CODESPECT

Merkle Token Distributor

SECURITY ASSESSMENT REPORT

28 April, 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**
6.1 BaseMerkleDistributor . . . 7
6.2 TokenTableNativeMerkleDistributor . . . 7
6.3 CustomFeesNativeMerkleDistributor . . . 7
6.4 SimpleERC721MerkleDistributor . . . 7
6.5 SimpleNoMintERC721MerkleDistributor . . . 7
6.6 NFTGatedMerkleDistributor . . . 7

7 Issues **8**
7.1 [Medium] Incorrect Withdrawal Implementation May Lead to Lock of Unclaimed NFTs . . . 8
7.2 [Low] Upgrade Permission for the Protocol Assigned to the Project Owner . . . 8
7.3 [Info] The NFT Fee Handling is Incompatible with BIPS Type of Fees . . . 9
7.4 [Info] `getClaimDelegate` Function Not Blocked When Delegated Claiming is Disabled . . . 9

8 Evaluation of Provided Documentation **10**

9 Test Suite Evaluation **11**
9.1 Compilation Output . . . 11
9.2 Tests Output . . . 12
9.3 Notes about Test suite . . . 13

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 merkle proof verification. The claiming process requires providing a valid merkle proof to successfully complete the claim. This collection of contracts supports both ERC20 and ERC721 token distribution.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found four points of attention, one classified as `Medium`, one as `Low` 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
Low	0	1	0
Informational	0	0	2
Total	**0**	**2**	**2**

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

5 Audit Summary

Audit Type	Security Review
Project Name	TokenTable
Type of Project	Merkle Token Distributor
Duration of Engagement	4 Days
Duration of Fix Review Phase	1 Day
Draft Report	April 28, 2025
Final Report	April 28, 2025
Repository	merkle-token-distributor
Commit (Audit)	96fedd0d945693149e0903c84502004bf819996c
Commit (Final)	0e4cd1d1c27dfbb98080728da8955a10d1143a9c
Documentation Assessment	Medium
Test Suite Assessment	Medium
Auditors	JecikPo, Bloqarl

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	LoC
1	core/MDCreate2.sol	53
2	core/BaseMerkleDistributor.sol	274
3	core/extensions/SimpleNoMintERC721MerkleDistributor.sol	8
4	core/extensions/TokenTableNativeMerkleDistributor.sol	27
5	core/extensions/CustomFeesNativeMerkleDistributor.sol	94
6	core/extensions/TokenTableMerkleDistributor.sol	45
7	core/extensions/SimpleERC721MerkleDistributor.sol	17
8	core/extensions/custom/NFTGatedMerkleDistributor.sol	102
	Total	**620**

