cross-chain request on the source chain. Subsequently, events emitted by the gateway contract will trigger orchestrators to detect, attest, and forward the request details to Router Chain, which will get processed by the cross-chain module. A relayer is then tasked with executing the transaction on the destination chain.

Upon successful execution on the destination chain, the corresponding gateway emits an event to signify completion. This event is relayed back to Router Chain by the orchestrators, and an acknowledgment is generated and transmitted to the source chain to notify the origin contract.

Throughout our review process, our auditors have tried to find any potential flaws in the interaction between components and in the protocol's state management.

Note that findings related to the integration of various components reported in other audit reports conducted for Router protocol components are also highlighted in the Integration Phase section.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Router Chain is a layer one blockchain focusing on blockchain interoperability, enabling cross-chain communication with CosmWasm middleware contracts.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the overall codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**High**	The overall code complexity is flagged as high due to the complex cross-chain interactions and multiple component integrations.
Code readability and clarity	**Low**	The overall code readability and clarity are flagged as low due to the unfinished codebases, which contain many TODO comments, mock functions, and unimplemented functionalities.
Level of documentation	**Medium-High**	The Router team provided detailed documentation, along with recorded videos and insightful architectural diagrams.
Test coverage	**Low**	The overall test coverage is flagged as low due to the lack of working tests across most components, test failures due to outdated test cases, and insufficient test coverage throughout the codebase.

Summary of Findings

No	Description	Severity	Status
1	Failed cross-chain requests are repeatedly processed in Router Chain's `EndBlocker`, potentially causing Denial-of-Service	Critical	Resolved
2	Malicious relayer can purposefully fail cross-chain requests on EVM destination chains	Major	Resolved
3	Fee settlement can be bypassed for inbound and crosstalk requests, resulting in relayers not being compensated	Major	Resolved
4	NEAR gateway contract does not allocate sufficient gas for handler calls	Major	Resolved
5	`AssetBridge` deviates from standard safety practices by omitting ASM for custom security checks	Minor	Acknowledged

Detailed Findings

1. Failed cross-chain requests are repeatedly processed in Router Chain's `EndBlocker`, potentially causing Denial-of-Service

Severity: Critical

Cross-chain requests are processed at the end of each block in Router Chain's `EndBlocker` function. A request that is ready to be executed, i.e., with the status set to `types.CROSSCHAIN_TX_READY_TO_EXECUTE`, is handled in the `Execute` function and remains in this status until the cross-chain request is executed on the destination chain and orchestrators report the execution back to Router Chain.

If executing the cross-chain request on the destination chain errors and the transaction reverts, relayers will attempt to retry the request. However, if the request repeatedly fails, Router Chain will indefinitely process the request in the `EndBlocker` function, resulting in an ever-growing list of requests to be processed. This can cause a Denial-of-Service (DoS) attack on Router Chain.

Such a failing cross-chain request can be caused on an EVM destination chain by having the `routeRecipient` set to `0x0000000000000000000000000000000000000000`, bypassing the empty address check in the `iSend` function and causing the transaction to revert on the destination chain due to the zero-address validation in the `iReceive` function.

Similarly, this can be caused by a malicious actor that uses an ASM contract that always reverts.

Recommendation

We recommend employing a timeout for executing cross-chain requests to ensure that requests that keep failing are not indefinitely processed in the `Execute` function in each block.

Status: Resolved

2. Malicious relayer can purposefully fail cross-chain requests on EVM destination chains

Severity: Major

The Solidity `GatewayUpgradeable` contract receives cross-chain requests with the `iReceive` function, called by permissionless relayers, and calls the specified handler contract's `iReceive` function.

This external `call` forwards all but 1/64 gas to the called contract, a measure employed by the EVM in EIP-150 to mitigate DoS attacks.

If the supplied 63/64 gas to the external call is insufficient, the call runs out of gas, while the caller `iReceive` function still has 1/64 gas available to finish the execution and to emit the `IReceiveEvent` event that will be picked up by orchestrators and forwarded to Router Chain as an acknowledgment request. This acknowledgment packet will have the `execFlag` set to `false`, signaling to the source chain contract that the cross-chain request has failed on the destination chain. This concludes the cross-chain request.

