CODESPECT

Merkle Token Distributor With Fees

SECURITY ASSESSMENT REPORT

June 27, 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 an EVM contract responsible for managing token distribution to recipients based on merkle proof verification and charging fees. The claiming process requires providing a valid merkle proof to successfully complete the claim. This contract supports both ERC20 and Native token distribution.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found one point of attention, classified as `Medium`.

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
Total	**0**	**1**	**0**

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

5 Audit Summary

Audit Type	Security Review
Project Name	Merkle Token Distributor With Fees
Type of Project	Token Distributor
Duration of Engagement	1 Day
Duration of Fix Review Phase	1 Day
Draft Report	June 27, 2025
Final Report	June 27, 2025
Repository	tokentable-evm
Commit (Audit)	ebb6ef16f4e26faa153efabca9c789bb516af12d
Commit (Final)	0e4cd1d1c27dfbb98080728da8955a10d1143a9c
Documentation Assessment	NaN
Test Suite Assessment	Medium
Auditors	Kalogerone

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	LoC
1	core/extensions/TokenTableMerkleDistributorWithFees.sol	103
	Total	**103**

5.2 Findings Overview

	Finding	Severity	Update
1	Contract handles native tokens but the withdraw function doesn't	Medium	Fixed

6 System Overview

TokenTable has introduced a set of EVM contracts that facilitate a token distribution mechanism based on merkle proof validation of the claim hash. Users provide data which describes their claim along with a merkle proof. The proof is validated by the contract, and if it succeeds deposited tokens can be claimed.

This audit focused on an extension of the `BaseMerkleDistributor` which supports both `ERC20` and `Native` tokens and charges fees based on the following principles:

- If claimable tokens and fee tokens are the same, the fee is deducted from the claimable amount and directly sent to the `FeeCollector` contract.
- If claimable tokens and fee tokens are different, the fee is charged separately and directly sent to the `FeeCollector` contract.

7 Issues

7.1 [Medium] Contract handles native tokens but the withdraw function doesn't

File(s): `TokenTableMerkleDistributorWithFees.sol`

Description: The `TokenTableMerkleDistributorWithFees` contract is supposed to handle airdrops/distributions of native tokens, hence the `receive()` function. However, the `withdraw(...)` function can only retrieve `ERC20` tokens:

```
function withdraw(bytes memory) external virtual override onlyOwner {
    IERC20 token = IERC20(_getBaseMerkleDistributorStorage().token);
    token.safeTransfer(owner(), token.balanceOf(address(this)));
}
```

Impact: When the `TokenTableMerkleDistributorWithFees` contract is used for native tokens, the `withdraw(...)` function can't be used and any excess or unclaimed tokens will be stuck in the contract.

Recommendation(s): Handle both cases appropriately, as the airdrop token can also be a `native` token.

Status: Fixed

Update from TokenTable: Fixed in f99596681d9d416beb2fa94ac08836ea21bcdd24

8 Evaluation of Provided Documentation

The TokenTable team did not provide any new documentation, as most contract components inherit functionality from previously reviewed contracts.

Despite the limited scope of the audit, the TokenTable team remained consistently available and responsive, promptly addressing all questions raised by **CODESPECT** throughout the evaluation process. This level of cooperation was sufficient for the needs of the engagement.

9 Test Suite Evaluation

9.1 Compilation Output

```
% forge build src/merkle/core/extensions/TokenTableMerkleDistributorWithFees.sol
[] Compiling...
[] Compiling 15 files with Solc 0.8.29
[] Solc 0.8.29 finished in 226.41ms
Compiler run successful!
```

9.2 Tests Output

```
% forge test test/merkle/TokenTableMerkleDistributorWithFees.t.sol
[] Compiling...
[] Compiling 81 files with Solc 0.8.29
[] Solc 0.8.29 finished in 3.33s
Compiler run successful!

Ran 10 tests for test/merkle/TokenTableMerkleDistributorWithFees.t.sol:TokenTableMerkleDistributorWithFeesTest
[PASS] test_ETHDistribution() (gas: 175931)
[PASS] test_claimOutsideTimeRange() (gas: 386298)
[PASS] test_claimWithERC20Fees() (gas: 567063)
[PASS] test_claimWithETHFees() (gas: 396980)
[PASS] test_claimWithSameTokenFees() (gas: 354440)
[PASS] test_doubleClaim() (gas: 363827)
[PASS] test_getContractAddress() (gas: 6804)
[PASS] test_leafEncoding() (gas: 13210)
[PASS] test_proofVerification() (gas: 114228)
[PASS] test_withdraw() (gas: 246539)
Suite result: ok. 10 passed; 0 failed; 0 skipped; finished in 7.31ms (10.18ms CPU time)

Ran 1 test suite in 10.86ms (7.31ms CPU time): 10 tests passed, 0 failed, 0 skipped (10 total tests)
```

9.3 Notes about Test suite

The TokenTable team delivered a test suite that includes a variety of unit tests to cover most of the flows and functionalities. These tests ensure that individual components of the code behave correctly in isolation.

CODESPECT has considered relevant to increase the test coverage by implementing additional fuzz tests for TokenTableMerkleDistributorWithFees contract. Incorporating fuzz tests to validate all functionalities and edge cases ensures that critical assumptions made during a manual audit are held true in order to maintain the protocol's security and stability.