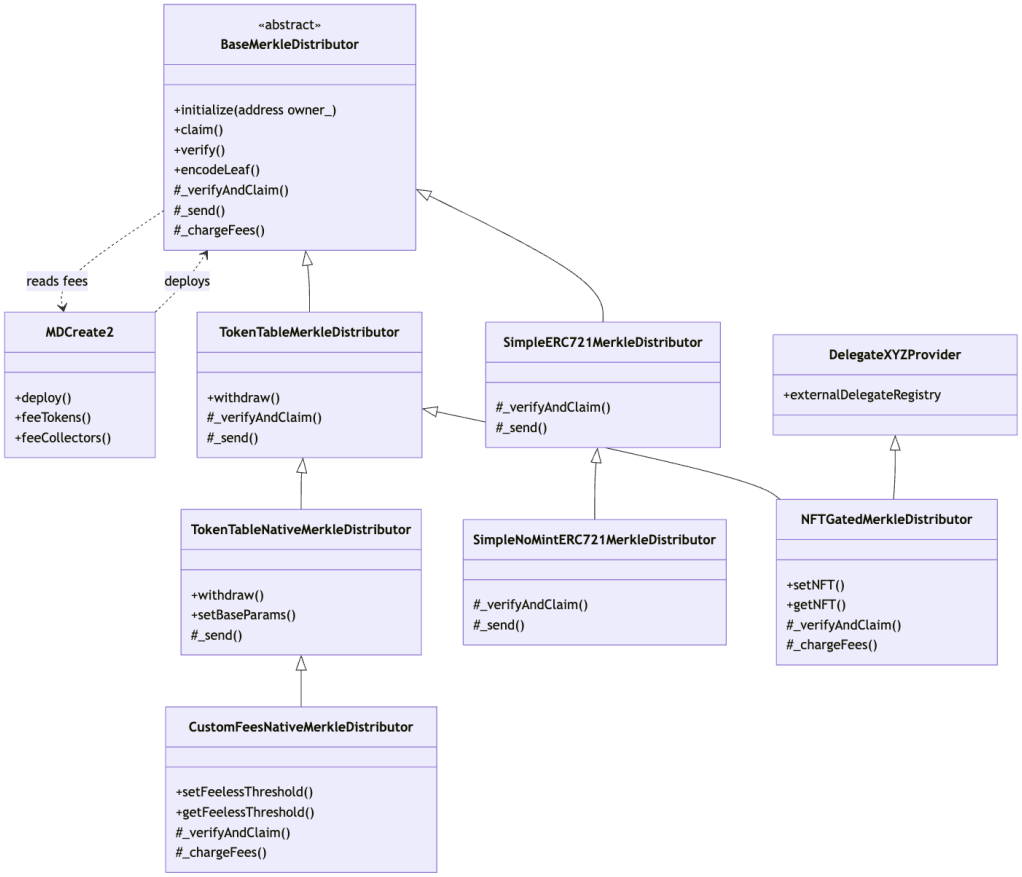
5.2 Findings Overview

	Finding	Severity	Update
1	Incorrect Withdrawal Implementation May Lead to Lock of Unclaimed NFTs	Medium	Fixed
2	Upgrade Permission for the Protocol Assigned to the Project Owner	Low	Fixed
3	The NFT Fee Handling is Incompatible with BIPS Type of Fees	Info	Acknowledged
4	`getClaimDelegate` Function Not Blocked When Delegated Claiming is Disabled	Info	Acknowledged

6 System Overview

TokenTable 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. Fees are charged on each claim. The collection of contracts support ERC20 and ERC721.

The following diagram depicts the contracts inheritance model:



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

- `TokenTableMerkleDistributor`: Provides basic token distribution functionality with dynamic fees controlled by the `FeeCollector` contract.
- `TokenTableNativeMerkleDistributor`: Contains a similar distribution mechanism, but for the native currency of the chain.
- `CustomFeesNativeMerkleDistributor`: Allows distribution of the native currency of the chain, with an additional feeless threshold for claims.
- `SimpleERC721MerkleDistributor`: Allows ERC721 token distribution with minting.
- `SimpleNoMintERC721MerkleDistributor`: Provides ERC721 token distribution of pre-minted tokens deposited to the distribution contract.
- `NFTGatedMerkleDistributor`: With the integration to Delegate.xyz, the contract provides capabilities of claiming based on ownership of a defined NFT tokens.

The `MDCreate2` contract allows deployment of Distributor instances.

6.1 BaseMerkleDistributor

The `BaseMerkleDistributor` contract is deployed by a `MDCreate2` contract and supports a single token distribution "event" based on a merkle tree proof verfication. The contract has an owner who can configure basic distribution parameters through the `setBaseParams` external function.

Claimants can call `claim` with a valid Merkle proof and associated claim data. Each claim is verified against the Merkle root and must match a unique leaf constructed from the Chain ID, contract address, recipient address, group identifier, and claim-specific data. Each leaf can only be claimed once.

The contract also supports delegated claiming via `delegateClaim` and batch delegated claiming through `batchDelegateClaim`. Delegates must be explicitly authorized by the contract owner using `setClaimDelegate`.

A flexible fee model is integrated, where fees may be charged per claim or based on the claimed amount. Fees are collected by an external fee collector contract determined by the deployment registry and can be payable in either native token or ERC20 tokens.

A customizable `claimHook` mechanism is available via the `IClaimHook` interface, enabling protocol-specific behaviour before and after claim execution, such as logging, external validation, or analytics.

The contract uses ERC-7201 storage slot conventions and exposes a version string through the `IVersionable` interface.

6.2 TokenTableNativeMerkleDistributor

The `TokenTableNativeMerkleDistributor` contract extends `TokenTableMerkleDistributor` to support native ETH distributions instead of ERC20 tokens. It enforces that no ERC20 token address is set via `setBaseParams`, reverting if a nonzero address is provided. Claims transfer native ETH directly to recipients, and any unclaimed balance can be withdrawn by the owner through the `withdraw` function. The contract also accepts ETH deposits via a `receive` fallback function, ensuring compatibility with native token workflows.

6.3 CustomFeesNativeMerkleDistributor

The `CustomFeesNativeMerkleDistributor` contract extends `TokenTableNativeMerkleDistributor` by introducing customizable fee logic based on claim amounts. It allows the owner to set a "feeless threshold", under which claims are exempt from fees. Claims now go through a two-step process: `_verifyAndClaim` validates and records the claim without transferring funds, while `_chargeFees` handles fee deductions and sends the appropriate native ETH amounts to claimants and fee collectors. The contract disables delegate claims to simplify the logic and ensure fee mechanisms remain consistent.

