Solana Unlocker-V2 Update

SECURITY ASSESSMENT REPORT

16 April, 2025

Prepared for

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 security assessment conducted by CODESPECT for the update of the Unlocker Solana programs of TokenTable. Unlocker is part of a larger suite of protocols designed to streamline token ownership registration and distribution.

This audit focuses on the review of code updates on two Solana programs, which allow users to unlock token distribution for a smaller group of recipients, such as investors or development teams. It offers unique advantages such as unruggability and complete decentralization.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found 8 points of attention, three classified as `Low`, five classified as `Informational`. All of the issues are summarised in Table 2.

Organization 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
Low	0	3	0
Informational	0	5	0
Total	**0**	**8**	**0**

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

5 Audit Summary

Audit Type	Security Review
Project Name	TokenTable
Type of Project	Update of Token Unlocker
Duration of Engagement	2 Days
Duration of Fix Review Phase	2 Days
Draft Report	April 10, 2025
Final Report	April 16, 2025
Repository	tokentable-unlocker-solana
Commit (Audit)	67a39faff7b848ae05c5e3ab45e36b60efcc622e
Commit (Final)	8edb2ab7e2a63c37258b78f365bce2d43db3403f
Documentation Assessment	Medium
Test Suite Assessment	Medium
Auditors	JecikPo, shaflow01

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	File	LoC
0	unlocker-v2-solana/src/events.rs	55
1	unlocker-v2-solana/src/state/claiming_delegate.rs	6
2	unlocker-v2-solana/src/state/mod.rs	8
3	unlocker-v2-solana/src/state/config.rs	7
4	unlocker-v2-solana/src/state/unlocker.rs	15
5	unlocker-v2-solana/src/state/misc.rs	6
6	unlocker-v2-solana/src/constants.rs	3
7	unlocker-v2-solana/src/instructions/deploy.rs	21
8	unlocker-v2-solana/src/instructions/transfer_program_admin.rs	26
9	unlocker-v2-solana/src/instructions/disable_withdraw.rs	18
10	unlocker-v2-solana/src/instructions/transfer_ownership.rs	20
11	unlocker-v2-solana/src/instructions/transfer_actual.rs	36
12	unlocker-v2-solana/src/instructions/set_claiming_delegate.rs	33
13	unlocker-v2-solana/src/instructions/disable_cancel.rs	18
14	unlocker-v2-solana/src/instructions/renounce_ownership.rs	16
15	unlocker-v2-solana/src/instructions/utils.rs	129
16	unlocker-v2-solana/src/instructions/withdraw_deposit.rs	63
17	unlocker-v2-solana/src/instructions/receive_program_admin.rs	24
18	unlocker-v2-solana/src/instructions/disable_create.rs	18
19	unlocker-v2-solana/src/instructions/claim_cancelled_actual_tokens.rs	167
20	unlocker-v2-solana/src/instructions/mod.rs	44
21	unlocker-v2-solana/src/instructions/initialize.rs	73
22	unlocker-v2-solana/src/instructions/cancel.rs	82
23	unlocker-v2-solana/src/instructions/set_fee_collector.rs	50
24	unlocker-v2-solana/src/instructions/disable_transfer_actual.rs	21
25	unlocker-v2-solana/src/instructions/set_fee_token.rs	22
26	unlocker-v2-solana/src/instructions/create_actual.rs	80
27	unlocker-v2-solana/src/instructions/create_preset.rs	99
28	unlocker-v2-solana/src/instructions/claim.rs	277
29	unlocker-v2-solana/src/instructions/deposit.rs	53
30	unlocker-v2-solana/src/errors.rs	46
31	unlocker-v2-solana/src/lib.rs	168
32	unlocker-v2-solana/src/models/preset.rs	63
33	unlocker-v2-solana/src/models/mod.rs	4
34	unlocker-v2-solana/src/models/actual.rs	28
35	fee-collector/src/traits/mod.rs	0
36	fee-collector/src/state/constants.rs	2
37	fee-collector/src/state/mod.rs	2
38	fee-collector/src/instructions/withdraw.rs	69
39	fee-collector/src/instructions/set_custom_fee_fixed.rs	41
40	fee-collector/src/instructions/init_fee_token.rs	33
41	fee-collector/src/instructions/transfer_ownership.rs	22
42	fee-collector/src/instructions/set_default_fee.rs	28
43	fee-collector/src/instructions/utils.rs	58
44	fee-collector/src/instructions/init_project_fee.rs	23
45	fee-collector/src/instructions/set_custom_fee_bips.rs	43
46	fee-collector/src/instructions/mod.rs	24
47	fee-collector/src/instructions/initialize.rs	20
48	fee-collector/src/instructions/get_fee.rs	23
49	fee-collector/src/instructions/collect_fee.rs	89
50	fee-collector/src/instructions/receive_ownership.rs	20
51	fee-collector/src/errors.rs	14
52	fee-collector/src/lib.rs	59
53	fee-collector/src/models/fee.rs	8
54	fee-collector/src/models/mod.rs	4
55	fee-collector/src/models/fee_collector_storage.rs	8
56	fee-collector/src/event.rs	15
	Total	**2404**

