CODESPECT

ECDSA Token Distrubutor (SUI)

SECURITY ASSESSMENT REPORT

August 21, 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 SUI 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 eight points of attention, one classified as `Critical`, two classified as `Medium`, two classified as `Low`, two classified as `Informational`, and one classified as `Best Practices`. 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
Critical	0	1	0
Medium	0	1	1
Low	0	2	0
Informational	0	2	0
Best Practices	0	1	0
Total	**0**	**7**	**1**

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

5 Audit Summary

Audit Type	Security Review
Project Name	ECDSA Token Distrubutor (SUI)
Type of Project	Token Distributor
Duration of Engagement	4 Days
Duration of Fix Review Phase	2 Days
Draft Report	August 21, 2025
Final Report	August 21, 2025
Repository	ecdsa-token-distributor-sui
Commit (Audit)	5536d80395d269b7d3392b20a924cbcae7a86344
Commit (Final)	3f3f05504e458fee6ce0163673fec0582ca7c3af
Documentation Assessment	Low
Test Suite Assessment	Low
Auditors	shaflow01, 0xluk3

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Finding	**Severity**	**Update**
1	The `send_tokens(...)` and `mark_claimed(...)` functions lack access control	Critical	Fixed
2	The `claim(...)` and `claim_with_fees(....)` functions lack token type checks	Medium	Fixed
3	`Distributor` has not been assigned a fee collection type	Medium	Acknowledged
4	The fee collector is optional on `Distributor` creation, but obligatory on claiming leading to temporary DoS	Low	Fixed
5	The token distributor cannot control distribution parameters and cannot withdraw undistributed tokens	Low	Fixed
6	Lack of one-time witness may allow objects created in the `init(...)` function to be created after upgrade	Info	Fixed
7	version string could be a single named constant instaed of 2 different strings	Info	Fixed
8	Redundant `get_distributor_info` function calls	Best Practices	Fixed

6 System Overview

TokenTable introduces a new Sui Move package containing five modules, which work together as an independent on-chain system to provide users with token distribution capabilities:

a) **ownable** module – used to create and manage global administrator privilege credentials.

b) **fee_collector** module – used to configure fee-related settings for token distribution modules and collect fees.

c) **base_distributor** module – used to create and configure token distribution objects for each project and manage the distribution of tokens.

d) **fungible_token_distributor** module – used to handle token claiming patterns based on fee configuration.

e) **fungible_token_with_fees_distributor** module – used to handle token claiming patterns based on fees collected from signed data.

This audit primarily focuses on the overall logical completeness and correctness of the package.

6.1 ownable

In the ownable module, the init function creates an `OwnerCap` object and transfers it to the caller. This serves as the privilege credential within the package, and many global configurations require holding an `OwnerCap` object.

The section below outlines the main functions of the module, categorized by the entities authorized to call them. Module functions can be called by the holder of the `OwnerCap` object:

1. `transfer_ownership` – transfer ownership of the `OwnerCap`

6.2 fee_collector

The `fee_collector` module is controlled by the holder of the `OwnerCap` object. It can configure default fees and default tokens, as well as set custom fee tokens and fee amounts for a specific Project. It stores the collected fees and allows the `OwnerCap` holder to withdraw them.

The section below outlines the main functions of the module, categorized by the entities authorized to call them.

Module functions can be called by the holder of the `OwnerCap` object:

1. `withdraw_fee` – used to withdraw the collected fees
2. `set_default_fee` – sets the default fee amount
3. `set_custom_fee_fixed` – sets a specific fixed fee for a given `distributor_address`
4. `set_default_fee_token` – sets the default fee token type
5. `set_custom_fee_token` – sets a specific fee token type for a given `distributor_address`

Module functions can be called by anyone:

1. `collect_fee` – pays the fee and stores it in a specific dynamic field within the `FeeCollectorConfig`

6.3 base_distributor

The `base_distributor` module allows any token distributor to create a `Distributor` object. After the `OwnerCap` completes the configuration, the token distributor deposits the tokens to be distributed. Then, the `Distributor`'s `authorized_signer` can construct the distribution data and distribute signatures to the recipients, who can later claim the tokens.

