Smart Contract Security Audit Report

1inch Solana Crosschain Update Audit

1. Contents

2. General Information

This report contains information about the results of the security audit of the 1inch (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 2025-07-18 to 2025-07-24.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: solana-crosschain-protocol. Initial review was done for the commit f9268b6 and the re-testing was done for the commit b2124f6.

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- Maker,
- Taker.

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result if this work we have discovered critical, high, medium and low security issues.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of August 19, 2025.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Mint account is not validated	cross-chain-escrow-src/src/lib.rs, cross-chain-escrow-dst/src/lib.rs	**Critical**	Fixed
Insufficient destination rescue access control.	cross-chain-escrow-dst/src/lib.rs	**High**	Fixed
Rescue start could be earlier than cancellation start	cross-chain-escrow-dst/src/lib.rs, cross-chain-escrow-src/src/lib.rs	**Medium**	Acknowledged
Timelocks are not verified during order creation	cross-chain-escrow-src/src/lib.rs	**Low**	Acknowledged
Timelocks are not included in seeds on destination chain	cross-chain-escrow-dst/src/lib.rs	**Low**	Acknowledged

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Mint account is not validated

Risk Level: Critical

Status: Fixed in the pull request 115

Contracts:

- cross-chain-escrow-src/src/lib.rs,
- cross-chain-escrow-dst/src/lib.rs

Description:

In `cross-chain-escrow-src/src/lib.rs`, the `Withdraw`, `PublicWithdraw`, `CancelEscrow`, and `PublicCancelEscrow` structs do not validate the mint account, allowing an attacker to pass an arbitrary mint account.

Example attack vector (by the taker):

1. Create an escrow.
2. Wait for cancellation.
3. Cancel using fake tokens by passing an arbitrary mint (the maker will receive the fake tokens).
4. Call `rescue_funds` for the escrow with the real mint (this can be done immediately because `rescue_start` is now `None`) and receive real tokens.

Additionally, this vulnerability allows bypassing the rescue delay via `Withdraw`. Another taker may also block someone's funds via `PublicWithdraw`, as the escrow account will be closed.

The same attack can be performed on the destination chain, since the mint is not validated in the `Withdraw`, `PublicWithdraw`, and `Cancel` structs.

Remediation:

Consider verifying that the mint account matches the one used in the initial order.

5.2. Insufficient destination rescue access control.

Risk Level: High

Status: Fixed in the pull request 115

Contracts:

- cross-chain-escrow-dst/src/lib.rs

Location: Function: `rescue_funds`.

Description:

Anyone can call `rescue` for any escrow on the destination chain. The creator is not validated in `RescueFunds`, allowing any user to invoke the rescue (which should only be done by the escrow creator or designated resolvers).

Remediation:

Consider validating that the `creator` is a legitimate resolver.

5.3. Rescue start could be earlier than cancellation start

Risk Level: Medium

Status: Client's comment: The timelocks parameters are validated on the backend, and within the contract, we assume that all durations individually and in total will be significantly less than 8 days, which is set as the delay before rescue_start.

Contracts:

- cross-chain-escrow-dst/src/lib.rs,
- cross-chain-escrow-src/src/lib.rs

Description:

The cancellation start may be intentionally or accidentally set by the maker after the rescue start in the timelock, allowing the resolver to extract funds from the escrow instead of canceling it.

Remediation:

Consider verifying that the rescue start is scheduled after each timelock period.

5.4. Timelocks are not verified during order creation

Risk Level: Low

Status: Client's comment: The timelocks parameters are checked on the backend, and resolvers can choose not to process orders with timelocks they find unacceptable.

Contracts:

- cross-chain-escrow-src/src/lib.rs

Location: Function: `create`.

Description:

The maker can create an order with invalid timelocks—for example, by setting all values to zero. These values are then mutated to `max(UINT256)` by the `set_deployed_at` function in `common/src/timelocks.rs`, after which the deployment date is set.

If a taker fills such an order, they will be forced to wait until the rescue start, as all timelock getter functions will revert due to a `uint32` overflow in the `get` function.

Additionally, timelocks do not verify that time deltas follow the correct chronological order, allowing, for example, a withdrawal start time to be set after the cancellation time.

Remediation:

Consider verifying that the timelocks provided by the maker do not result in a `uint32` overflow after mutation and that the time deltas follow the correct chronological order.

5.5. Timelocks are not included in seeds on destination chain

Risk Level: Low

Status: Client's comment: The timelocks parameters will still need to be checked at least once on the frontend, at the moment of creating escrow_dst. Indeed, there are differences in protocol consistency, but in practice, the frontend uses the input data of the escrow_dst creation transaction to validate the parameters, and adding timelocks to the seeds will not improve the usability of the protocol.

Contracts:

- cross-chain-escrow-dst/src/lib.rs

Location: Function: `create`.

Description:

Based on the Solidity logic (see BaseEscrowFactory.sol, line 134), the parameters used for order creation are hashed and used as a salt, ensuring integrity between chains.

However, in the Solana implementation, the taker can pass arbitrary parameters (with timelocks being the primary concern). This allows the taker to create an escrow with parameters that may be unsuitable for the maker—for example, setting the cancellation start time earlier than the withdrawal start time, which would cause the `withdraw` function to revert and lock the funds.

This forces the maker to perform additional validation before proceeding.

Remediation:

It is recommended to include timelocks in the seed parameters of the destination chain escrow account to ensure consistency and prevent tampering.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.