CODESPECT

# ECDSA
# Token Distributor

## SECURITY ASSESSMENT REPORT

23 April, 2025

*Prepared for*

TOKENTABLE

# Contents

# 1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

**Smart Contract Auditing:** Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

**Secure Design & Architecture Consultancy:** At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

**Tailored Cybersecurity Solutions**: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

# 2 Disclaimer

**Limitations of this Audit:** This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

**Inherent Risks of Blockchain Technology:** Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

**Purpose and Reliance of this Report:** This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

**Liability Disclaimer:** To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

**Third-Party Products and Services:** CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

**Further Recommendations:** We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

**Disclaimer of Advice:** FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

# 3 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 |

Table 1: Risk Classification Matrix based on Likelihood and Impact

### 3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

### 3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

### 3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

# 4 Executive Summary

This document presents the results of a security assessment conducted by CODESPECT for TokenTable. TokenTable is a token distribution platform that facilitates airdrops, vesting, and other mechanisms for distributing tokens.

This audit focuses on the EVM contracts responsible for managing token distribution to recipients based on crafted signed data. The claiming process requires a valid ECDSA signature to successfully complete the claim.

**The audit was performed using:**

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found three points of attention, one classified as `Medium` and two classified as `Informational`. All of the issues are summarised in Table 2.

**Organisation of the document is as follows:**

- **Section 5** summarizes the audit.
- **Section 6** describes the system overview.
- **Section 7** presents the issues.
- **Section 8** discusses the documentation provided by the client for this audit.
- **Section 9** presents the compilation and tests.

## Issues found:

| Severity | Unresolved | Fixed | Acknowledged |
|---|---|---|---|
| Medium | 0 | 1 | 0 |
| Informational | 0 | 0 | 2 |
| **Total** | **0** | **1** | **2** |

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

# 5 Audit Summary

| | |
|---|---|
| **Audit Type** | Security Review |
| **Project Name** | TokenTable |
| **Type of Project** | ECDSA Distributor |
| **Duration of Engagement** | 3 Days |
| **Duration of Fix Review Phase** | 1 Day |
| **Draft Report** | April 17, 2025 |
| **Final Report** | April 23, 2025 |
| **Repository** | ecdsa-token-distributor |
| **Commit (Audit)** | 6d5db7f144d7468644313c98f9f310dbaadd1b01 |
| **Commit (Final)** | 9c2cc47a5ed2dd745a3396f9c51733f9f25a69b6 |
| **Documentation Assessment** | Medium |
| **Test Suite Assessment** | Medium |
| **Auditors** | JecikPo, shaflow01 |

Table 3: Summary of the Audit

## 5.1 Scope - Audited Files

| | File | LoC |
|---|---|---|
| 1 | core/BaseECDSADistributor.sol | 248 |
| 2 | extensions/FungibleTokenWithFeesECDSADistributor.sol | 103 |
| 3 | extensions/FungibleTokenECDSADistributor.sol | 64 |
| 4 | interfaces/IClaimHook.sol | 20 |
| | **Total** | **435** |

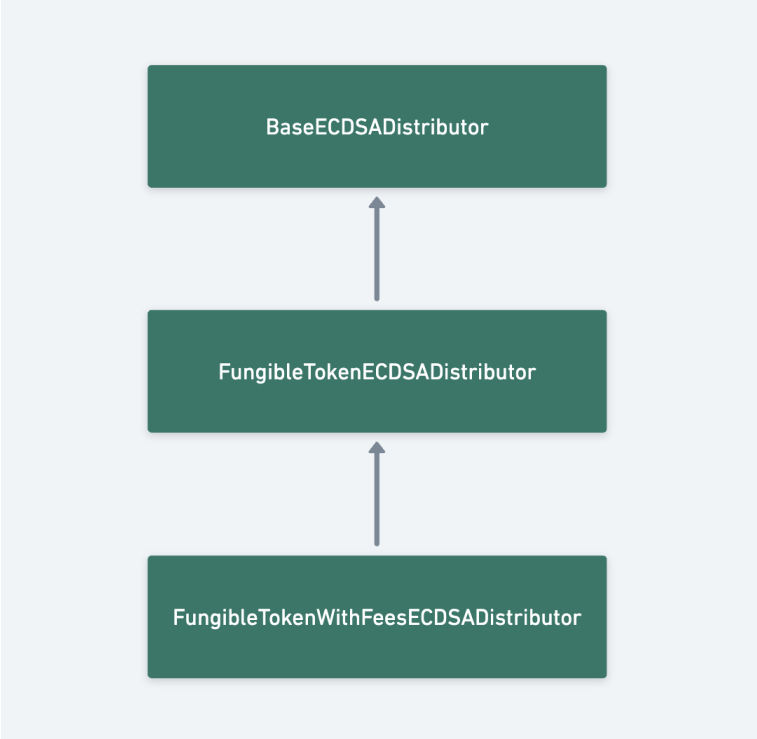
## 5.2 Findings Overview

| | Finding | Severity | Update |
|---|---|---|---|
| 1 | The upgrade permission for the protocol was assigned to the wrong role. | Medium | Fixed |
| 2 | Hook call contains untrusted data | Info | Acknowledged |
| 3 | The `version` information is not included in the signed data. | Info | Acknowledged |

# 6 System Overview

TokenTable introduced a set of EVM contracts that facilitate a token distribution mechanism based on claim signature validation. Users provide data which describes their claim along with a signature. The signature is then validated by the contract, and if the validation succeeds deposited tokens can be claimed. Fees are charged on each claim.