The section below outlines the main functions of the module, categorized by the entities authorized to call them.

Module functions can be called by the holder of the `OwnerCap` object:

1. `set_base_params` – configures relevant parameters for a given `Distributor` object, such as `start_time`, `end_time`, and `authorized_signer`.
2. `toggle_pause` – pauses token distribution for a given `Distributor` object.
3. `set_fee_collector` – configures the `fee_collector` for a given `Distributor` object.
4. `withdraw` – withdraws the undistributed tokens from a given `Distributor` object.

Module functions can be called by anyone:

1. create_distributor – creates a new Distributor object for token distribution.
2. deposit – deposits tokens into the given Distributor for distribution.

Module functions can be called by the holder of the given Distributor object:

1. share_distributor – change the given Distributor into a shared object so that it can be configured by the administrator and claimed by recipients.

6.4 fungible_token_distributor

The fungible_token_distributor module is used to handle logic related to token recipients claiming tokens. The token claiming data does not include fee information; the type and amount of fees paid for claiming tokens are obtained from the fee_collector.

The section below outlines the main functions of the module, categorized by the entities authorized to call them.

Module functions can be called by anyone:

1. claim – token recipients submit the distribution data signed by the authorized_signer of the Distributor object. After verification, fees are collected according to the fee_collector configuration, and the distributable tokens are transferred from the Distributor object to the receiver.

6.5 fungible_token_with_fees_distributor

The fungible_token_with_fees_distributor module is used to handle the logic for token recipients claiming tokens. The claim data includes fee information, and the amount of fees to be paid for claiming tokens is obtained from the claim data itself.

The section below outlines the main functions of the module, categorized by the entities authorized to call them.

Module functions can be called by anyone:

1. claim_with_fees – token recipients submit the distribution data signed by the authorized_signer of the Distributor object. After verification, the fees are collected according to the claim data, and the distributable tokens are transferred from the Distributor object to the receiver.

7 Issues

7.1 [Critical] The `send_tokens(...)` and `mark_claimed(...)` functions lack access control

File(s): `base_distributor.move`

Description: During the token claim process, the `mark_claimed(...)` function is called to mark tokens as claimed to prevent replay, and then the `send_tokens(...)` function is called to transfer tokens from the `Distributor` treasury to the recipient.

```
public fun mark_claimed<T>(
    distributor: &mut Distributor<T>,
    claim_id: vector<u8>
) {
    assert!(!is_claimed(distributor, claim_id), E_CLAIM_ALREADY_CLAIMED);
    table::add(&mut distributor.claimed_claims, claim_id, true);
}

public fun send_tokens<T>(
    distributor: &mut Distributor<T>,
    recipient: address,
    amount: u64,
    ctx: &mut TxContext
) {
    let token_balance = sui::balance::split(&mut distributor.token_balance, amount);
    let tokens = sui::coin::from_balance(token_balance, ctx);
    transfer::public_transfer(tokens, recipient);
}
```

However, both functions lack access control, allowing anyone to call them directly.

Impact: A malicious actor can call `mark_claimed(...)` before a user claims, marking the tokens as claimed and causing the user's claim to fail. They can also call `send_tokens(...)` directly to steal all unclaimed tokens from the `Distributor` treasury.

Recommendation(s): Use `public(package)` to restrict these functions so they can only be called by this package.

