certora

Security Assessment
Final Report

1inch

Cross Chain Swap

August 2025

Prepared for 1inch

Table of Contents

Project Summary

Project Scope

Project Name	Repository (link)	First Commit Hash	Platform
Cross Chain Swap	https://github.com/1inch/cross-chain-swap	d0a59ab	EVM

Project Overview

This document describes the specification and verification of **Cross Chain Swap** using the Certora Prover. The work was undertaken from **August 25, 2025** to **September 3, 2025**

The following contract list is included in our scope:

{
    contracts/* (excluding contracts/mocks)
}

The Certora Prover demonstrated that the implementation of the **Solidity** contracts above is correct with respect to the formal rules written by the Certora team. During the verification process, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Protocol Overview

The cross-chain swap allows executing an atomic swap across two different chains without relying on a bridge.
This is done by deploying escrows on both chains, where funds would be sent to the destinations only when the secret to the escrow's hashlock is revealed.
This ensures withdrawal is atomic; every party is assured that the other party can't withdraw unless the first party withdraws as well.

The execution of the txs on chain is done by the taker and resolvers, where the safety deposit (deposited by the taker) is used as an incentive to execute those txs. The taker gets exclusivity first on executing the tx and getting his safety deposit back, and only if he fails to do that, then the rest of the resolves get a chance to execute the tx and get the safety deposit as a reward.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	3	3	0
Informational	1	1	0
Total	4	4	0

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
L-01	L-01 Makers can steal taker's funds by releasing the secret in the last second	Low	Acknowledged
L-02	L-02 Makers can prevent the taker from retrieving their safety deposit on the destination chain	Low	Acknowledged (would address in future versions)
L-03	L-03 Taker can steal funds from the maker by using the fee recipients to revert withdrawal on dest chain	Low	Acknowledged

Low Severity Issues

L-01 Makers can steal taker's funds by releasing the secret in the last second

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: BaseEscrowFactory.sol	Status: Acknowledged	

Description:

When a taker fills a cross chain swap, they must deploy an escrow on the destination chain to deliver the tokens to the maker.

Upon the deployment, `BaseEscrowFactory::createDstEscrow()` checks that the destination cancellation timestamp is no greater than the source cancellation timestamp:

```
JavaScript
        // Check that the escrow cancellation will start not later than the cancellation time
on the source chain.
         if (immutables.timelocks.get(TimelocksLib.Stage.DstCancellation) >
srcCancellationTimestamp) revert InvalidCreationTime();
```

This check protects the taker, as it prevents a scenario where the maker would be able to cancel the swap on the source chain (getting their tokens back) AND withdraw the tokens on the destination chain.

This is a critical protection for the taker, since it is the maker who holds the secret to unlock the withdrawal - and thus could only release the secret once the cancellation timestamp has already been reached on the source chain, but not on the destination chain.

However, the current implementation does not properly protect the taker; it's still possible to have a withdrawal on the dest chain and a cancellation on the dest chain when `DstCancellation == srcCancellationTimestamp` due to a few reasons:

- Block stuffing on the source chain
- Longer time to include tx on the source chain
- Block time difference between chains
    - E.g., Mainnet (as src) has 12 seconds block time while Optimism (as dest) has 2 seconds. So if cancellation timestamp on both chains is xx100 and Mainnet's last block is xx88, while OP's last block is xx98 then there's a 10 seconds gap between the actual cancellation timestamp

Exploit Scenario:

- Taker fills a swap where `DstCancellation == srcCancellationTimestamp`
- Maker waits until the last second to release the secret
- A resolver calls `EscrowDst::publicWithdraw()`, which sends the destination tokens to the maker
- The original taker tries to call `EscrowSrc::withdraw()`, but it reverts with `InvalidTime()` error due to the cancellation timestamp being reached
- A resolver calls `EscrowSrc::publicCancel()`, which sends the source tokens to the maker

Recommendations:

Ensure the dest cancellation timestamp is greater than src cancellation timestamp by some gap. This can be done by modifying the check above during dest escrow deployment. Or can be done off-chain by the taker modifying the `srcCancellationTimestamp` to account also for the gap (making it lower than the actual cancellation timestamp).

Alternatively, given that only the resolvers can run `publicWithdraw()` on the dest chain, if we ensure that they don't run it if we're too close to the source cancellation timestamp, this can also be a reasonable mitigation for this issue.

Customer's response:

This scenario is impossible due to the backend implementation. The Fusion+ backend strictly enforces that the secret can only be revealed if both src and dst escrows exist, the finality lock has passed, and we are within the private withdrawal period on the source chain. Any attempt to reveal the secret in the last block before cancellation would be rejected by our off-chain validation layer. Fusion+ also provides a sufficient buffer to safely deploy the destination escrow, execute withdrawals, or cancel the swap, with additional safeguards implemented on both the frontend and backend. Therefore, a maker cannot exploit this behavior to steal funds.

L-02 Makers can prevent the taker from retrieving their safety deposit on the destination chain		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: EscrowDst.sol	Status: Acknowledged	

Description:

Upon calling `withdraw()` or `publicWithdraw()` on the destination chain, the `_withdraw()` internal function is executed, transferring the swapped tokens to the maker and the safety deposit to the caller.

Normally, the caller will be the original taker, who has a privileged period where only they can call `withdraw()`, which will allow them to retrieve their safety deposit.