This issue can be exploited by a malicious relayer by frontrunning a legitimate relayer, using its payload, and meticulously setting the gas limit such that the external call to the handler contract will be guaranteed to run out of gas.

Consequently, this allows an attacker to cause a Denial-of-Service for cross-chain requests, as the destination contracts will not be able to process the request and run out of gas successfully.

Similarly, this vulnerability has also been observed in the `iAck` function of the `GatewayUpgradeable` contract and the `iReceive` function of the `AssetBridge` contract.

Recommendation

We recommend enforcing a reasonably high minimum gas threshold for the remaining gas after the external call, similar to the `iRelayMessage` function of the `AssetForwarder` Solidity contract.

Status: Resolved

3. Fee settlement can be bypassed for inbound and crosstalk requests, resulting in relayers not being compensated

Severity: Major

Cross-chain request fees are settled on Router Chain as part of the `SettleFees` function called in every block.

However, if the fee settlement in the called `SettleFeeCrosstalkRequest` function errors, for example, if the fee payer is unapproved, the acknowledgment request's status is anyway set to `CROSSCHAIN_ACK_TX_COMPLETED` in the `CompleteNoAckRequests` function, effectively bypassing the fee settlement. As a result, the relayer will not be compensated for the cross-chain request.

Similarly, this issue also exists in the `SettleAckFees` function.

Recommendation

We recommend settling the fees in the `CompleteNoAckRequests` and `CompleteAckRequests` functions if the fees have not been settled yet.

Status: Resolved

4. NEAR gateway contract does not allocate sufficient gas for handler calls

Severity: Major

When the NEAR `gateway` contract calls the `i_receive` function of the handler contract, it only allocates a gas limit of 5 * `TGAS`. This is problematic because the allocated gas amount is insufficient to finish all the executions without causing an out-of-gas error.

The NEAR `asset-forwarder` and `asset-bridge` contracts both implement the `i_receive` function that will be executed by the `gateway` contract. However, both contracts will consume more than the allocated gas amount, causing the transaction to fail due to an out-of-gas error.

For example, the NEAR `asset-forwarder` contract will allocate 10 * `TGAS` when transferring funds to the recipient, while the NEAR `asset-bridge` contract already consumes the full gas amount when calling the `handle_ireceive_callback` function.

Consequently, all handler calls that consume gas units larger than 5 * `TGAS` will fail due to an out-of-gas error.

Recommendation

We recommend allocating more static gas when calling the handler's `i_receive` function in the `gateway` contract.

Status: Resolved

5. `AssetBridge` deviates from standard safety practices by omitting ASM for custom security checks

Severity: Minor

The Router protocol suggests that applications use Additional Security Modules (ASM) for additional security checks while processing the cross-chain request. As the `AssetBridge` contract is an application implemented by the Router team for token bridging, it should use ASM for additional security checks.

The current impact is that there are no maximum limits on deposits, which is highly advisable. Additionally, ASM modules could be upgraded to enhance the bridge's security further.

Recommendation

We recommend using ASM during packet construction for the `iSend` function of the `gateway` contract.

Status: Acknowledged

The client states that the ASM provides additional security on top of Router Chain’s 2/3 voting power. It is an optional feature that can be used for added security measures like delayed execution and Oracle-based execution. Currently, the client does not need these features. However, should they require them in the future, they will utilize the functionality provided by Router Chain.

Integration Phase

Phases

The Router audit is split into multiple phases that each consist of one or more components. All components are reviewed by Oak Security as part of the audit engagement and published as individual reports. Detailed architectural diagrams that illustrate the flow and structure can be found in Appendix A.

Note that integration-based findings from other phases are also highlighted in the relevant sections below. For more information regarding the audit scope and findings, please refer to the reports respectively.

Router Chain

Router Chain is the main application of the Router protocol and is built based on Cosmos SDK and Tendermint. All cross-chain requests must pass through Router Chain for relaying to the destination chain.

This component is mainly responsible for validating orchestrator attestations, handling cross-chain requests, transfers and fees, and Oracle price feeds. Most configurations can be updated through governance, while the Router team controls the rest.