Status: Fixed

Update from TokenTable: 44f9228953fb383f3f356061ba9e245e22389d8d Note: public(friend) is deprecated

7.2 [Medium] The `claim(...)` and `claim_with_fees(....)` functions lack token type checks

File(s): `fungible_token_distributor.move`, `fungible_token_with_fees_distributor.move`

Description: In the `claim(...)` function, there is no check to verify whether the fee token type provided by the caller matches the type configured in `fee_collector`, allowing the caller to use a self-created worthless token to pay the fee.

```
public entry fun claim<T, F>(...) {
    //...
    assert!(option::is_some(&fee_collector_addr), E_FEE_COLLECTOR_NOT_SET);
    let base_fee = fee_collector::get_fee(fee_config, distributor_address);
    let expected_amount = base_fee * multiplier;
    assert!(coin::value(&fee_payment) == expected_amount, E_INCORRECT_FEES);
    fee_collector::collect_fee(fee_config, fee_payment, ctx);
}
```

In the `claim_with_fees(...)` function, although the fee token is arbitrarily specified by the `authorized_signer`, the signed data contains only the fee amount and no fee token type, making it impossible to verify whether the token paid by the user matches the `authorized_-signer`'s expectation.

```
// Claim data structure with fees
public struct ClaimDataWithFees has drop {
    claimable_timestamp: u64,
    claimable_amount: u64,
    fees: u64
}
```

Impact: Users can evade paying token claim fees to the project team.

Recommendation(s): It is recommended that the `claim(...)` function check whether the fee token provided by the user matches the type configured in `fee_collector`.

For `claim_with_fees(...)`, it is recommended to include the fee token type in the signature fields so that the `claim_with_fees` function can perform the check.

Status: Fixed

Update from TokenTable: 355c59dae35234a78a9f00893bb3ab5814875dcf

7.3 [Medium] `Distributor` has not been assigned a fee collection type

File(s): `base_distributor.move`

Description: The current system has two fee collection methods: one is managed by `OwnerCap` through `fee_collector`, which configures the fee token and fee amount; the other allows the `authorized_signer` in each `Distributor` object to arbitrarily specify it in the signed data.

```
public entry fun claim<T, F>(...) {
    //...
}

public entry fun claim_with_fees<T, F>(...) {
    //...
}
```

However, the `Distributor` object does not have a field distinguishing the fee collection method. As a result, the `authorized_signer` can choose either method or mix them at will.

Impact: This causes fee collection to be outside the control of the project team and allows the token issuer to specify it arbitrarily. The project team may suffer losses from the fees.

Recommendation(s): It is recommended to add a field in the `Distributor` to distinguish the fee collection type.

Status: Acknowledged

Update from TokenTable: Since fee value will be baked into the signature for claim type claim_with_fee, mixed up usaged will cause claim to fail. So there is no risk of fee loss.

Update from CODESPECT The issue in the report is not that users can freely use signatures from the `authorized_signer`—because mixed-up usage would cause the claim to fail—but rather that the `authorized_signer` (controlled by the token distributor) can arbitrarily assign signature types. For example, the project team may want a certain distributor to use the fees configured in `fee_collector`, but the `authorized_signer` could choose to distribute some signatures related to `ungible_token_with_fees_distributor` with a fee of 0, allowing their users to avoid paying fees to the project team. If the project team is willing to accept this, the issue will be marked as acknowledged.

Update From TokenTable: The claim process will be handled by our front-end which will call the right claim type. We will also communicate with `authorized_signer` (the token distributor) in advance to make sure the correctness of signature.

7.4 [Low] The fee collector is optional on `Distributor` creation, but obligatory on claiming leading to temporary DoS

File(s): `fungible_token_distributor.move`, `fungible_token_with_fees_distributor.move`, `base_distributor.move`

Description: Distributors can be created without a fee collector (its an `Option<address>`), however, the claiming routine enforces the existence of such, otherwise the claims revert. This allows for the creation of default broken Distributors, where claiming will not be possible, requiring Owner intervention to set a fee collector in order to restore the Distributor state.

```
//In claim routine:
assert!(option::is_some(&fee_collector_addr), E_FEE_COLLECTOR_NOT_SET);`\
[...]
//In the distributor creation:
public fun set_fee_collector<T>(distributor: &mut Distributor<T>, fee_collector: Option<address>, _owner_cap:
 ↪  &OwnerCap)`\
```

Impact: It is possible to create non-functional distributors which will not allow claiming, leading to partial DoS condition until manual intervention is performed.