6.4 SimpleERC721MerkleDistributor

The `SimpleERC721MerkleDistributor` contract extends `TokenTableMerkleDistributor` to support ERC721 token distribution. Instead of transferring fungible tokens or native ETH, it either safely transfers existing ERC721 tokens back to the owner using the `withdraw` function or mints new NFTs directly to recipients during claims via the `_send` function. The number of NFTs minted corresponds to the amount specified in the claim, and minting is done through a safe minting interface (`IERC721SafeMintable`).

6.5 SimpleNoMintERC721MerkleDistributor

The `SimpleNoMintERC721MerkleDistributor` contract builds upon `SimpleERC721MerkleDistributor` but modifies the `_send` logic to transfer pre-existing ERC721 tokens rather than minting new ones. Here, the `amount` parameter is repurposed to represent the `tokenId`, and each claim results in the specified NFT being safely transferred from the distributor contract to the recipient.

6.6 NFTGatedMerkleDistributor

The `NFTGatedMerkleDistributor` contract extends `TokenTableMerkleDistributor` and introduces NFT-based gating for claims. Each airdrop claim is tied to the ownership of a specific NFT token ID, with eligibility optionally extended via Delegate.xyz delegation. The contract prevents setting claim delegates or batch claiming, enforces optional expiry timestamps on claims, and disables fee collection. Claims must be verified against the Merkle tree and the NFT ownership at the time of claiming, ensuring only the rightful NFT owner or authorized delegate can claim the associated airdrop.

7 Issues

7.1 [Medium] Incorrect Withdrawal Implementation May Lead to Lock of Unclaimed NFTs

File(s): `SimpleERC721MerkleDistributor.sol`

Description: The `SimpleERC721MerkleDistributor` contract allows claiming of ERC721 tokens instead of ERC20. It inherits most of its functions from `TokenTableMerkleDistributor`. The main difference is that `_send()` and `withdraw()` functions handle ERC721 token minting and transfers instead of ERC20 transfers.

The claiming process in this case relies on minting the NFTs directly from the `token` contract. In the ERC20 versions the project owner is equipped with `withdraw()`, which allows to recover all non-claimed tokens.

In this `SimpleERC721MerkleDistributor` contract, however the the overloaded `withdraw()` attempts to transfer existing NFTs from the contract. This will not work because the unclaimed NFTs are not actually minted to that contract.

```
function withdraw(bytes memory extraData) external virtual override onlyOwner {
    uint256[] memory tokenIds = abi.decode(extraData, (uint256[]));
    for (uint256 i = 0; i < tokenIds.length; i++) {
        IERC721(_getBaseMerkleDistributorStorage().token).safeTransferFrom(address(this), owner(), tokenIds[i]);
    }
}

function _send(address recipient, address token, uint256 amount) internal virtual override {
    for (uint256 i = 0; i < amount; i++) {
        IERC721SafeMintable(token).safeMint(recipient);
    }
}
```

Impact: In the case where the minting permissions are granted to the distributor contract, but not to the project owner, the owner cannot mint the unclaimed tokens for himself and they might end up locked (or rather never minted).

Recommendation(s): Change the `withdraw()` function in `SimpleERC721MerkleDistributor` so that it mints the NFTs instead of transferring them. **Warning!** If that change is implemented the `withdraw()` function must also be placed into the `SimpleNoMintERC721MerkleDistributor` contract as it inherits from `SimpleERC721MerkleDistributor`. If that is not done, then withdrawals will not work for `SimpleNoMintERC721MerkleDistributor`

Status: Fixed

Update from TokenTable: Revised withdraw logic in `0e4cd1d1c27dfbb98080728da8955a10d1143a9c`.