5.2 Findings Overview

	Finding	Severity	Update
1	Rent is refunded to the wrong address when the `pending_amount_claimable_for_cancelled_actuals` account is closed	Low	Fixed
2	The `_preset_is_empty` function should not consider `num_of_unlocks_for_each_linear`	Low	Fixed
3	The creation of `pending_amount_claimable_for_cancelled_actuals` account may lead to rent loss	Low	Fixed
4	Allow the `fee_collector` to be set arbitrarily during the initialization of the unlocker account	Info	Fixed
5	Changing the `fee_collector` to a different program will cause instructions to fail	Info	Fixed
6	Lack of `Option` wrapper on `fee` account	Info	Fixed
7	Redundant code	Info	Fixed
8	The fee_collector constraint prevents the unlocker from claiming without a fee	Info	Fixed

5.2.1 Findings Raised During Fix Review Phase

	Finding	Severity	Update
1	The `set_default_fee_collector` instruction cannot be executed	Medium	Fixed
2	Redundant check	Info	Fixed

6 System Overview

TokenTable introduces two new Solana programs which work in tandem as an independent on-chain system to provide users with token distribution capabilities:

a) **Unlocker** program – used to store distributed assets and all the records necessary for distribution.

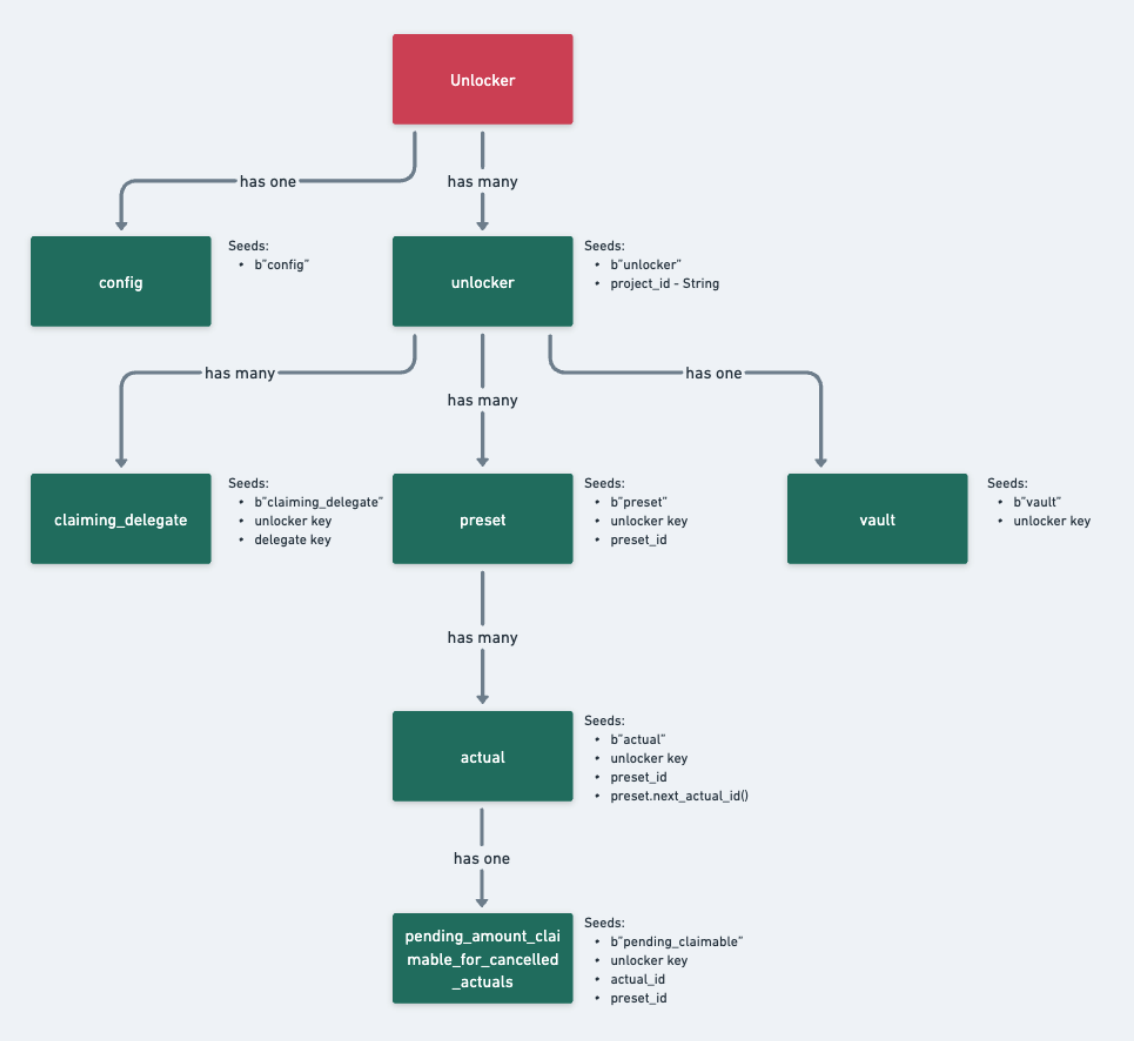
b) **Fee Collector** program – responsible for collecting protocol fees.