Recommendation(s): Enforce setting a fee collector on Distributor creation or allow Distributors without fee collectors (depending on the business needs of the protocol).

Status: Fixed

Update from TokenTable: The configuration will be done manually and carefully managed to avoid human error.

7.5 [Low] The token distributor cannot control distribution parameters and cannot withdraw undistributed tokens

File(s): base_distributor.move

Description: The set_base_params(...) function sets token distribution parameters, including start and end times, etc. The withdraw(...) function is used to claim the tokens that the issuer deposited into the Distributor. Both functions can only be called by the project team (OwnerCap holder), so the token issuer cannot call them freely. This is inconsistent with implementations on other chains.

```
public fun set_base_params<T>(
    distributor: &mut Distributor<T>,
    start_time: u64,
    end_time: u64,
    authorized_signer: address,
    _owner_cap: &OwnerCap
) {
    //...
}

public entry fun withdraw<T>(
    distributor: &mut Distributor<T>,
    _owner_cap: &OwnerCap,
    amount: Option<u64>,
    ctx: &mut TxContext
) {
    //...
}
```

Impact: This permission restriction reduces the token issuer's flexibility in distribution, as they cannot freely start or end a distribution nor claim back undistributed tokens on their own.

Recommendation(s): It is recommended to create a permission credential object for each Distributor. This object should include a field pointing to the Distributor ID to distinguish credentials for different distributors.

Status: Fixed

Update from TokenTable: Github Commit

Update from CODESPECT:It is recommended to set the permission for set_fee_collector to AdminCap. Although this field currently has no effect, in other versions the fee_collector is set by the administrator.

We also noticed that in the new version of the code, a project_id is assigned to the distributor. Currently, creating a distributor requires no permission. If a malicious actor were to preemptively occupy a project_id, would there be any impact, considering that the permissions of the two functions have now been moved to the creator?

If the project_id has a special meaning or purpose and is not randomly generated, and if it is not associated on the backend after the transaction, then it is recommended that the AdminCap first assigns the associated project_id → creator, and then the creator can create the distributor and freely configure its parameters.

Update from TokenTable:

1. We made another update that eliminates the need to set fee_collector. Now all contract calls will all refer to the global FeeCollectorConfig. This part is handled by our front-end;
2. projec_id has no special meaning so it is fine;

Commit: b40f93884b1c21d4bcf780f4d06c5964a3b4eca6

We made another update that requires each distribution to pass in a fee collector reference during creation. Also changed get_distribution_info to public entry to facilitate front-end. 3f3f05504e458fee6ce0163673fec0582ca7c3af

7.6 [Info] Lack of one-time witness may allow objects created in the `init(...)` function to be created after upgrade

File(s): `ownable.move`, `fee_collector.move`

Description: For objects that need to ensure global uniqueness, we typically create them in the `init` function because the `init` function is only executed once during package deployment. For example, the `FeeCollectorConfig` object in `fee_collector` and the `OwnerCap` object in `ownable`.

However, since a one-time witness is not used, there is no guarantee that these objects are globally unique. Subsequent upgrades could introduce new functions to create these objects.

Impact: `OwnerCap` and `FeeCollectorConfig` may no longer remain globally unique after an upgrade, introducing potential risk such as permission escalation and configuration conflicts.

Recommendation(s): It is recommended to use a one-time witness for objects that need to ensure global uniqueness.

Status: Fixed

Update from TokenTable: Github Commit

Update from CODESPECT Not fixed. It seems that the `witness` is not actually being used. To ensure the uniqueness of the object, the `witness` should be used in a manner like this.

```
public struct OWNABLE has drop {}

public struct OwnerCap<phantom T> has key {
    id: UID,
}

fun init(witness: OWNABLE, ctx: &mut TxContext) {
    // The witness proves this is the first and only time this is called
    assert!(sui::types::is_one_time_witness(&witness), 0);

    transfer::transfer(OwnerCap<OWNABLE> {
        id: object::new(ctx),
    }, tx_context::sender(ctx));
}
```