The following findings are from the Router Chain 2 report, but have been added here since they related to the integration of the various Router components:

No	Description
1	Blocked cross-chain requests are indefinitely retried and could be exploited for denial-of-service attacks
3	Unapproved fee payers can block legitimate requests which can be exploited for denial-of-service attacks
4	Anyone can submit forged cross-chain requests
5	Anyone can submit forged cross-chain ack requests
9	Failed cross-chain ack requests are incorrectly stored as successful
10	Executing crosstalk requests with a missing IBC relayer config results in indefinite retries and liveness issues
28	The `GetTokenPrice` query plugin lacks support for price staleness check

Router Orchestrator

Router Orchestrator is a standalone application run by validators to attest to events continuously emitted by the gateway and voyager contracts. The queried events are converted into messages and dispatched to Router Chain in batches.

This component is mainly responsible for picking up the events emitted by transactions and forwarding them to Router Chain. This includes user-side transactions initiated in the source chain and acknowledgments back to the destination chain.

The following findings are from the Router Orchestrator report, but have been added here since they related to the integration of the various Router components:

No	Description
1	Events emitted from the EVM gateway and voyager contracts are potentially processed out of order, resulting in events being skipped and not sent to Router Chain
2	Inbound and outbound `CROSSTALK` requests originating from non-Cosmos chains are ignored and not able to be relayed
3	Transaction origin is hardcoded as an empty string when transforming an `iSend` event
4	A potential unconfirmed block is processed in the NEAR event listener
8	Incorrect gas price denom used
14	The NEAR network used for querying events is hard-coded to the testnet

Router EVM and NEAR Gateway Contracts and WASM Bindings

This phase consists of three components: the Router EVM gateway contract, the Router NEAR gateway contract, and the Router WASM bindings.

The Router gateway contracts are bridge contracts that can connect other chains to Router Chain. For example, the Router EVM gateway contract can be deployed on any EVM-compatible chains to bridge the chain to Router Chain. They represent the entrance and exit points of a cross-chain request and are deployed on both source and destination chains.

The gateway components are mainly responsible for initiating a cross-chain request with the `iSend` function, handling incoming requests with the `iReceive` function, and handling acknowledgments returned from destination chains with the `iAck` function. The `iReceive` function is also implemented to handle token minting of `ROUTE` tokens, Additional Security Modules (ASM) validation, and handler address execution.

The `iReceive` and `iAck` functions ensure that at least 66% of the network validators have signed and attested to the message to prevent fraudulent requests. The `setDappMetadata` entry point is implemented for dApps to specify the fee payer address that will cover the transaction fees when initiating cross-chain requests.

The Router WASM bindings represent custom bindings that allow CosmWasm contracts to access Router features. For example, the contract can query the token price by its symbol using the `RouterQuery::TokenPrice` message.

The following findings are from the Router EVM and NEAR Gateway Contracts and WASM Bindings report, but have been added here since they related to the integration of the various Router components:

No	Description
1	Incorrect `isReadCall` implementation allows infinite token mints
2	Duplicate `IReceiveEvent` event nonces in the NEAR gateway contract resulting in stuck cross-chain requests
3	`ROUTE` tokens are not minted for invalid requests, causing a loss of funds
4	ASM contract state is committed when token mint fails
7	Incomplete state rollback for failures in minting the `ROUTE` token
8	Execution status is incorrectly set to success when the handler address cannot be parsed
9	State rollbacks are not implemented correctly, preventing the failed packet from being retried
10	The reentrancy lock mechanism in the NEAR `GatewayUpgradeable` contract can be abused to grief the contract

No	Description
12	NEAR gateway contract does not handle read calls
26	Inconsistent function signature when verifying cross-chain requests

Router Voyager Forwarder and CosmWasm Gateway

This phase consists of three components: the Router Asset Forwarder contracts, the Router Asset Forwarder middleware contract, and the CosmWasm gateway contract.