6.1 Unlocker

In the Unlocker program, the owner of the protocol creates an `unlocker` account a.k.a. Project (the protocol is not permissionless) that is a basis for a redistribution of a single asset. The following points describe a high-level view of a Project:

1. The protocol owner creates a Project (represented by a unique `unlocker` account). Permission is granted to that Project to a user.
2. The user creates Presets that represent the distribution schedule and Actuals that define recipients and their amounts.
3. A Recipient can claim the tokens accordingly.

The following picture presents the account structure of the Unlocker program:



The section below outlines the programs' changes to selected instructions mad which were in the scope of this audit. For full description of the protocol, consult the first audit document.

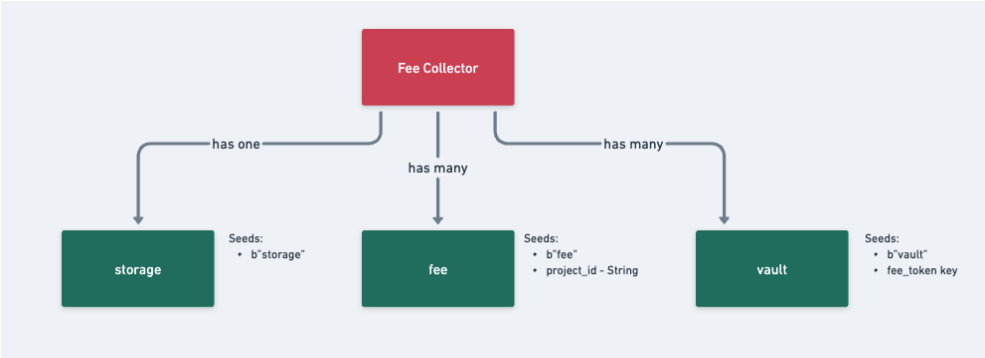
Unlocker's instruction updates:

