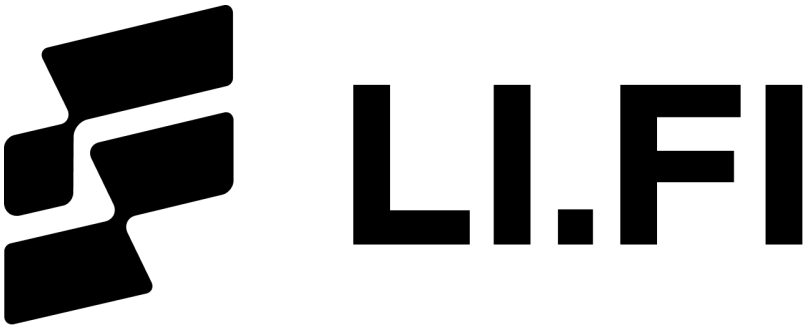

LI.FI Security Review

AcrossFacetPackedV4(v1.0.0), AcrossFacetV4(v1.0.0), ReceiverAcrossV4(v1.0.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

September 1, 2025

Contents

1 About Researcher

Sujith Somraaj is a distinguished security researcher and protocol engineer with over eight years of comprehensive experience in the Web3 ecosystem.

In addition to working as a Security researcher at Spearbit, Sujith is also the security researcher and advisor for leading bridge protocol LI.FI and also is a former founding engineer and current CISO at Superform, a yield aggregator with over $170M in TVL.

Sujith has experience working with protocols / funds including Layerzero, Edge Capital, Berachain, Optimism, Ondo, Sonic, Monad, Blast, ZkSync, Decent, Drips, SuperSushi Samurai, DistrictOne, Omni-X, Centrifuge, Superform-V2, Tea.xyz, Paintswap, Bitcorn, Sweep n' Flip, Byzantine Finance, Variational Finance, Satsbridge, Rova, Horizen, Earthfast and Angles

Learn more about Sujith on sujithsomraaj.xyz or on cantina.xyz

2 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of that given smart contract(s) or blockchain software. i.e., the evaluation result does not guarantee against a hack (or) the non existence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, I always recommend proceeding with several audits and a public bug bounty program to ensure the security of smart contract(s). Lastly, the security audit is not an investment advice.

This review is done independently by the reviewer and is not entitled to any of the security agencies the researcher worked / may work with.

3 Scope

- src/Facets/AcrossFacetV4.sol(v1.0.0), src/Facets/AcrossFacetPackedV4.sol(v1.0.0)
- src/Periphery/ReceiverAcrossV4.sol(v1.0.0)
- src/Interfaces/IAcrossSpokePoolV4.sol(v1.0.0), src/Libraries/LibAsset.sol(v2.1.2)

4 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

4.1 Impact

High leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.

Medium global losses <10% or losses to only a subset of users, but still unacceptable.

Low losses will be annoying but bearable — applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

4.2 Likelihood

High almost certain to happen, easy to perform, or not easy but highly incentivized

Medium only conditionally possible or incentivized, but still relatively likely

Low requires stars to align, or little-to-no incentive

4.3 Action required for severity levels

Critical Must fix as soon as possible (if already deployed)

High Must fix (before deployment if not already deployed)

Medium Should fix

Low Could fix

5 Executive Summary

Over the course of 13 hours in total, LI.FI engaged with the researcher to audit the contracts described in section 3 of this document ("scope").

In this period of time a total of 12 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	650d18e
Audit Timeline	August 21, 2025 - September 1, 2025
Methods	Manual Review
Documentation	High
Test Coverage	Medium-High

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Gas Optimizations	0
Informational	9
Total Issues	**12**

6 Findings

6.1 Low Risk

6.1.1 Lossy decode for non-EVM receiver

Context: AcrossFacetPackedV4.sol#L339, AcrossFacetPackedV4.sol#L374

Description: In both decoders, we set `bridgeData.receiver = address(uint160(uint256(bytes32(...))))`. For non-EVM receivers (e.g., Solana), this truncates to 20 bytes and is not round-trippable.

Recommendation: Continue to set bridgeData.receiver for EVM destination chains, and expose the raw receiver as part of AcrossV4Data or document that callers must read acrossData.receiverAddress for non-EVM flows.

LI.FI: Fixed in 5e797a1

Researcher: The fix works for all canonical EVM addresses (left-padded 12 zero bytes). However, it can misclassify a non-EVM 32-byte value that happens to have 12 leading zeros as EVM (although this is rare but possible). Since the exact receiver is exposed in acrossData.receiverAddress (lossless), this is acceptable.