Update From TokenTable: 163eb236b6725a04efb29316c1fe1e7640b7ae31

7.7 [Info] version string could be a single named constant instead of 2 different strings

File(s): `fee_collector.move`, `ownable.move`

Description: In both the fee_collector and ownable modules, the version function uses the same version string (converted via string::utf8).

```
module ecdsa_token_distributor_sui::ownable {
    //...
    public fun version(): String {
        string::utf8(b"0.1.0")
    }
    //...
}

module ecdsa_token_distributor_sui::base_distributor {
    //...
    public fun version(): String {
        string::utf8(b"0.1.0")
    }
    //...
}
```

Impact: Defining the same version string in two separate places creates maintenance overhead, as upgrading the version requires modifying two hardcoded instances in the code.

Recommendation(s): It is recommended to optimise this by using a single named constant instead of redundantly defining two separate strings.

Status: Fixed

Update from TokenTable: 2f6b14dc43be6cfd664eaa0ff6b884393d1f0898

7.8 [Best Practice] Redundant `get_distributor_info` function calls

File(s): `fungible_token_distributor.move`, `fungible_token_distributor.move`

Description: In the `claim(...)` and `claim_with_fees(...)` functions, `get_distributor_info(...)` is called multiple times to retrieve the same data, and the number of redundant calls increases with the number of loop iterations.

```
public entry fun claim<T, F>(...) {
    let (_, _, _, _, _, paused, _, _, _) = base_distributor::get_distributor_info(distributor);
    //...
    while (i < len) {
        //...
        let (_, authorized_signer, _, _, _, _, _, _, _) = base_distributor::get_distributor_info(distributor);
        //...
    };
    let multiplier = if (delegate_mode) len else 1;
    let (distributor_address, _, _, _, _, _, fee_collector_addr, _, _) =
    ↪  base_distributor::get_distributor_info(distributor);
    //...
}

public entry fun claim_with_fees<T, F>(...) {
    let (_, _, _, _, _, paused, _, _, _) = base_distributor::get_distributor_info(distributor);
    //...
    while (i < len) {
        //...
        let (_, authorized_signer, _, _, _, _, _, _, _) = base_distributor::get_distributor_info(distributor);
        //...
    };
    let (_, _, _, _, _, _, fee_collector_addr, _, _) = base_distributor::get_distributor_info(distributor);
    //...
}
```

Impact: Such redundant calls increase gas consumption and reduce code readability.

Recommendation(s): It is recommended to retrieve all required variables in a single function call and use them directly in subsequent operations.

Status: Fixed

Update from TokenTable: 2f6b14dc43be6cfd664eaa0ff6b884393d1f0898

8 Evaluation of Provided Documentation

The **TokenTable** team provided official documentation. It contains the protocol's design and implementation details, providing an overview of the protocol's purpose for both users and auditors. However, the current state of the documentation website does not contain a version for Sui contracts.

The documentation provided valuable insights into the protocol, significantly aiding **CODESPECT**'s understanding.

However, the code lacks comments for key processes and design, and expanding documentation coverage would enhance the overall comprehensibility of the code.

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

9 Test Suite Evaluation

9.1 Compilation Output

```
% sui move build
[note] Dependencies on Bridge, MoveStdlib, Sui, and SuiSystem are automatically added, but this feature is disabled for
 ↪  your package because you have explicitly included dependencies on Sui. Consider removing these dependencies from
 ↪  Move.toml.
UPDATING GIT DEPENDENCY https://github.com/MystenLabs/sui.git
INCLUDING DEPENDENCY Sui
INCLUDING DEPENDENCY MoveStdlib
BUILDING ecdsa_token_distributor_sui
```

