CODESPECT

Batched Merkle Token Distributor

SECURITY ASSESSMENT REPORT

August 27, 2025

Prepared for

TOKENTABLE

Contents

1 About CODESPECT 2

2 Disclaimer 2

3 Risk Classification 3

4 Executive Summary 4

5 Audit Summary 5
5.1 Scope - Audited Files . . . 5
5.2 Findings Overview . . . 5

6 System Overview 6

7 Issues 7
7.1 [Info] Missing token index bounds validation leads to array out-of-bounds revert in batch claims . . . 7

8 Evaluation of Provided Documentation 8

9 Test Suite Evaluation 9
9.1 Compilation Output . . . 9
9.2 Tests Output . . . 9
9.3 Notes about Test suite . . . 9

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 reviews an EVM smart contract responsible for managing token distributions through Merkle proof verification. The contract introduces support for **batched claims**, where a single Merkle leaf can represent multiple claims within one batch. To complete a claim, recipients must provide a valid Merkle proof, ensuring the integrity and correctness of the distribution process. The contract supports both **ERC20 tokens** and the **native token**, providing flexibility for diverse distribution scenarios.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found one point of attention, one 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
Informational	0	0	1
Total	**0**	**0**	**1**

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

5 Audit Summary

Audit Type	Security Review
Project Name	TokenTable
Type of Project	Token Distributor
Duration of Engagement	2 Days
Duration of Fix Review Phase	1 Day
Draft Report	June 27, 2025
Final Report	June 27, 2025
Repository	tokentable-evm
Commit (Audit)	442aca0222550cf985e2b94c07ba0255bf972471
Commit (Final)	133fe98851bdbddd8d73d9763153607c72753621
Documentation Assessment	NaN
Test Suite Assessment	High
Auditors	0xluk3, Bloqarl

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	LoC
1	extensions/TokenTableMerkleMultitokenDistributorBatched.sol	173
	Total	**103**

5.2 Findings Overview

	Finding	Severity	Update
1	Missing token index bounds validation leads to array out-of-bounds revert in batch claims	Info	Acknowledged

6 System Overview

TokenTable has introduced an extension of the `BasedMerkleDistributor` contract, enabling token distribution based on valid Merkle proof verification. This audited extension modifies the way claims are handled: each leaf of the Merkle tree now represents a **group of claims**, encapsulated in the `BatchClaimData` struct.

The `BatchClaimData` struct contains two fields:

- `packedRecipientTimeToken` – a packed representation of the claim recipient, the claimable timestamp, and the token to be claimed.
- `claimableAmount` – the amount of tokens assigned for distribution.

Claims are executed via the `batchClaim(...)` function:

```
function batchClaim(
    BatchClaimData[] calldata batchData,
    bytes32[] calldata proof
)
    external
    virtual
    onlyActive
    whenNotPaused
    nonReentrant
{
```

This function allows anyone to submit a valid Merkle proof for a given batch, ensuring the appropriate token amounts are distributed to the correct recipients.

7 Issues

7.1 [Info] Missing token index bounds validation leads to array out-of-bounds revert in batch claims

File(s): TokenTableMerkleMultitokenDistributorBatched.sol

Description: The `_sendByIndex(...)` function accesses the `tokens` array using a user-provided `tokenIndex` without validating that the index is within the array bounds:

```
function _sendByIndex(address recipient, uint256 tokenIndex, uint256 amount) internal {
    address token = tokens[tokenIndex];  // No bounds checking
    // ...
}
```

While the `packRecipientTimeToken(...)` function validates that `tokenIndex < 2^56` to prevent bit-packing overflow, it doesn't validate against the actual `tokens.length`. This creates a gap where values like `tokenIndex = 999` pass the bit-packing validation but cause array out-of-bounds reverts when accessing `tokens[999]` in an array that typically will not contain many elements.

Impact: Batch claims containing invalid token indices will revert with unnecessary array out-of-bounds errors

Recommendation(s): Add explicit bounds validation in the `packRecipientTimeToken(...)` which will be used for crafting the batches.

Status: Acknowledged

Update from TokenTable: Acknowledged

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 --match-contract TokenTableMerkleMultitokenDistributorBatchedTest

Ran 16 tests for
 ↪  test/merkle/TokenTableMerkleMultitokenDistributorBatched.t.sol:TokenTableMerkleMultitokenDistributorBatchedTest
[PASS] testBatchClaimMultipleTokens() (gas: 435527)
[PASS] testBitPacking() (gas: 22038)
[PASS] testCalldataOptimization() (gas: 3166)
[PASS] testClaimOutsideTimeRange() (gas: 209659)
[PASS] testETHTransferFailedReverts() (gas: 28635)
[PASS] testGenerateGroupHash() (gas: 24630)
[PASS] testIndividualClaimReverts() (gas: 12186)
[PASS] testInvalidProofInBatch() (gas: 179946)
[PASS] testLargeBatchSize() (gas: 521972)
[PASS] testPauseFunctionality() (gas: 75343)
[PASS] testSetTokensConstraints() (gas: 136778)
[PASS] testTokenIndexing() (gas: 170349)
[PASS] testUnsupportedOperationReverts() (gas: 13805)
[PASS] testVersion() (gas: 12345)
[PASS] testWithdrawMultipleTokens() (gas: 145126)
[PASS] testWithdrawSpecificToken() (gas: 149630)
Suite result: PASSED. 16 passed; 0 failed; 0 skipped; finished in 15.31ms (20.93ms CPU time)


% forge test --match-contract  TokenTableMerkleMultitokenDistributorBatchedFuzzTest

Ran 9 tests for
 ↪  test/merkle/TokenTableMerkleMultitokenDistributorBatchedFuzz.t.sol:TokenTableMerkleMultitokenDistributorBatchedFuzzTest
[PASS] testFuzz_BitPacking(address,uint40,uint56) (runs: 256, : 14345, ~: 14345)
[PASS] testFuzz_ClaimAmounts(uint256,uint8) (runs: 256, : 871434, ~: 742056)
[PASS] testFuzz_GroupHash(uint8,uint256) (runs: 256, : 61028, ~: 41289)
[PASS] testFuzz_MultipleRecipients(uint8,uint256) (runs: 256, : 408122, ~: 396952)
[PASS] testFuzz_SetTokens(address[]) (runs: 256, : 1322384, ~: 1286883)
[PASS] testFuzz_SetTokensEmpty() (gas: 13770)
[PASS] testFuzz_TimeValidation(uint256,uint256,uint256) (runs: 256, : 294814, ~: 292906)
[PASS] testFuzz_TimestampOverflow(address,uint256,uint56) (runs: 256, : 13164, ~: 14222)
[PASS] testFuzz_TokenIndexOverflow(address,uint40,uint256) (runs: 256, : 13329, ~: 14342)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 859.73ms (1.87s CPU time)
```

9.3 Notes about Test suite

The TokenTable team provided a comprehensive test suite covering a wide range of flows and functionalities. The unit tests ensure correctness of individual components in isolation, while the inclusion of fuzz tests demonstrates diligence in validating behaviour under a variety of randomised inputs.

CODESPECT recognises the value of this approach, as fuzz testing helps uncover edge cases that might not be anticipated through manual test design. This strengthens confidence that the system's assumptions hold true and contributes positively to the overall robustness and security of the protocol.