Smart Contract Security Audit Report

By

Igor Gulamov

Cross-chain swap v2

Table of contents

1. Disclaimer

Important to remember:

1. This audit was performed based on the current state of the code at the time of evaluation. Any subsequent changes or modifications to the codebase could render auditors' findings obsolete. Re-audit is recommended post any alterations.
2. While we strive for accuracy, auditors cannot guarantee that all potential vulnerabilities or bugs have been identified. The auditor is not responsible for any overlooked issues.
3. It's always recommended to have multiple layers of checks and balances, including but not limited to, regular code reviews and updated audits.

2. Summary

2.1. Project Description

V2 Changes

1. Added partial fills
2. Added zkSync deployment feature
3. Added missing events
4. Fixed minor issues and refactorings

V2 changes on github

https://github.com/1inch/cross-chain-swap/compare/2604086c45850117d16e020ea7a4e4ee798302a5...391183551a597c63c2185eb0497430057b63f652

Repo

https://github.com/1inch/cross-chain-swap

Scope

- contracts/BaseEscrow.sol
- contracts/BaseEscrowFactory.sol
- contracts/Escrow.sol
- contracts/EscrowDst.sol
- contracts/EscrowFactory.sol
- contracts/EscrowSrc.sol
- contracts/MerkleStorageInvalidator.sol
- contracts/interfaces/IBaseEscrow.sol
- contracts/interfaces/IEscrow.sol
- contracts/interfaces/IEscrowDst.sol
- contracts/interfaces/IEscrowFactory.sol
- contracts/interfaces/IEscrowSrc.sol
- contracts/interfaces/IMerkleStorageInvalidator.sol
- contracts/interfaces/IResolverMock.sol
- contracts/libraries/ImmutablesLib.sol
- contracts/libraries/ProxyHashLib.sol
- contracts/libraries/TimelocksLib.sol
- contracts/zkSync/EscrowDstZkSync.sol
- contracts/zkSync/EscrowFactoryZkSync.sol
- contracts/zkSync/EscrowSrcZkSync.sol
- contracts/zkSync/EscrowZkSync.sol
- contracts/zkSync/MinimalProxyZkSync.sol
- contracts/zkSync/ZkSyncLib.sol

2.2. Scope

- 1inch/cross-chain-swap

2.3. Conclusion

We consider commit 30cb5aabd37e00a7ca2c2d7e55bcd8de84cf4098 as a safe version from the informational security point of view.

3. Issue Summary

	Critical	Major	Medium	Minor
Fixed	0	0	0	2
Not Fixed	0	0	0	2

4. Issues

4.1. Critical

No critical issues found.

4.2. Major

No major issues found.

4.3. Medium

No medium issues found.

4.4. Minor

4.4.1. Optimize abi decode at `takerInteraction`

- 1inch/cross-chain-swap/39118.../contracts/MerkleStorageInvalidator.sol#L55

Severity: Minor
Category: optimization
Status: Fixed

Solidity-generated abi decoder is inefficient. We propose replacing

```
(
    bytes32 root,
    bytes32[] memory proof,
    uint256 idx,
    bytes32 secretHash
) = abi.decode(extraData, (bytes32, bytes32[], uint256, bytes32));
```

with

```
assembly("memory-safe") {
    root := calldataload(extraData.offset)
    let ptr := add(extraData.offset, calldataload(add(extraData.offset, 0x20)))
    proof.offset := add(ptr, 0x20)
    proof.length := calldataload(ptr)

    idx := calldataload(add(extraData.offset, 0x40))
    secretHash := calldataload(add(extraData.offset, 0x60))

    // assert extraData.length == proof.length*0x20 + 0xa0
    if iszero(eq(sub(extraData.length, mul(proof.length, 0x20)), 0xa0)) {
        revert(0,0)
    }

    //assert extraData.offset+extradata.length == proof.offset+proof.length*0x20
    if iszero(eq(add(extraData.offset, 0xa0), proof.offset)) {
        revert(0,0)
    }
}
```

It is 30% more gas efficient.

Feedback:

Fixed in PR 89

4.4.2. Optimize `key` computation at `_postInteraction`

- 1inch/cross-chain-swap/39118.../contracts/BaseEscrowFactory.sol#L80

Severity: Minor
Category: optimization
Status: Will Not Fix

We propose to replace

```
80 bytes32 key = keccak256(abi.encodePacked(orderHash, uint240(uint256(extraDataArgs.hashlock))));
```

with

```
assembly ("memory-safe") {
  mstore(0x1e, hashlock)
  mstore(0x00, orderHash)
  key := keccak256(0x00, 62)
}
```

for gas optimization.

Feedback:

Noted

4.4.3. Optimize `key` computation at `takerInteraction`

- 1inch/cross-chain-swap/39118.../contracts/MerkleStorageInvalidator.sol#L56

Severity: Minor
Category: optimization
Status: Will Not Fix

We propose to replace

```
56 bytes32 key = keccak256(abi.encodePacked(orderHash, uint240(uint256(extraDataArgs.hashlock))));
```

with

```
assembly ("memory-safe") {
  mstore(0x1e, hashlock)
  mstore(0x00, orderHash)
  key := keccak256(0x00, 62)
}
```

for gas optimization.

Feedback:

Noted

4.4.4. Add more documentation to `TimelocksLib`

- 1inch/cross-chain-swap/39118.../contracts/libraries/TimelocksLib.sol#L71

Severity: Minor
Category: documentation
Status: Fixed

We propose adding more descriptions to the `get` function, especially the reasons why the lowest `uint32` (`bitShift=0`) means only the half of the resulting timestamp, according to

```
71     function get(Timelocks timelocks, Stage stage) internal pure returns (uint256) {
72         uint256 data = Timelocks.unwrap(timelocks);
73         uint256 bitShift = uint256(stage) * 32;
74         // The maximum uint32 value will be reached in 2106.
75         return uint32(data) + uint32(data >> bitShift);
76     }
```

Feedback:

Fixed in PR 101