6.1.2 Add validations for `modifiedAcrossData.outputAmount` post correction

Context: AcrossFacetV4.sol#L133

Description: The output amount calculation uses division without checking for overflow/underflow conditions:

```
modifiedAcrossData.outputAmount =
    (_bridgeData.minAmount * _acrossData.outputAmountMultiplier) /
    1e18;
```

If the `_bridgeData.minAmount * _acrossData.outputAmountMultiplier` is less than `1e18` (or) if the outputAmountMultiplier is passed in as zero, the overall outputAmount will be zero, and an across intent is created with zero output amount, leading to loss of user funds.

Recommendation: Consider validating if outputAmount is not zero post correction:

```
modifiedAcrossData.outputAmount =
    (_bridgeData.minAmount * _acrossData.outputAmountMultiplier) /
    1e18;
+ if (modifiedAcrossData.outputAmount == 0) revert();
```

LI.FI: Acknowledged. Since we (aka our backend) will calculate the outputAmountMultiplier, we trust the calldata. Added a disclaimer comment for full transparency: a265751

Researcher: The additional warning documentation added appears to be satisfactory. Acknowledged.

6.1.3 Ensure `refundAddress` is non-zero

Context: AcrossFacetV4.sol#L145

Description: The refundAddress parameter in the `_acrossData` is passed to the spoke pool without validations. This data is crucial to receive refunds / update the intent order posted to Across.

So, passing zero by mistake or accident could potentially lead to lost user funds if the intent can't be fulfilled.

Recommendation: Consider validating if the refundAddress is non zero as follows:

```
function _startBridge(
 ILiFi.BridgeData memory _bridgeData,
 AcrossV4Data memory _acrossData
) internal {
  ....
+ if(_acrossData.refundAddress == bytes32(0)) revert();
  ....
}
```

LI.FI: Fixed in ed698cc

Researcher: Verified fix.

6.2 Informational

6.2.1 Multiple redundant type casts

Context: AcrossFacetPackedV4.sol#L267-L269, AcrossFacetPackedV4.sol#L272, AcrossFacetPackedV4.sol#L314-L320, AcrossFacetPackedV4.sol#L327, AcrossFacetPackedV4.sol#L163, AcrossFacetPackedV4.sol#L253

Description: Multiple lines in the `AcrossFacetPackedV4` implements redundant typecasting from between same types (bytes32 -> bytes32, bytes8 -> bytes8), which are redundant.

Recommendation: Consider removing the redundant type casts.

LI.FI: Fixed in 85ab7b3

Researcher: Verified fix.

6.2.2 Typo

Context: AcrossFacetPackedV4.sol#L22

Description: There's a typo in the documentation of the `SPOKE_POOL` variable. The documentation states the following:

```
/// @notice The contract address of the cbridge on the source chain.
```

Which is redundant/irrelevant.

Recommendation: Consider fixing the above-mentioned doc issue.

LI.FI: Fixed in b220822

Researcher: Verified fix.

6.2.3 Centralization risk in `AcrossFacetPackedV4`

Context: AcrossFacetPackedV4.sol#L416

Description: The `executeCallAndWithdraw()` function allows the owner to make arbitrary external calls:

```
function executeCallAndWithdraw(
    address _callTo,
    bytes calldata _callData,
    address _assetAddress,
    address _to,
    uint256 _amount
) external onlyOwner {
    (bool success, ) = _callTo.call(_callData);
    // ...
}
```

Since the `_callTo` address is not validated, this function could be weaponized by the owner to cause potential DoS by revoking existing approvals or loss of user funds.

Recommendation: Restrict the **_callTo** address behind a time-locked whitelist, providing time for external integrators to understand the function's behavioral changes.

LI.FI: Acknowledged. We do not add this function to the diamond. Therefore, only the packedFacet as a standalone could be weaponized, and since this contract is used by only a minimal number of partners (and the function is admin-only), we can accept the risk.

Researcher: Acknowledged.

6.2.4 Possible burn to the zero address

Context: ReceiverAcrossV4.sol#L109

Description: If a token swap fails when receiving an across message, the failure is handled and the tokens are returned to the receiver address decoded from that message.

However, the `receiver` is directly taken from the message without validation. If a malformed message or a rare upstream backend bug results in `address(0)`, the fallback will send tokens to 0x0, which is irrecoverable.

Recommendation: Validate `receiver != address(0)` (and optionally assetId != address(0)).

LI.FI: Acknowledged. We rely on the correctness of the data we produce.

Researcher: Acknowledged.

