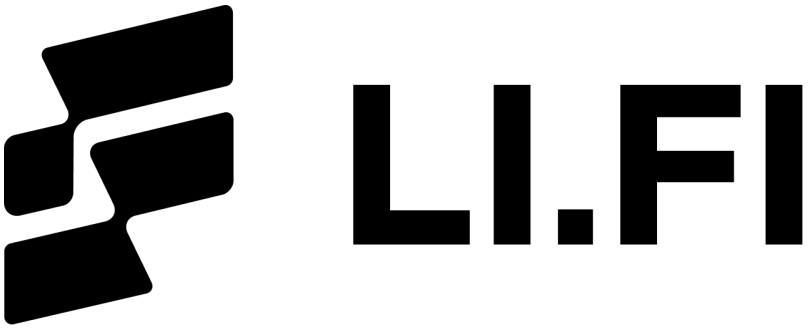

LI.FI Security Review

LiFiDEXAggregator(v1.9.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

May 12, 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 including Berachain, Optimism, Fantom, 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, 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/Periphery/LiFiDEXAggregator.sol(v1.9.0)
- src/Interfaces/IAlgebraRouter.sol(v1.0.0)
- src/Interfaces/IAlgebraQuoter.sol(v1.0.0)
- src/Interfaces/IAlgebraPool.sol(v1.0.0)
- src/Interfaces/IAlgebraFactory.sol(v1.0.0)

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 4 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 4 issues were found. This review focussed only on the changes made from the previous version, not the code on its entirety.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	eb255e6240.....677284681
Audit Timeline	May 10, 2025
Methods	Manual Review
Documentation	Medium
Test Coverage	Medium

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	1
Gas Optimizations	0
Informational	3
Total Issues	**4**

6 Findings

6.1 Low Risk

6.1.1 Remove try-catch pattern in `swapAlgebra()` function

Context: LiFiDEXAggregator.sol#L889-L909

Description: The current implementation of `swapAlgebra()` uses a try-catch pattern to handle the `swapSupportingFeeOnInputTokens` call. This approach introduces unnecessary complexity and potential security risks.

The code assumes failure is only due to the method not existing. But multiple other failing scenarios exist, including: Insufficient allowance, Invalid pool state, Slippage protection triggered, Out of gas, Access control restrictions, Arithmetic overflows, and Invalid parameters.

In these scenarios, using the `swap()` for fee-on-transfer tokens could be logically incorrect and lead to unexpected protocol behavior.

Recommendation: Remove the try-catch pattern and let the function revert naturally if `swapSupportingFeeOnInputTokens` fails. This aligns with other swap implementations and provides better error handling.

LI.FI: Fixed in 6da37c48d1f521395c41148a9b38651858ba9812

Researcher: Verified fix

6.2 Informational

6.2.1 Inaccurate `supportsFeeOnTransfer` flag validation

Context: LiFiDEXAggregator.sol#L873

Description: The `swapAlgebra()` function mistakenly checks the `supportsFeeOnTransfer` flag from the input stream. It uses `stream.readUint8() > 0` to decide if `supportsFeeOnTransfer` should be executed.

As a result, this implementation allows any non-zero value (1-255) to enter the supportsFeeOnTransfer route, while the documentation specifies that only the value `1` should activate the supportsFeeOnTransfer route.

Recommendation: Replace the loose comparison with a strict equality check:

```
uint8 constant FEE_ON_TRANSFER_FLAG = 1;

function swapAlgebra(
  uint256 stream,
  address from,
  address tokenIn,
  uint256 amountIn
) private {
  /// ...
  bool supportsFeeOnTransfer = stream.readUint8() == FEE_ON_TRANSFER_FLAG;
  /// ...
}
```

Alternatively, update the documentation in the tests and clearly state that any flag value > 0 will enter the fee on transfer tokens swap route.

LI.FI: Fixed in 401b4a62bbdfd77d63c4e102952b0aadbad10d74

Researcher: Verified fix

6.2.2 Inconsistent direction handling

Context: LiFiDEXAggregator.sol#L871

Description: The `swapAlgebra()` function uses `bool direction = stream.readUint8() > 0` instead of `DIRECTION_TOKEN0_TO_TOKEN1` constant to determine the swap direction, exhibiting inconsistency.

Recommendation: Consider updating the direction check as follows:

```
function swapAlgebra(
   uint256 stream,
   address from,
   address tokenIn,
   uint256 amountIn
) private {
   ....
  // direction indicates the swap direction: true for token0 -> token1, false for token1 -> token0
  bool direction = stream.readUint8()  == DIRECTION_TOKEN0_TO_TOKEN1;
  ....
}
```

LI.FI: Fixed in 9c6c695972f1939fb6e875cb35b2be0ca5b85728

Researcher: Verified fix

6.2.3 Sanity check `recipient` and `pool` address in `swapAlgebra()` function

Context: LiFiDEXAggregator.sol#L864

Description: The function `swapAlgebra()` decodes the pool and recipient addresses from the `stream` parameter. These values are not sanity checked like in other swap functions, and create an inconsistency.

Recommendation: Consider validating input parameters as follows:

```
function swapAlgebra(
  uint256 stream,
  address from,
  address tokenIn,
  uint256 amountIn
  ) private {
    address pool = stream.readAddress();
    bool direction = stream.readUint8() > 0;
    address recipient = stream.readAddress();
    bool supportsFeeOnTransfer = stream.readUint8() > 0;

   // validating parameters
   if(pool == address(0) || pool == IMPOSSIBLE_POOL_ADDRESS || recipient == address(0)) revert
   ↪  InvalidCallData();

   // ...
}
```

LI.FI: Fixed in a18db220dc3fcb3484962d7552600286184c5800

Researcher: Verified fix