9.2 Tests Output

```
% sui move test
[note] Dependencies on Bridge, MoveStdlib, Sui, and SuiSystem are automatically added, but this feature is disabled for
↪  your package because you have explicitly included dependencies on Sui. Consider removing these dependencies from
↪  Move.toml.
INCLUDING DEPENDENCY Sui
INCLUDING DEPENDENCY MoveStdlib
BUILDING ecdsa_token_distributor_sui

Running Move unit tests
[ PASS    ] ecdsa_token_distributor_sui::base_distributor_tests::test_deposit_and_withdraw
[ PASS    ] ecdsa_token_distributor_sui::fungible_token_distributor_tests::test_complete_claim_flow
[ PASS    ] ecdsa_token_distributor_sui::fungible_token_with_fees_distributor_tests::test_complete_claim_with_fees_flow
[debug] "=== CLAIM DATA FOR SIGNATURE GENERATION ==="
[debug] "Distributor ID:"
[debug] @0x381dd9078c322a4663c392761a0211b527c127b29583851217f948d62131f409
[debug] "Recipient:"
[debug] @0x456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123
[debug] "Claim ID:"
[debug] 0x746573745f636c61696d5f303031
[debug] "Claimable Timestamp:"
[debug] 1400
[debug] "Claimable Amount:"
[debug] 1000000000000000000
[debug] "Encoded Claim Data (64 bytes):"
[debug] 0x00000000000000000000000000000000000000000000000000000000000005780000000000000000000000000000000000000000000|
↪  00000de0b6b3a7640000
[debug] "Message Hash (32 bytes) - Message to-be-signed"
[debug] 0xaf8472d39646cd2a6eaa9dad0f4b02a9a1d2f8dd448238441b586d5a682b13cc
[ PASS    ] ecdsa_token_distributor_sui::base_distributor_tests::test_generate_hash_for_signature
[ PASS    ]
↪  ecdsa_token_distributor_sui::fungible_token_with_fees_distributor_tests::test_encode_decode_with_fees_consistency
[ PASS    ] ecdsa_token_distributor_sui::fungible_token_distributor_tests::test_encode_decode_consistency
[debug] "=== CLAIM DATA WITH FEES FOR SIGNATURE GENERATION ==="
[debug] "Distributor ID:"
[debug] @0x381dd9078c322a4663c392761a0211b527c127b29583851217f948d62131f409
[debug] "Recipient:"
[debug] @0x456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123
[debug] "Claim ID:"
[debug] 0x746573745f636c61696d5f3030325f66656573
[debug] "Claimable Timestamp:"
[debug] 1500
[debug] "Claimable Amount:"
[debug] 2000000000000000000
[debug] "Fees:"
[debug] 50000000000000000
[debug] "Encoded Claim Data (96 bytes ABI format):"
[debug] 0x00000000000000000000000000000000000000000000000000000000000005dc0000000000000000000000000000000000000000000|
↪  00001bc16d674ec800000000000000000000000000000000000000000000000000000000b1a2bc2ec50000
[debug] "Message Hash (32 bytes) - Message to-be-signed"
[debug] 0xc21ded7ab8f02117069c08323e659604e6f7d7ba03086a083254faccdf77d292
[ PASS    ] ecdsa_token_distributor_sui::base_distributor_tests::test_generate_hash_for_signature_with_fees
[ PASS    ] ecdsa_token_distributor_sui::base_distributor_tests::test_verify
[ PASS    ] ecdsa_token_distributor_sui::base_distributor_tests::test_verify_with_fees
[ PASS    ] ecdsa_token_distributor_sui::base_distributor_tests::test_verify_wrong_signer
Test result: OK. Total tests: 10; passed: 10; failed: 0
```

9.3 Notes about Test suite

The **TokenTable** team delivered a test suite covering signature verification, deposit and withdrawal flows, as well as the two type fee claiming process. These unit tests validate the correctness of key logic in each component, ensuring that all components operate correctly.

However, the current test suite mostly tests the expected execution flows, with only a single test for an expected failure.

CODESPECT has considered it necessary to add more expected-failure tests to ensure that functions throw the intended error codes, as well as to include additional boundary condition tests. Incorporating these tests ensures that the program can correctly handle exceptional inputs, thereby improving the security and stability of the protocol.