The following diagram depicts the contracts inheritance model:




The `BaseECDSADistributor` is an abstract contract containing common functionality and a framework for specialised distributors:

- `FungibleTokenECDSADistributor`: Provides basic token distribution functionality with dynamic fees controlled by the `FeeCollector` contract.
- `FungibleTokenWithFeesECDSADistributor`: Contains a similar distribution mechanism, with fixed fees defined by the signed claim data.

## 6.1 FungibleTokenECDSADistributor

The `FungibleTokenECDSADistributor` contract is deployed by a deployer contract and supports a single token distribution "event". The contract has an owner who can set basic distribution parameters through the `setBaseParams` external function:

- `startTime` and `endTime`: which define time constraints on the token distribution.
- `token`: the specific ERC20 token address of the distributed asset.
- `authorizedSigner`: The address which was used to create claim data signatures and hence will also be used for signature validations.

Claimants can call `claim(...)` with valid signatures and claim data. Each claim is verified against the expected message format (including Chain ID, contract address, recipient, and claim ID), and must be signed by the authorised signer. A unique `userClaimId` ensures that each claim can only be fulfilled once.

The contract optionally supports a fee model, charging a fixed fee per claim or group of claims, payable in either ERC20 or native ETH. Fees are collected via an external fee collector contract determined by the deployment registry.

A customizable `claimHook` mechanism is available via the `IClaimHook` interface, allowing protocol-specific behaviour to be injected before and after each claim is processed (e.g., logging, validation, or analytics).

The contract uses ERC-7201 storage slot conventions and provides a version string through the `IVersionable` interface. It supports UUPS upgradeability, with upgrade authorization restricted to the contract owner via the `_authorizeUpgrade()` method.

## 6.2 FungibleTokenWithFeesECDSADistributor

The `FungibleTokenWithFeesECDSADistributor` contract extends the ECDSA-based token distribution mechanism of the `FungibleTokenECDSADistributor`, with added support for individualised per-claim fee amounts.

Each claimable entry includes not only the claimable token amount and unlock timestamp but also a specified fee amount, all of which are encoded and signed off-chain. Upon calling the `claim(...)` function, users provide an array of signed payloads that the contract verifies using ECDSA signatures. Each valid claim triggers the internal distribution of tokens to the specified recipient and accumulates the specified fee into a running total.

Unlike its base implementation, where the fee is charged per claim invocation, this contract charges fees per individual claim, allowing for finer-grained cost accounting. Once all claims are processed, the total accumulated fee is collected using the `_chargeFixedFee(...)` mechanism. Depending on the deployment configuration, fees may be charged in ERC20 tokens or in native ETH.

The contract overrides and disables several inherited methods, such as `decodeUserClaimData` and `_afterClaim()`, indicating that such behaviours are either unnecessary or intentionally unsupported in this variant.

The claim data for each recipient must conform to the

- `FungibleTokenWithFeesECDSADistributorData` struct, which includes:
- `claimableTimestamp`: the UNIX timestamp after which the tokens may be claimed,
- `claimableAmount`: the amount of tokens the user is entitled to receive,
- `fees`: the fee associated with processing this individual claim.

Claims submitted outside the valid time window are rejected, as are duplicate claims or those with invalid signatures. As with all `BaseECDSADistributor`-based contracts, the implementation is upgradeable and enforces strict access control, pausing, and reentrancy protections.

This contract is intended for use cases where each recipient must bear an individualised cost for claiming tokens, enabling more flexible and dynamic fee models.

# 7 Issues

## 7.1 [Medium] The upgrade permission for the protocol was assigned to the wrong role.

**File(s)**: `BaseECDSADistributor.sol`

**Description**: In the protocol, there are two roles: one is the `deployer` contract controlled by the `tokenTable`, which is responsible for initializing the `ECDSADistributor` contract and includes the fee parameters required for token distribution. The second role is the `contract owner`, which is controlled by the project team responsible for the token distribution. However, the upgrade privilege is assigned to the `contract owner`, which can lead to potential issues.

```
// solhint-disable-next-line no-empty-blocks
function _authorizeUpgrade(address newImplementation) internal virtual override onlyOwner { }
```

**Impact**: The project team can upgrade the `ECDSADistributor` contract and set the `deployer` address to a malicious implementation they control. This allows them to bypass paying fees to the `tokenTable` or even steal the fees.

**Recommendation(s)**: It is recommended to transfer the upgrade authority to the `deployer` contract controlled by the `tokenTable`.