The asset forwarder contracts allow users to perform token transfers from the source chain and forwarders to complete the transfer on the destination chain. For example, a user calls the `iDepositMessage` function to deposit USDT on the source chain. In the meantime, the forwarder notices the deposit request and sends the funds to the destination chain's asset forwarder contract with the `iRelay` function.

Both transactions' events will eventually be attested by orchestrators and sent to Router Chain. This would cause the asset forwarder middleware contract's sudo handler to be executed.

The middleware contract is responsible for ensuring deposit requests are satisfied by forwarders correctly and blocked requests are refundable. Router Chain will execute the contract's sudo handler during user deposits, forwarders pay the funds, or acknowledgment needs to be relayed back to the source chain. The contract also allows the withdrawal of blocked funds and the refund of failed requests.

The CosmWasm gateway contract design is similar to the previous section's Router EVM and NEAR gateway contracts, with the difference of supporting Cosmos SDK chains with the `wasm` module enabled. The contract utilizes IBC to initiate and receive cross-chain requests through established channels. The contract initiates a cross-chain request by sending the packet through IBC, which will be parsed by Router Chain and forwarded to the destination chain's middleware contract `ibc_packet_receive` handler. Once the execution finishes, Router Chain will send an acknowledgment packet to the source chain's `ibc_packet_receive` handler for further processing.

The following findings are from the Router Voyager Forwarder and CW Gateway report, but have been added here since they related to the integration of the various Router components:

No	Description
2	Attackers can create deposit requests without including funds
4	Nonce is incorrectly incremented when fungible tokens transfer fails
6	Lowercasing case-sensitive addresses causes unexpected behavior and loss of funds
7	Incorrect tokens are used to account for claimable tokens, causing a loss of funds
9	Erroneous claimable updates lead to the loss of the forwarder's funds
10	Extra fees can be unboundedly added to a deposit, resulting in failure to conclude the deposit
11	Updating the forwarder's claimable amount during a withdrawal request uses the

	wrong decimal precision, resulting in the ability to steal funds
14	Continuously failing ERC-20 token transfers in the `iReceive` function leads to stuck funds
15	Timeout for `SendPacket::ReceivePayload` IBC packets are not handled, resulting in lost `ROUTE` tokens
10	`HandlerExecMsg::IReceive` message is sent to the handler contract regardless of the ASM contract execution result
16	Unordered IBC channel is incorrectly enforced, resulting in out-of-order IBC packets
18	Incorrect data parsing for `HandlerExecMsg::IReceive` and `HandlerExecMsg::IAck` messages resulting in lost `ROUTE` tokens
19	Panic in handler callback causes denial of service and loss of funds
20	Inability to withdraw pending forwarder funds if there is no matching fund deposit
21	Failed `ASM` contract call or `ROUTE` token minting does not abort the whole transaction, causing partial state to be committed
22	The reentrancy lock mechanism in the NEAR `asset-forwarder` contract can be abused to grief the contract
24	Hardcoded gas limits might cause cross-chain messages to fail
26	Extra fees added to a relayed deposit cannot be withdrawn
27	User's `create_refund_request` can grief forwarders, preventing fund retrieval
28	The sender chain will not be notified of errors via IBC acknowledgment
29	Handling `RequestPayload` IBC packet will fail due to insufficient integer values
32	Denial of service due to unbounded processing of forwarder balances
34	Extra fee token's liquidity is updated with an incorrect value
37	Callback functions may run out of gas, resulting in inconsistent states of the NEAR `asset-forwarder` contract
38	Failed `iAck` messages can be replayed
41	Specifying a large `dest_amount` value could lead to funds overspending
44	Using Solidity's `transfer` function may prevent relaying funds to the destination chain
50	Incompatibility of deposit ID integer types
56	Hardcoded packet version for cross-chain requests

Router Voyager Forwarder (application)

Not to be confused with asset voyager contracts above, the Router Voyager Forwarder is a standalone application run by forwarders that continuously listens to fund deposit events on the source chain and relays the funds to the destination chain. Forwarders are incentivized to relay profitable cross-chain requests, which are determined by the fees set by the user. Once the relay is completed, the forwarder can claim a refund with the fees from the asset forwarder middleware contract.