1. `cancel` – Changed the contents of `pending_amount_claimable_for_cancelled_actuals` account and how its values are updated.
2. `claim_cancelled_actual_tokens` – **New Instruction** – Used to claim tokens of a canceled Actual. This can be done either by a delegatee or the original recipient of the cancelled Actual. Added option to claim by a delegatee.
3. `claim` – Removed the `pending_amount_claimable_for_cancelled_actuals` account and code related to handling it, which is now moved to the `claim_cancelled_actual_tokens` instruction.
4. `deposit` – Added Token program validation.
5. `disable_cancel` – Added check if `unlocker.is_cancellable` is already set, throws error instead of log.
6. `disable_create` – Added check if `unlocker.is_creatable` is already set, throws error instead of log.
7. `disable_transfer_actual` – Added check if `unlocker.is_transferrable` is already set, throws error instead of log.
8. `disable_withdrawable` – Added check if `unlocker.is_withdrawable` is already set, throws error instead of log.
9. `initialize` – Added `init_fee_account` argument to the instruction. If set to true initializes the the project's `fee` account by calling `init_project_fee()` on the Fee Collector program with the necessary accounts.
10. `receive_program_admin` – **New Instruction** – Manages the takeover of the ownership in a two-step way.
11. `set_fee_collector` – **New Instruction** – Initializes the the project's `fee` account by calling `init_project_fee()` on the Fee Collector program with necessary accounts. It works only if the `fee` wasn't initialized during `initialize()`.
12. `transfer_program_admin` – Instruction doesn't directly transfer ownership of the program, but updates the `new_admin` for two-step process.
13. `withdraw_deposit` – Added Token program validation.

6.2 Fee Collector

The Fee Collector program is controlled only by the protocol owner. It is used to create specific `fee` accounts that hold fee rates for each Project. This program is also responsible for the correct calculation and collection of fees. The fees collected are stored in `vault` accounts owned by the program. Protocol owner can withdraw collected fees.

The following picture presents the account structure of the Fee Collector program:



The `storage` and `config` accounts are used to store the protocol's owner account and other global settings.

Fee Collector's instruction updates:

1. `init_project_fee` – **New Instruction** – Creates the `fee` account for the project and sets `fee.default` to true.
2. `collect_fee` – Changed default fee selection condition, which now is based on the `fee.default` value.
3. `withdraw` – Added Token program validation.
4. `get_fee` – Removed the `fee` account initialization.
5. `init_fee_token` – Added Token program validation.
6. `receive_ownership` – **New Instruction** – Manages the takeover of the ownership in a two step way.
7. `set_custom_fee_bips` – Disables the default fee.
8. `set_custom_fee_fixed` – Disables the default fee.

7 Issues

7.1 [Low] Rent is refunded to the wrong address when the `pending_amount_claimable_for_cancelled_actuals` account is closed

File(s): `claim_cancelled_actual_tokens.rs`

Description: The `pending_amount_claimable_for_cancelled_actuals` account is created in the `cancel` instruction, with rent paid by `unlocker.owner`, and is closed in the `claim_cancelled_actual_tokens` instruction, with rent mistakenly refunded to the `receipt` address.

```
#[derive(Accounts)]
#[instruction(_project_id: String, _preset_id: u64, _actual_id: u64)]
pub struct ClaimCancelledActualTokens<'info> {
  //...
  #[account(
    mut,
    seeds = [
      b"pending_claimable".as_ref(),
      unlocker.key().as_ref(),
      _preset_id.to_le_bytes().as_ref(),
      _actual_id.to_le_bytes().as_ref(),
    ],
    bump,
    close = recipient // NOTE: We are closing the account here.
  )]
  pub pending_amount_claimable_for_cancelled_actuals: Box<
    Account<'info, PendingAmountClaimableForCancelledActualsAccount>
  >,
  //...
}
```

Impact: The `recipient` address will receive the additional rent that should have been refunded to `unlocker.owner`.

Recommendation(s): When closing the `pending_amount_claimable_for_cancelled_actuals` account, the rent should be refunded to `unlocker.owner`.

Status: Fixed

Update from TokenTable: Funds are now returned to `unlocker.owner` in `a71216f80e80fe9c1e6c62ce8d786d3522a572f7`.

7.2 [Low] The `_preset_is_empty` function should not consider `num_of_unlocks_for_each_linear`

File(s): `create_actual.rs`

Description: When `preset.stream` is true, the length of `preset.num_of_unlocks_for_each_linear` is not strictly limited to save rent. Therefore, in a successfully created `preset` account, `preset.num_of_unlocks_for_each_linear` may be 0. As a result, when `_preset_is_empty` checks whether the `preset` account is initialized, it should not consider the `num_of_unlocks_for_each_linear` field.

```
#[allow(unused_parens)] // Allowing unused_parens to ignore Prettier formatting
fn _preset_is_empty(preset: anchor_lang::prelude::Account<'_, PresetAccount>) -> bool {
  return (
    preset.linear_bips.len() *
      preset.linear_start_timestamps_relative.len() *
      preset.num_of_unlocks_for_each_linear.len() *
      (preset.linear_end_timestamp_relative as usize) == 0
  );
}
```

Impact: An already initialized `preset` account may be mistakenly considered uninitialized, preventing the creation of its corresponding `actual` account.