**Status**: Fixed

**Update from TokenTable**: Fixed at 9c2cc47a5ed2dd745a3396f9c51733f9f25a69b6

## 7.2 [Info] Hook call contains untrusted data

**File(s)**: BaseECDSADistributor.sol, FungibleTokenWithFeesECDSADistributor.sol

**Description**: The `userClaimData` in the hook call is not part of the signed data and is instead allowed to be arbitrarily provided by the caller.

```
function claim(
    ...
    bytes[] calldata extraDatas
)
   ...
{
    //...
}
```

```
function __tryCallClaimHook(...)
    private
{
    address claimHook = _getBaseECDSADistributorStorage().claimHook;
    if (claimHook != address(0)) {
        isBeforeClaim
            ? IClaimHook(claimHook).beforeClaim(delegate, recipient, group, data, extraData)
            : IClaimHook(claimHook).afterClaim(delegate, recipient, group, data, claimedAmount, extraData);
    }
}
```

**Impact**: If the hook incorrectly assumes that this field is trustworthy data, it could impact the system. Moreover, since anyone can claim the receipt on behalf of the user, it's possible that the input for this field is not what the receipt expected.

**Recommendation(s)**: It is recommended to include this data within the signed data to ensure its trustworthiness.

**Status**: Acknowledged

**Update from TokenTable**: Acknowledged

## 7.3 [Info] The `version` information is not included in the signed data.

**File(s)**: `BaseECDSADistributor.sol`

**Description**: The `ECDSADistributor` contract is an upgradeable contract and contains version information. After the contract is upgraded, the version information may be changed.

```
function version() public pure virtual override returns (string memory) {
    return "0.4.1";
}
```

However, when constructing the signed data for user claims, the version information is not included in the signature, which may introduce potential risks.

```
function encodeHashToBeSigned(...)
    public
    view
    virtual
    returns (bytes32)
{
    return keccak256(abi.encode(block.chainid, address(this), recipient, userClaimId, userClaimData));
}
```

**Impact**: After the contract version information is changed, previously generated signatures can still be used. This may pose some potential risks.

**Recommendation(s)**: It is recommended to include the version information in the signed data.

**Status**: Acknowledged

**Update from TokenTable**: Acknowledged

# 8 Evaluation of Provided Documentation

The TokenTable team provided documentation in a single format:

- **Natspec Comments:** The code includes comments for some processes, which explained the purpose of complex functionality in detail and facilitated understanding of individual functions.

The documentation provided by TokenTable while fundamental and basic in nature, was adequate given the limited scope of the audit. It met the necessary requirements and supported the key areas under review. However, the public technical documentation could be further improved to better present the protocol's overall functionality and facilitate the understanding of each component.

Additionally, the TokenTable team was consistently available and responsive, promptly addressing all questions raised by CODESPECT during the evaluation process.

# 9 Test Suite Evaluation

## 9.1 Compilation Output

ECDSA Distributor's compilation output:

```
% forge build
[] Compiling...
[] Compiling 29 files with Solc 0.8.28
[] Solc 0.8.28 finished in 1.58s
Compiler run successful!
```

## 9.2 Tests Output

ECDSA Distributor's test output:

```
% forge test
[] Compiling...
No files changed, compilation skipped

Ran 5 tests for test/FungibleTokenWithFeesECDSADistributor.test.sol:FungibleTokenWithFeesECDSADitributorTest
[PASS] test_Claim_OutsideTimeRangeAndInactive() (gas: 335892)
[PASS] test_claim() (gas: 362189)
[PASS] test_claim_chargeFixedFee() (gas: 264712)
[PASS] test_decodeUserClaimData() (gas: 11536)
[PASS] test_getStorage() (gas: 37372)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 6.89ms (4.88ms CPU time)

Ran 8 tests for test/FungibleTokenECDSADistributor.test.sol:FungibleTokenECDSADitributorTest
[PASS] testFuzz_getClaimStatus(bytes32[]) (runs: 256, : 159690, ~: 159198)
[PASS] testFuzz_withdraw(uint256,uint256,bool) (runs: 256, : 80247, ~: 68367)
[PASS] test_claim() (gas: 424695)
[PASS] test_decodeUserClaimData() (gas: 10537)
[PASS] test_encodeHashToBeSigned() (gas: 12869)
[PASS] test_getStorage() (gas: 37125)
[PASS] test_setBaseParams() (gas: 54970)
[PASS] test_verify() (gas: 43684)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 53.08ms (63.19ms CPU time)

Ran 2 test suites in 141.73ms (59.97ms CPU time): 13 tests passed, 0 failed, 0 skipped (13 total tests)
```

## 9.3 Notes about Test suite

The TokenTable team delivered a test suite that includes coverage of the basic flows and functionalities. It also includes a variety of fuzzing tests. The use of fuzzing tests enabled coverage of numerous edge cases. These tests validate the behavior of the system under a wide range of inputs, uncovering potential vulnerabilities or inconsistencies that could emerge under unexpected conditions.

**CODESPECT** also recommends explicitly defining strict invariants that the protocol must uphold. Incorporating tests to validate these invariants would ensure that critical assumptions about the system's behaviour are consistently maintained across all functionalities, further bolstering the protocol's security and stability.