OpenZeppelin | security

1inch Cross-Chain Swap V1.1.0 Audit

August 21, 2025

Table of Contents

Summary

Type	Cross-Chain
Timeline	From 2025-08-07 To 2025-08-11
Languages	Solidity

Total Issues	4 (3 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	1 (1 resolved)
Notes & Additional Information	3 (2 resolved)

Scope

OpenZeppelin conducted a differential audit of the 1inch/cross-chain-swap repository at commit ac88553 against commit d0a59ab. The full list of changes can be found here.

In scope were the following files:

```
contracts/
├── BaseEscrow.sol
├── BaseEscrowFactory.sol
├── Escrow.sol
├── EscrowDst.sol
├── EscrowFactory.sol
├── EscrowSrc.sol
├── interfaces/
│   ├── IBaseEscrow.sol
│   └── IEscrowFactory.sol
├── libraries/
│   └── ImmutablesLib.sol
└── zkSync/
    ├── EscrowDstZkSync.sol
    ├── EscrowFactoryZkSync.sol
    ├── EscrowSrcZkSync.sol
    └── EscrowZkSync.sol
```

System Overview

This audit reviews the changes introduced between version 1 and version 1.1 of the cross-chain swap protocol. The contracts have already been independently audited in previous engagements, and readers are encouraged to consult those reports for in-depth details on their internal logic and security properties.

The main functional change in the new version is the adjustment made to the fee collection mechanism for makers. In the original design, maker fees were deducted at the order fill stage on the source chain. In the updated implementation, these fees are instead charged when the maker withdraws their funds on the destination chain. This moves the point of fee deduction closer to the actual delivery of the swapped assets.

In the previous version, the protocol relied on `BaseExtension.sol` and `ResolverValidationExtension.sol`. These contracts have now been restructured and replaced by `FeeTaker.sol` and `SimpleSettlement.sol`, respectively, in the current commit. Both of these contracts have undergone audits, and readers should refer to those reviews for more granular security analysis.

Previously, maker fees were charged on the source chain at fill time via the `FeeBankCharger` contract. Under the new architecture, this process has been removed in favor of deducting fees on the destination chain during withdrawal. Specifically, fees are charged in the `_withdraw` function, which transfers the net amount to the maker after the appropriate fee amount has been calculated, deducted, and distributed.

Aside from the changes mentioned above, some minor changes were also made to various validation and helper functions and modifiers. For example, the `onlyValidImmutables` function has been changed to accept the hash of the immutables rather than an `Immutables` struct, although the underlying functionality essentially remains the same. These small changes make the contracts in scope more extensible and more gas efficient without impacting the functionality much.

Security Model and Trust Assumptions

The trust assumptions and security model established in the previous audits remain applicable in the updated version. All prior guarantees regarding the correctness, permissions, and operational safety of the system must still hold for the protocol to function securely. This includes the security of the underlying audited contracts, the correctness of cross-chain message handling, and the integrity of off-chain actors involved in settlement. Any change in these foundational assumptions could impact the overall security posture of the system.

Below are some additional trust assumptions that need to be upheld to ensure the security of the protocol:

- Since the protocol and integrator fee amounts, as well as recipient addresses, are now included in `DstImmutablesComplement`, the resolver (`taker`) responsible for creating the destination escrow must ensure that these values are correctly propagated from the source chain output. Any deviation could alter the intended payout.
- The relayer, operated by 1inch, is responsible for sharing the hashlock secret with resolvers. As in the previous versions, resolvers must verify the correctness of the destination escrow's immutable parameters before relying on it, including the hashlock, token, and amounts. With the new changes, this verification must also cover the fee amounts and recipient addresses defined in `DstImmutablesComplement`.
- The addresses of the newly deployed factories must be communicated to makers so they can include the correct factory address when creating new orders. This ensures that orders reference the intended, up-to-date contract and can be processed correctly.
- It is assumed that the underlying tokens revert or fail when a transfer is attempted without sufficient balance in the sender's account, preventing silent failures or partial transfers that could disrupt escrow funding and withdrawal flows. Since the protocol accepts tokens that comply with an ERC-20 interface, users should always be aware of the tokens they are interacting with and understand that maliciously designed tokens may not behave as expected.
- It is assumed that out-of-scope integration contracts (`SimpleSettlement` and `FeeTaker`) function as documented in place of the previously used contracts.

Low Severity

L-01 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `ImmutablesLib.sol`, the `protocolFeeAmountCd` function
- In `ImmutablesLib.sol`, the `integratorFeeAmountCd` function
- In `ImmutablesLib.sol`, the `protocolFeeRecipientCd` function
- In `ImmutablesLib.sol`, the `integratorFeeRecipientCd` function

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #138 at commit be18321.*

Notes & Additional Information

N-01 Custom Errors in `require` Statements

Solidity version `0.8.26` added custom error support to `require` statements. Initially, this feature was only available through the IR pipeline. However, Solidity version `0.8.27` extended support to the legacy pipeline as well.

`ImmutablesLib.sol` contains multiple instances of `if-revert` statements that could be replaced with `require` statements:

- The `if (parameters.length < 0x20) revert IndexOutOfRange()` statement
- The `if (parameters.length < 0x40) revert IndexOutOfRange()` statement
- The `if (parameters.length < 0x60) revert IndexOutOfRange()` statement
- The `if (parameters.length < 0x80) revert IndexOutOfRange()` statement

- The `if (parameters.length < 0x20) revert IndexOutOfRange()` statement
- The `if (parameters.length < 0x40) revert IndexOutOfRange()` statement
- The `if (parameters.length < 0x60) revert IndexOutOfRange()` statement
- The `if (parameters.length < 0x80) revert IndexOutOfRange()` statement.

For conciseness and gas savings, consider replacing `if-revert` statements with `require` statements.

Update: *Acknowledged, not resolved. The 1inch team stated:*

> *We didn't make changes for "N-01 Custom Errors in require Statements" because our project uses Solidity 0.8.23 and we haven't got plans to upgrade it to 0.8.27.*

N-02 Modifier Naming Improvement Suggestion

After the changes, the `onlyTaker` modifier name has become misleading, as it accepts an address parameter and checks it against `msg.sender`. If the provided address is not the taker, the modifier will not specifically enforce a taker check.

Consider renaming the modifier to something more descriptive that accurately reflects its actual behavior.

Update: *Resolved in pull request #138 at commit be18321.*

N-03 Fee Amount May Exceed Taking Amount

While the protocol fees and integration fees are set by the maker at order creation, the current implementation allows the sum of `integratorFeeAmount + protocolFeeAmount` to exceed the `takingAmount`. This can result in orders where the entire taker amount is consumed by fees, leaving no meaningful payout. This can also lead to a confusing failure in the `withdraw` function call.

Consider adding a validation check in the `_postInteraction` function to ensure that the total fee amount is strictly less than `takingAmount`. This would prevent the creation of orders that are non-functional or misleading to other participants.

Update: *Resolved in pull request #138 at commit be18321.*

Conclusion

OpenZeppelin has reviewed the changes introduced in this update to the cross-chain swap protocol. Our analysis found the code to be functional and consistent with the intended design. No critical-, high-, or medium-severity issues were identified during the course of this review.

The 1inch team is appreciated for their responsiveness and cooperation throughout the audit process.