6.2.5 Incorrect calldata documentation in `startBridgeTokensViaAcrossV4NativePacked()` function

Context: AcrossFacetPackedV4.sol#L95-L107

Description: The documentation in `startBridgeTokensViaAcrossV4NativePacked()` does not match the actual implementation:

Documented Layout:

```
[76:108] - sendingAssetId (bytes32)
[108:140] - outputAmount
[140:172] - destinationChainId
[172:204] - exclusiveRelayer
[204:208] - quoteTimestamp
[208:212] - fillDeadline
[212:216] - exclusivityDeadline
[216:] - message
```

Actual Layout:

```
[76:108] - sendingAssetId (WRAPPED_NATIVE hardcoded)
[108:140] - receivingAssetId (outputToken)
[140:172] - outputAmount
[172:176] - destinationChainId (4 bytes, not 32)
[176:208] - exclusiveRelayer
[208:212] - quoteTimestamp
[212:216] - fillDeadline
[216:220] - exclusivityDeadline
[220:] - message
```

Recommendation: Consider fixing the inline-documentation to reflect the exact calldata layout as incorrect documentation could lead to integration errors.

LI.FI: Fixed in 9f8d566.

This finding is more significant than you may have initially noticed. It led me to realize that there is an unnecessary parameter in the PackedParameters struct (sendingAssetId is only needed for ERC20 functions since we use the constant WRAPPED_NATIVE for native functions).

Researcher: Verified fix. Agree, the sendingAssetId parameter is unused/unnecessary.

6.2.6 Incorrect calldata length validation in `decode_startBridgeTokensViaAcrossV4NativePacked()` function

Context: AcrossFacetPackedV4.sol#L325

Description: The function validates the `data.length < 188` but then attempts to read data beyond this range. Specifically, the function reads:

```
data[216:220] for exclusivityDeadline
data[220:] for message
```

This creates a scenario where calldata with length between 188-219 bytes will pass validation but cause out-of-bounds access when reading the exclusivityDeadline field at position 216-220.

Recommendation: Update the validation to ensure the minimum required length covers all field accesses:

```
if (data.length < 220) {  //  Correct validation
    revert InvalidCalldataLength();
}
```

LI.FI: Fixed in 4bce2a1

Researcher: Verified fix.

Note: However, this change has been reverted because the sendingAssetId parameter has been removed in the upcoming PRs.

6.2.7 Outdated `deposit()` function documentation in interface

Context: IAcrossSpokePoolV4.sol#L24

Description: In the last Across Spoke pool contract, the `exclusivityDeadline` is replaced with `exclusivityParameter`, and there exists some documentation mismatch between the Across contract and the LI.FI interface contract.