7.2 [Low] Upgrade Permission for the Protocol Assigned to the Project Owner

File(s): `BaseMerkleDistributor.sol`

Description: In the protocol, there are two roles:

- The `MDCreate2` contract controlled by TokenTable, which is responsible for initialising the contracts inheriting from `BaseMerkleDistributor` and includes the fee parameters required for token distribution;
- 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 { }
```

It gives the project owner control to upgrade the distribution contracts.

Impact: The project team can upgrade the contract and set the deployer address to a malicious implementation they control. This allows them to bypass paying fees to TokenTable or even take the fees for themselves.

Recommendation(s): Removal of the upgradability option.

Status: Fixed

Update from TokenTable: Fixed in `c991b09f8da9eba24b0a789e6c7cb332d0394f40`.

7.3 [Info] The NFT Fee Handling is Incompatible with BIPS Type of Fees

File(s): `SimpleERC721MerkleDistributor.sol`

Description: In case of ERC721 token distribution, the `claimedAmount` of the individual claim is representing the amount of tokens to be minted or transferred. As is the case with NFTs that variable can be small, i.e. 1 or 2. The `claimedAmount` is then passed into the `ITTUFeeCollector::getFee()` to calculate the exact fee amount charged to the claimer. While it works fine in case fixed fees are configured, it may not work as expected in case the protocol owner would like to charge fees based on bips, as the following calculation from the `ITTUFeeCollector::getFee()` may return zero in case of a low value of `claimedAmount`:

```
tokensCollected = (tokenTransferred * feeBips) / BIPS_PRECISION;
```

The exact number when zero is returned depends on the value of `feeBips`.

Impact: Project needs to stick to fixed fees in case of ERC721 distributions, hence, fee policy flexibility is lost.

Recommendation(s): multiply the `claimedAmount` by `BIPS_PRECISION` before sending it to `ITTUFeeCollector::getFee()`.

Status: Acknowledged

Update from TokenTable: Acknowledged.

7.4 [Info] `getClaimDelegate` Function Not Blocked When Delegated Claiming is Disabled

File(s): `NFTGatedMerkleDistributor.sol`

Description: The `NFTGatedMerkleDistributor` contract has delegated claims disabled. The following functions:

- `setClaimDelegate();`
- `batchDelegateClaim();`
- `delegateClaim();`

are blocked using `revert`. However the `getClaimDelegate()` is not.

Impact: No impact to the protocol functionality, only inconsistent implementation.

Recommendation(s):

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

Merkle Distributor's compilation output:

```
% forge build
[] Compiling...
[] Compiling 91 files with Solc 0.8.28
[] Solc 0.8.28 finished in 3.47s
Compiler run successful with warnings:
Warning (5740): Unreachable code.
   --> src/core/BaseMerkleDistributor.sol:115:52:
    |
115 |         for (uint256 i = 0; i < recipients.length; i++) {
    |                                                    ^^^

Warning (5740): Unreachable code.
  --> node_modules/@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol:76:9:
   |
76 |         _nonReentrantAfter();
   |         ^^^^^^^^^^^^^^^^^^^

Warning (5740): Unreachable code.
   --> src/core/BaseMerkleDistributor.sol:137:9:
    |
137 |         emit Claimed(recipient, group, data);
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning (5740): Unreachable code.
  --> src/core/extensions/TokenTableMerkleDistributor.sol:43:9:
   |
43 |         if (
   |         ^ (Relevant source part starts here and spans across multiple lines).

Warning (5740): Unreachable code.
  --> src/core/extensions/TokenTableMerkleDistributor.sol:47:9:
   |
47 |         _send(recipient, _getBaseMerkleDistributorStorage().token, decodedData.claimableAmount);
   |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning (5740): Unreachable code.
  --> src/core/extensions/TokenTableMerkleDistributor.sol:48:9:
   |
48 |         return decodedData.claimableAmount;
   |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

9.2 Tests Output

Merkle Distributor's test output:

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

Ran 4 tests for test/NFTGatedMerkleDistributor.t.sol:NFTGatedMerkleDistributorTest
[PASS] testFuzz_decodeMOCALeafData(uint256,uint256,uint256,uint256,uint256) (runs: 260, : 15375, ~: 15375)
[PASS] testFuzz_verifyAndClaim(address[],bytes32[],uint256[],uint8[],uint128[],uint8[]) (runs: 259, : 47978415, ~:
↪  45271423)
[PASS] test_disabledFunctions() (gas: 29628)
[PASS] test_setNft() (gas: 40938)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 20.99s (20.98s CPU time)

