



LI.FI Security Review

PolymerCCTPFacet.sol(v1.0.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

December 1, 2025

Contents

1	About Researcher	2
2	Disclaimer	2
3	Scope	2
4	Risk classification	2
4.1	Impact	2
4.2	Likelihood	3
4.3	Action required for severity levels	3
5	Executive Summary	3
6	Findings	4
6.1	Informational	4
6.1.1	Missing docstrings	4
6.1.2	Missing validation of final swap output token in swapAndStartBridgeTokensViaPolymerCCTP	4
6.1.3	Magic value bytes32(0) used for destinationCaller parameter	4
6.1.4	Insufficient validation of bridgeAmount can lead to underflow revert or zero-amount bridging	5
6.1.5	Redundant virtual keyword in initPolymerCCTP() function	5
6.1.6	Testnet chain ids present in mapping function	5

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 Lead 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 security advisor at Superform, a yield aggregator with over \$170M in TVL.

Sujith has experience working with protocols / funds including Coinbase, Uniswap, 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/PolymerCCTPFacet.sol(v1.0.0)
- src/Interfaces/ITokenMessenger.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 12 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 6 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	605f7e14af
Audit Timeline	November 19, 2025 - November 21, 2025
Methods	Manual Review
Documentation	High
Test Coverage	High

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Gas Optimizations	0
Informational	6
Total Issues	6

6 Findings

6.1 Informational

6.1.1 Missing docstrings

Context: [PolymerCCTPFacet.sol#L45](#)

Description: The PolymerCCTPFacet.sol contract contains several declarations that lack proper NatSpec documentation, which reduces code maintainability and makes it harder for developers and auditors to understand the contract's functionality.

The following elements are missing documentation:

1. Event PolymerCCTPFeeSent
2. Immutable State Variables: TOKEN_MESSENGER, USDC, POLYMER_FEE_RECEIVER

Recommendation: Add comprehensive NatSpec documentation for all identified elements.

LI.FI: Fixed in [6d9361d](#) and [26a90ca](#)

Researcher: Verified fix.

6.1.2 Missing validation of final swap output token in `swapAndStartBridgeTokensViaPolymerCCTP`

Context: [PolymerCCTPFacet.sol#L148](#)

Description: The `swapAndStartBridgeTokensViaPolymerCCTP()` function allows users to perform token swaps before bridging via PolymerCCTP. However, there is no validation that the final swap in the `_swapData` array actually outputs USDC, which is the only token supported by this facet.

Recommendation: Add explicit validation to ensure the final swap outputs USDC:

```
if (_swapData[_swapData.length - 1].receivingAssetId != USDC) revert InvalidReceivingToken();
```

LI.FI: Acknowledged (no fix). If the swapdata does not output USDC then the transaction will simply fail since the diamond will not have the required funds to send the transaction. We also ensure that the bridgeData has USDC as sendingAsset ID through `onlyAllowSourceToken` modifier

Researcher: Acknowledged.

6.1.3 Magic value `bytes32(0)` used for `destinationCaller` parameter

Context: [PolymerCCTPFacet.sol#L186](#), [PolymerCCTPFacet.sol#L201](#)

Description: In the `PolymerCCTPFacet._startBridge()` function, the `depositForBurn()` method is called twice with a hardcoded magic value `bytes32(0)` as the `destinationCaller` parameter.

The value `bytes32(0)` has a specific semantic meaning in the CCTP protocol: it indicates that any address can complete the attestation and minting process on the destination chain, rather than restricting it to a specific caller.

Recommendation: Replace the magic value with a well-named constant to improve code readability:

```

/// @dev bytes32(0) allows any address to complete the CCTP transfer on destination chain
bytes32 private constant DESTINATION_CALLER = bytes32(0);

TOKEN_MESSENGER.depositForBurn(
    bridgeAmount,
    _chainIdToDomainId(_bridgeData.destinationChainId),
    _polymerData.nonEVMReceiver,
    USDC,
    DESTINATION_CALLER,
    _polymerData.maxCCTPFee,
    _polymerData.minFinalityThreshold
);

```

LI.FI: Fixed in [6d9361d](#)

Researcher: Verified fix.

6.1.4 Insufficient validation of `bridgeAmount` can lead to underflow revert or zero-amount bridging

Context: [PolymerCCTPFacet.sol#L171-L172](#)

Description: In the `_startBridge()` function, the `bridgeAmount` is calculated by subtracting `polymerTokenFee` from `_bridgeData.minAmount`:

```
uint256 bridgeAmount = _bridgeData.minAmount - _polymerData.polymerTokenFee;
```

If `_polymerData.polymerTokenFee >= _bridgeData.minAmount`, the subtraction will underflow. If `_polymerData.polymerTokenFee == _bridgeData.minAmount`, the calculation results in `bridgeAmount = 0`, which initiates a redundant bridging.

Recommendation: Add explicit validation in the `_startBridge()` function before the subtraction:

```
if (_bridgeData.minAmount <= _polymerData.polymerTokenFee) revert InvalidAmount();
```

LI.FI: Acknowledged, will not fix to save gas. Added more documentation in [e31eef9](#)

Researcher: Added documentation looks good.

6.1.5 Redundant `virtual` keyword in `initPolymerCCTP()` function

Context: [PolymerCCTPFacet.sol#L96](#)

Description: The `initPolymerCCTP()` function in `PolymerCCTPFacet.sol` contains the **virtual** keyword in its function signature, which is redundant.

Recommendation: Remove the **virtual** keyword from the `initPolymerCCTP()` function signature.

LI.FI: Fixed in [5b69e49](#)

Researcher: Verified fix.

6.1.6 Testnet chain ids present in mapping function

Context: [PolymerCCTPFacet.sol#L298-L308](#)

Description: The `_chainIdToDomainId()` function in `PolymerCCTPFacet.sol` contains mappings for testnet chain IDs that resolve to the duplicate CCTP domain IDs as their mainnet counterparts:

- Chain ID 11155111 (Sepolia testnet) → Domain ID 0 (Ethereum mainnet)
- Chain ID 11155420 (OP Sepolia testnet) → Domain ID 2 (OP Mainnet)
- Chain ID 84532 (Base Sepolia testnet) → Domain ID 6 (Base mainnet)

Suppose a user mistakenly provides a testnet chain ID as the destinationChainId parameter in a mainnet transaction. In that case, the contract will accept it and bridge USDC to the corresponding mainnet domain instead of reverting.

While funds are not at risk (they still reach a valid mainnet destination), this creates operational confusion and pollutes event logs with semantically incorrect data.

Recommendation: Remove the testnet chain ID mappings from the production `_chainIdToDomainId()` function.

LI.FI: Fixed in [15bca15](#)

Researcher: Verified fix.