However, if the receiving token is the native token, transferring it to the maker will give them control over the execution, allowing them to revert at will.

So a malicious maker can prevent takers from retrieving their safety deposit by reverting upon the receipt of native tokens.

Alternatively, the maker can release the secret only after the privileged `withdraw()` period has passed, such that another resolver calls `publicWithdraw()`.

Exploit Scenario:

- Write a malicious contract that only reverts if the caller of the function is the original taker
- Precompute the address of this contract, such that the taker cannot know beforehand that the transfer will fail
- Deploy the contract after the taker has already created the escrow on the destination chain

Alternatively to this, the maker can release the secret after the privileged withdrawal period, such that another resolver will call `publicWithdraw()` and take the original taker's safety deposit.

Recommendations:

Send native tokens on a try/catch block, making them available to the maker if the transfer fails.

Alternatively, wrap native tokens before transferring.

Additionally, making the secret available first to the original taker and then to all other resolvers.

Customer's response:

Given that only the resolvers can execute the withdrawal and benefit from this, and given that all resolvers are KYC'd - this isn't very likely to happen. Action would be taken against the resolver in such case, and they would have more to lose than to profit from this.

Given that we won't make any changes for the current version, however in the next protocol release we will address it with design-level improvements.

L-03 Taker can steal funds from the maker by using the fee recipients to revert withdrawal on dest chain

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: EscrowDst.sol	Status: Acknowledged	

Description: During withdrawal on the destination chain fees are sent to the integrator and protocol recipients. The sending of the fees might revert, reverting the entire withdrawal tx.
A malicious taker might use that to steal funds from the user, by supplying the maker with fee recipients that would revert - blacklisted addresses for ERC20 transfer or contracts that revert on ETH transfer.

Exploit Scenario:

- An integrator's service gets exposed by an attacker
- The attacker injects a blacklisted address as the integrator fee recipient, and adds themselves to the whitelist of takers
- User inspects the message, but given that nothing looks malicious, they sign it
- Attacker becomes the taker of the order, deploying source and destination escrows
- User releases the secret as nothing looks suspicious yet
- Taker executes withdraw on the source chain
- Given that withdraw on dest chain reverts, the taker waits till cancellation and calls cancellation
- Taker got both ends of the swap, stealing funds from the maker

Recommendations:

- Don't revert if the call to send fee reverts
    - Ensure enough gas is supplied to the call, to prevent the caller from reverting it due to low gas

- Ensure the call to send ETH is gas-limited to prevent it from draining the entire gas
- Alternatively, this can be mitigated off-chain by publishing a list of authorized fee recipients and warning the users to always verify that the fee recipients are from that list.

Customer's response:

Fee recipients are either integrators or protocol-controlled addresses. All integrators undergo KYC/KYB and must explicitly request access to the fees feature. Only approved integrators can set an integratorFeeRecipient. If an integrator deploys malicious code (e.g., a reverting receive()), they would primarily harm their own users. In such cases, we would immediately revoke access and blacklist the integrator. The protocol fee address is controlled and trusted.

Certora's response:

That makes sense, however it's best if users can double check and verify the fee addresses before signing the order, to protect against a case where the integrator service was hacked and fee address was replaced with a malicious one.

Informational Issues

I-01. The escrow on the destination chain rely entirely on offchain validation

Description:
When a taker fills an order, an escrow is created in the source chain, which emits an event with an `immutables` and an `immutablesComplement` structs.

Then, the taker must deploy an escrow on the destination chain with an `immutables` argument that mixes params from both `immutables` and `immutablesComplement`.

The only validation that the destination chain `immutables` matches the source chain structs is that the hashlock must be the same, such that the secret unlocks both escrows.

However, even if the other params are manipulated, the secret would still unlock the escrow, as the secret is only tied to the hashlock, and not any of the other params.

The validation that prevents the taker from manipulating dst `immutables` params to steal funds from either the maker or the fee receiver is performed entirely off-chain.

Therefore, to ensure a safe operation, the off-chain code must perform the following validations:

- Dst `immutables` has the same following values as in src `immutables`:
  - `orderHash`
  - `hashlock`
  - `taker`
- Dst `immutables` has the same following values as in src `immutablesComplement`:
  - `maker`
  - `token`
  - `amount`
  - `safetyDeposit`
  - `params`
- Both immutables use the same raw `timelocks`
- Dst escrow should be deployed after src escrow

Recommendation:

Validate the parameters as listed above.

Customer's response:
This is a design limitation, which we compensate for with strict backend controls and additional verification in our services.
The addresses of the src and dst escrow contracts are calculated deterministically as `keccak256(0xff + escrowFactoryAddress + salt + keccak256(escrow_[src/dst]_initialisation_code))[12:]`,
where salt is the hash of the escrow immutables (maker/recipient, taker, tokens, amounts, timelocks, hashlock, order hash, safety deposits, and fee parameters).

Fusion+ enforces comprehensive backend checks. If any of these conditions are not met, the service will not allow the maker to reveal the secret. These checks include, but are not limited to:

- verifying that for a specific hashlock both dst and src escrows are deployed at the predefined addresses derived from the original immutables,
- ensuring that the finality lock has passed,
- verifying that we are within the private withdrawal period.

Even if a taker attempts to manipulate the dst escrow address by changing immutable values, the maker will not be able to reveal the secret

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.