Ran 16 tests for test/TokenTableMerkleDistributor.t.sol:TokenTableMerkleDistributorTest
[PASS] testFuzz_claim_fail_notActive(uint128,uint128) (runs: 258, : 87619, ~: 88314)
[PASS] testFuzz_decodeLeaf(uint256,uint256,uint256) (runs: 260, : 13653, ~: 13653)
[PASS] testFuzz_delegateClaim_fail_notActive(address,uint128,uint128) (runs: 257, : 112869, ~: 113644)
[PASS] testFuzz_delegateClaim_fail_notDelegate(address,uint128,uint128) (runs: 257, : 113159, ~: 113469)
[PASS] testFuzz_encodeLeaf_verify_and_claim(address[],bytes32[],uint256[],uint64[],uint128[]) (runs: 259, : 30529348,
↪  ~: 29165225)
[PASS] testFuzz_encodeLeaf_verify_and_delegateClaim(address,address[],bytes32[],uint256[],uint64[],uint128[]) (runs:
↪  260, : 25263405, ~: 21769939)
[PASS] testFuzz_encodeLeaf_verify_no_claim(address[],bytes32[],uint256[],uint256[],uint256[]) (runs: 259, : 14291434,
↪  ~: 10845576)
[PASS] testFuzz_externalTTUFeeCollector_erc20Fees(address[],bytes32[],uint256[],uint64[],uint128[]) (runs: 259, :
↪  35704294, ~: 34149394)
[PASS] testFuzz_externalTTUFeeCollector_etherFees(address[],bytes32[],uint256[],uint64[],uint128[]) (runs: 259, :
↪  32555965, ~: 29520057)
[PASS] testFuzz_setBaseParams_fail_badTime(address,uint256,uint256,bytes32) (runs: 258, : 14393, ~: 14393)
[PASS] testFuzz_setBaseParams_fail_notOwner(address,address,uint256,uint256,bytes32) (runs: 258, : 16293, ~: 16293)
[PASS] testFuzz_setBaseParams_succeed_0(address,uint256,uint256,bytes32) (runs: 258, : 107163, ~: 107549)
[PASS] testFuzz_setClaimDelegate_fail_notOwner(address,address) (runs: 260, : 15987, ~: 15987)
[PASS] testFuzz_setClaimDelegate_succeed_0(address) (runs: 260, : 37875, ~: 37952)
[PASS] testFuzz_withdraw_fail_notOwner(address) (runs: 260, : 14991, ~: 14991)
[PASS] testFuzz_withdraw_succeed_0(uint256) (runs: 260, : 125581, ~: 125735)
Suite result: ok. 16 passed; 0 failed; 0 skipped; finished in 22.09s (98.53s CPU time)

Ran 2 test suites in 22.10s (43.08s CPU time): 20 tests passed, 0 failed, 0 skipped (20 total tests)
```

NFTGatedMerkleDistributorTest's test output:

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

Ran 14 tests for test/NFTGatedMerkleDistributor.t.sol:NFTGatedMerkleDistributorTest
[PASS] testFuzz_decodeMOCALeafData(uint256,uint256,uint256,uint256,uint256) (runs: 260, : 16177, ~: 16177)
[PASS] testFuzz_verifyAndClaim(address[],bytes32[],uint256[],uint8[],uint128[],uint8[]) (runs: 259, : 48561928, ~:
↪  45823278)
[PASS] testFuzz_verifyAndClaimWithDifferentBalances(address,uint256,uint256) (runs: 258, : 328633, ~: 328708)
[PASS] testFuzz_verifyAndClaimWithDifferentTimestamps(address,uint256,uint256,uint256,uint256) (runs: 258, : 344934, ~:
↪  344963)
[PASS] testFuzz_verifyAndClaimWithMultipleLeaves(uint256,uint256) (runs: 260, : 548406, ~: 502226)
[PASS] test_disabledFunctions() (gas: 30326)
[PASS] test_revertWhenClaimingAfterExpiry(address) (runs: 260, : 302125, ~: 302125)
[PASS] test_revertWhenClaimingWithoutDelegation(address,address) (runs: 260, : 313387, ~: 313387)
[PASS] test_revertWhenNonNFTOwnerClaims(address,address,uint256) (runs: 260, : 311759, ~: 311759)
[PASS] test_setNft() (gas: 42521)
[PASS] test_successWhenClaimingBeforeExpiry(address) (runs: 260, : 307863, ~: 307863)
[PASS] test_successWhenClaimingThroughDelegateXYZ(address,address) (runs: 260, : 345110, ~: 345110)
[PASS] test_successWhenClaimingWithNoExpiry(address) (runs: 260, : 307708, ~: 307708)
[PASS] test_successWhenNFTOwnerClaims(address) (runs: 260, : 307946, ~: 307946)
Suite result: ok. 14 passed; 0 failed; 0 skipped; finished in 15.60s (16.31s CPU time)

Ran 1 test suite in 15.60s (15.60s CPU time): 14 tests passed, 0 failed, 0 skipped (14 total tests)
```

9.3 Notes about Test suite

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

CODESPECT has considered relevant to increase the test coverage by implementing additional fuzz tests for NFTGatedMerkleDistributorTest 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.