This component is mainly responsible for ensuring user-deposited funds are bridged while ensuring liquidity is available in the destination chain. In return, forwarders can receive fees as part of the bridging process. However, they can choose not to relay unprofitable transactions, potentially causing deposit requests to be unfulfilled.

The following findings are from the Router Voyager Forwarder report, but have been added here since they related to the integration of the various Router components:

No	Description
11	Missing query pagination handling could lead to partial data retrieval
18	Administrators updating on-chain parameters could cause all connected forwarders to crash
20	Hardcoded high gas price leads to inefficiencies and potential stop of forwarder operations

Router DexSpan

The Dexspan contract represents a swap-aggregator facilitating token exchanges when performing token bridging. This component is integrated with the Router EVM asset bridge contract to allow users to swap their tokens before performing a token transfer request, all in a single transaction.

The following findings are from the Router DexSpan report, but have been added here since they related to the integration of the various Router components:

No	Description
1	Missing transaction revert could lead to loss of funds
6	Deposits in native currency always revert
13	The `assetBridge` and `assetForwarder` could drain funds from the contract by sending a faulty swap message
31	Missing usage of the `IBridge` interface

Router Asset Bridge

The asset bridge contracts act as an application that allows users to transfer funds from the source chain to the destination chain with optional execution instructions. Behind the scenes, the asset bridge contract calls the gateway contract to initiate the cross-chain request. At the time of writing, Solidity and NEAR asset bridge contracts are available to support EVM-based and NEAR blockchains.

This component also features a CosmWasm middleware contract that validates and forwards requests to the destination chain. Sudo handlers are implemented to receive and handle token transfer requests and IBC acknowledgments.

If any error occurs when receiving a cross-chain request, the request is reverted back to the source chain as an inbound request. If not, the request will be dispatched to the destination chain as an outbound request. Successful executions will return an acknowledgment to the source chain to notify the user, while failed executions can be retried later by the owner.

This component allows users to transact tokens from the source chain to the destination chain seamlessly. Unlike asset forwarder contracts, this component relies on token minting and burning mechanisms. Funds sent by users will be locked on the source chain and released on the destination chain.

If the destination contract does not hold sufficient funds, a liquidity token version of the asset is minted to the recipient. Users can redeem the liquidity token for the underlying asset once liquidity becomes available. Similarly, the liquidity token will be burned if the user decides to bridge the asset back to the source chain.

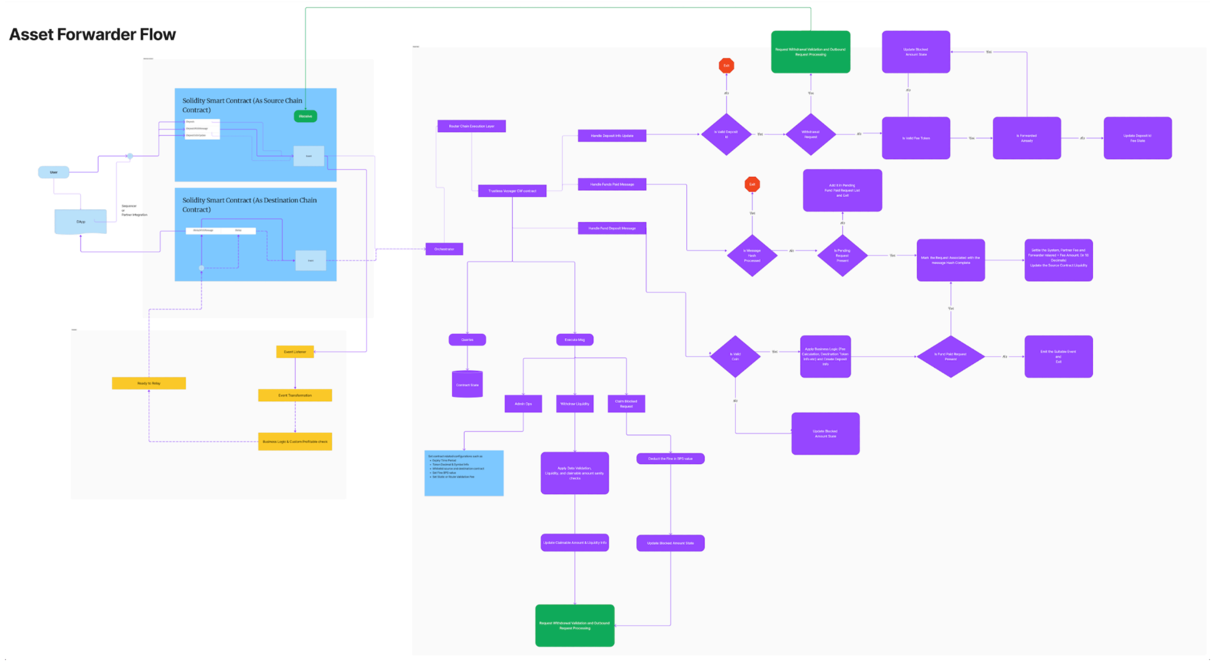
The following findings are from the Router Asset Bridge report, but have been added here since they related to the integration of the various Router components:

No	Description
4	Inability to decode request packet payload due to incorrect indexes results in failed cross-chain requests
6	NEAR tokens are incorrectly transferred to the `gateway` contract instead of being escrowed in the `AssetBridge` contract
7	Fungible token refunds will fail, causing a loss of funds for users
9	Native funds cannot be attached on `ft_on_transfer` calls
11	Failed cross-chain requests cannot be retried
13	The same gas limit is used for all destination chains, resulting in out-of-gas errors
17	Lowercasing case-sensitive addresses causes unexpected behavior and incorrect outbound calls
19	Callback functions may run out of gas, resulting in inconsistent states of the NEAR

	`asset-bridge` contract
26	`AssetBridge` Solidity contract is unable to transfer certain ERC-20 tokens
27	`AssetBridge` Solidity contact is not compatible with ERC-20 tokens that charge a transfer fee

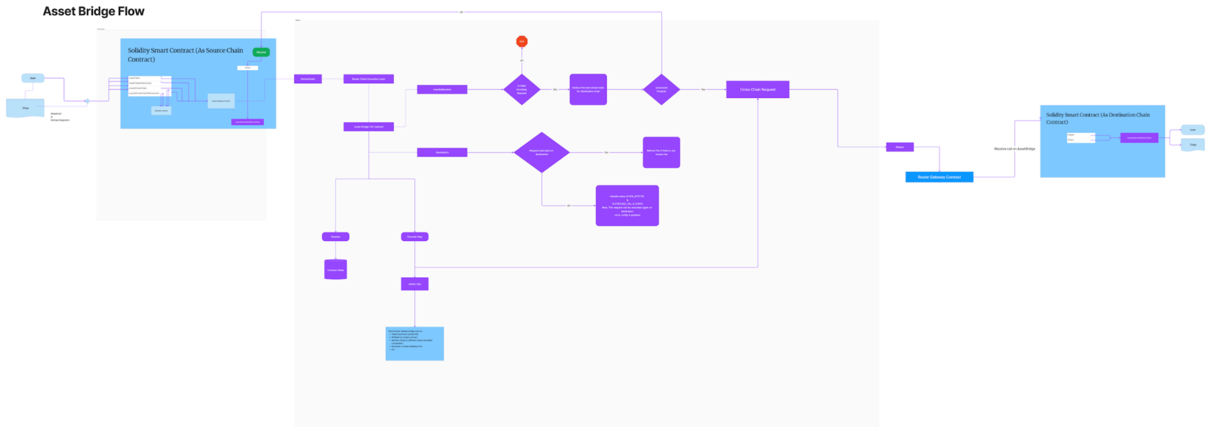
Appendix A: Architecture Diagrams

Asset Forwarder Flow



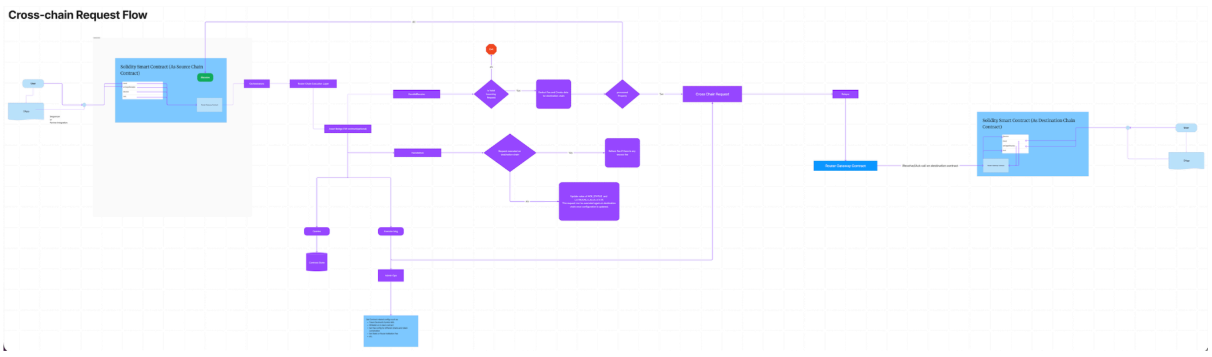
Asset Forwarder Flow
Solidity Smart Contract (As Source Chain Contract)
iReceive
Event
User
DApp
Sequencer or Partner Integration
Solidity Smart Contract (As Destination Chain Contract)
Event
Event Listener
Event Transformation
Business Logic & Custom Profitable check
Ready to Relay
Router Chain Execution Layer
Orchestrator
Trustless Voyager CW contract
Handle Deposit Info Update
Handle Funds Paid Message
Handle Fund Deposit Message
Is Valid Deposit Id
Exit
Withdrawal Request
Request Withdrawal Validation and Outbound Request Processing
Is Valid Fee Token
Update Blocked Amount State
Is Forwarded Already
Update Deposit Id Fee State
Is Message Hash Processed
Is Pending Request Present
Add it in Pending Fund Paid Request List and Exit
Mark the Request Associated with the message Hash Complete
Settle the System, Partner Fee and Forwarder relayed + Fee Amount. (in 18 Decimals) Update the Source Contract Liquidity