Recommendation: Consider fixing the documentation mismatch:

```
/**
     * @notice Previously, this function allowed the caller to specify the exclusivityDeadline,
     ↪  otherwise known as the
     * as exact timestamp on the destination chain before which only the exclusiveRelayer could fill the
     ↪  deposit. Now,
     * the caller is expected to pass in a number that will be interpreted either as an offset or a
     ↪  fixed
     * timestamp depending on its value.
     * @notice Request to bridge input token cross chain to a destination chain and receive a specified
     ↪  amount
     * of output tokens. The fee paid to relayers and the system should be captured in the spread
     ↪  between output
     * amount and input amount when adjusted to be denominated in the input token. A relayer on the
     ↪  destination
     * chain will send outputAmount of outputTokens to the recipient and receive inputTokens on a
     ↪  repayment
     * chain of their choice. Therefore, the fee should account for destination fee transaction costs,
     * the relayer's opportunity cost of capital while they wait to be refunded following an optimistic
     ↪  challenge
     * window in the HubPool, and the system fee that they'll be charged.
```

```
* @dev On the destination chain, the hash of the deposit data will be used to uniquely identify
↪  this deposit, so
* modifying any params in it will result in a different hash and a different deposit. The hash will
↪  comprise
* all parameters to this function along with this chain's chainId(). Relayers are only refunded for
↪  filling
* deposits with deposit hashes that map exactly to the one emitted by this contract.
* @param depositor The account credited with the deposit who can request to "speed up" this deposit
↪  by modifying
* the output amount, recipient, and message.
* @param recipient The account receiving funds on the destination chain. Can be an EOA or a
↪  contract. If
* the output token is the wrapped native token for the chain, then the recipient will receive
↪  native token if
* an EOA or wrapped native token if a contract.
* @param inputToken The token pulled from the caller's account and locked into this contract to
* initiate the deposit. The equivalent of this token on the relayer's repayment chain of choice
↪  will be sent
* as a refund. If this is equal to the wrapped native token then the caller can optionally pass in
↪  native token as
* msg.value, as long as msg.value = inputTokenAmount.
* @param outputToken The token that the relayer will send to the recipient on the destination
↪  chain. Must be an
* ERC20.
* @param inputAmount The amount of input tokens to pull from the caller's account and lock into
↪  this contract.
* This amount will be sent to the relayer on their repayment chain of choice as a refund following
↪  an optimistic
* challenge window in the HubPool, less a system fee.
* @param outputAmount The amount of output tokens that the relayer will send to the recipient on
↪  the destination.
* @param destinationChainId The destination chain identifier. Must be enabled along with the input
↪  token
* as a valid deposit route from this spoke pool or this transaction will revert.
* @param exclusiveRelayer The relayer that will be exclusively allowed to fill this deposit before
↪  the
* exclusivity deadline timestamp. This must be a valid, non-zero address if the exclusivity
↪  deadline is
* greater than the current block.timestamp. If the exclusivity deadline is < currentTime, then this
↪  must be
* address(0), and vice versa if this is address(0).
* @param quoteTimestamp The HubPool timestamp that is used to determine the system fee paid by the
↪  depositor.
*  This must be set to some time between [currentTime - depositQuoteTimeBuffer, currentTime]
* where currentTime is block.timestamp on this chain or this transaction will revert.
* @param fillDeadline The deadline for the relayer to fill the deposit. After this destination
↪  chain timestamp,
* the fill will revert on the destination chain. Must be set before currentTime +
↪  fillDeadlineBuffer, where
* currentTime is block.timestamp on this chain or this transaction will revert.
* @param exclusivityParameter This value is used to set the exclusivity deadline timestamp in the
↪  emitted deposit
* event. Before this destination chain timestamp, only the exclusiveRelayer (if set to a non-zero
↪  address),
* can fill this deposit. There are three ways to use this parameter:
*     1. NO EXCLUSIVITY: If this value is set to 0, then a timestamp of 0 will be emitted,
*        meaning that there is no exclusivity period.
*     2. OFFSET: If this value is less than MAX_EXCLUSIVITY_PERIOD_SECONDS, then add this value to
*        the block.timestamp to derive the exclusive relayer deadline. Note that using the
↪  parameter in this way
*        will expose the filler of the deposit to the risk that the block.timestamp of this event
↪  gets changed
```

```
     *        due to a chain-reorg, which would also change the exclusivity timestamp.
     *     3. TIMESTAMP: Otherwise, set this value as the exclusivity deadline timestamp.
     * which is the deadline for the exclusiveRelayer to fill the deposit.
     * @param message The message to send to the recipient on the destination chain if the recipient is
     ↪  a contract.
     * If the message is not empty, the recipient contract must implement handleV3AcrossMessage() or the
     ↪  fill will revert.
     */
    function deposit(
        bytes32 depositor,
        bytes32 recipient,
        bytes32 inputToken,
        bytes32 outputToken,
        uint256 inputAmount,
        uint256 outputAmount,
        uint256 destinationChainId,
        bytes32 exclusiveRelayer,
        uint32 quoteTimestamp,
        uint32 fillDeadline,
        uint32 exclusivityParameter,
        bytes calldata message
    ) public payable;
```

LI.FI: Updated the parameter descriptions in the facet, test files, demoscript, docs, etc in 0d0c7f

Researcher: Verified fix.

6.2.8 Replace `1e18` with constant

Context: AcrossFacetV4.sol#L135

Description: The division by `1e18` is a magic number in the correction of outputAmount post swap. This could be declared as a constant with a more meaningful name.

Recommendation: Consider updating the magic number with a constant as follows:

```
+ uint256 CORRECTION_CONSTANT = 1e18;

  modifiedAcrossData.outputAmount =
     (_bridgeData.minAmount * _acrossData.outputAmountMultiplier) /
-    1e18
+    CORRECTION_CONSTANT;
```

LI.FI: Fixed in 0d19015

Researcher: Verified fix.

6.2.9 Inaccurate multiplier calculation documentation

Context: AcrossFacetV4.sol#L131

Description: The inline documentation for the multiplier calculation in `AcrossFacetV4` ignores the multiplier percentage, leading to confusion.

The documentation currently states that:

```
// The multiplier should be calculated as: 1e18 * 10^(outputDecimals - inputDecimals)
```

However, the right calculation should be:

```
// The multiplier should be calculated as:  multiplierPercentage * 1e18 * 10^(outputDecimals -
↪  inputDecimals)
```

For example, if we are swapping 5000 USDC to 1 WETH and bridging 1 WETH for 5000 USDT, the correct multiplier will be 5000e6.

Recommendation: Consider updating the documentation to reflect the exact behavior of the multiplier.

LI.FI: Fixed in f6fee27

Researcher: Verified fix.

7 Additional Comments

The initial review was conducted on commit 50dd74f. During the review process, the LI.FI team identified a critical issue with the use of a bytes8 destination chain ID, which was insufficient to represent the uint64-sized Solana chain ID required by Across. To address this limitation, the audit was subsequently extended to include a later implementation, with the final review performed on commit d1d6d6 of the AcrossPackedFacetV4 contract.