Is Valid Coin
Apply Business Logic (Fee Calculation, Destination Token Info etc) and Create Deposit Info
Is Fund Paid Request Present
Emit the Suitable Event and Exit
Update Blocked Amount State
Queries
Contract State
Execute Msg
Admin Ops
Withdraw Liquidity
Claim Blocked Request
Deduct the Fine in BPS value
Apply Data Validation, Liquidity, and claimable amount sanity checks
Update Claimable Amount & Liquidity Info
Update Blocked Amount State
Request Withdrawal Validation and Outbound Request Processing
Yes
No

Asset Bridge Flow



Asset Bridge Flow
Solidity Smart Contract (As Source Chain Contract)
Receive
User
DApp
Sequencer or Partner Integration
Orchestrator
Router Chain Execution Layer
Asset Bridge CW contract
HandleReceive
Is Valid Incoming Request
Exit
Deduct Fee and Create data for destination chain
processed Properly
Cross Chain Request
HandleAck
Request executed on destination
Refund Fee if there is any excess fee
Queries
Contract State
Execute Msg
Admin Ops
Relayer
Router Gateway Contract
Solidity Smart Contract (As Destination Chain Contract)
Yes
No

Cross-chain Request Flow


Cross-chain Request Flow
User
DApp
Sequencer
or
Partner Integration
Solidity Smart Contract (As Source Chain Contract)
Receive
Router Gateway Contract
Orchestrators
Router Chain Execution Layer
Asset Bridge CW contract(optional)
HandleReceive
HandleAck
Is Valid Incoming Request
Exit
No
Yes
Deduct Fee and Create data for destination chain
processed Properly
Yes
No
Cross Chain Request
Request executed on destination chain
Yes
Refund Fee if there is any excess fee
No
Update value of ACK_STATUS and OUTBOUND_CALLS_STATE. This request can be executed again on destination chain once configuration is updated.
Queries
Contract State
Execute Msg
Admin Ops
Relayer
Router Gateway Contract
iReceive/iAck call on destination contract
Solidity Smart Contract (As Destination Chain Contract)
Router Gateway Contract
User
DApp