Recommendation(s): In the `_preset_is_empty` function, the `num_of_unlocks_for_each_linear` field should not be considered when `preset.stream` is true.

Status: Fixed

Update from TokenTable: We now take `preset.stream` into account when determining if a preset is empty in `b98faf703c27399307598bcb21bf6f647fc143bf`.

7.3 [Low] The creation of `pending_amount_claimable_for_cancelled_actuals` account may lead to rent loss

File(s): `cancel.rs`

Description: The `pending_amount_claimable_for_cancelled_actuals` account is always created with rent paid by `unlocker.owner`. However, when `should_wipe_claimable_balance` is true or `delta_amount_claimable` is 0, the account cannot be closed in the `claim_cancelled_actual_tokens` instruction.

```
fn _claim_pending_amount(
  ctx: Context<ClaimCancelledActualTokens>,
  project_id: String,
  actual_id: u64,
  batch_id: u64
) -> Result<()> {
  let delta_amount_claimable =
    ctx.accounts.pending_amount_claimable_for_cancelled_actuals.pending_amount_claimable_for_cancelled_actuals;
  require!(delta_amount_claimable != 0, TokenTableError::NotClaimable);
  // ...
}
```

Impact: The `pending_amount_claimable_for_cancelled_actuals` account cannot be closed, causing the rent to be locked.

Recommendation(s): It is recommended not to create the `pending_amount_claimable_for_cancelled_actuals` account when `should_wipe_claimable_balance` is true and to directly close the account when `delta_amount_claimable` is 0.

Status: Fixed

Update from TokenTable: In `9f97ad413026962fd0d544be537d634aebba401c`, automatically close `pending_amount_claimable_for_cancelled_actuals` account in `cancel()` if `should_wipe_claimable_balance` is true or `delta_amount_claimable` is 0. Also, allow `claim_cancelled_actual_tokens()` to close the `pending_amount_claimable_for_cancelled_actuals` account if `delta_amount_claimable` is 0 and the account already exists.

7.4 [Info] Allow the `fee_collector` to be set arbitrarily during the initialization of the unlocker account

File(s): `initialize.rs`

Description: In the `initialize` instruction, if `init_fee_account` is set to `false`, then the check `ctx.accounts.fee_collector.as_ref().unwrap().key() == fee_collector.key()` is skipped. This means that it allows the `unlocker` owner to initialize any `fee_collector`.

```
pub fn initialize(...) -> Result<()> {
  //...
  ctx.accounts.unlocker.fee_collector = fee_collector;

  if init_fee_account {
    // Before we init the fee account, ensure we are calling the expected fee_collector program from
    // the parameters and that all required accounts are provided.
    require!(
      ctx.accounts.fee_collector.is_some() &&
        ctx.accounts.fee.is_some() &&
        ctx.accounts.fee_collector_storage.is_some() &&
        ctx.accounts.fee_collector.as_ref().unwrap().key() == fee_collector.key(),
      TokenTableError::InvalidFeeCollector
    );
```

Impact: In the current system, allowing the `fee_collector` account to be set arbitrarily during initialization does not cause any loss, because the no-fee claim, as designed by the protocol, fails due to a constraint in the `ctx`. However, the `unlocker.owner` may have the motivation to initialize the `fee_collector` as `pubkey::default` during initialization. This would enable claims related to that account to be processed without any fees.

Recommendation(s): It is recommended not to allow the `unlocker.owner` to arbitrarily initialize the `fee_collector`.

Status: Fixed

Update from TokenTable: In `aa48e8ab3c30b65f0e90a3be35cdb81a7f7f9461`, `fee_collector` program account verification is handled manually. Anchor now expects an `UncheckedAccount<>`, and in all instructions where `fee_collector` can be set, we verify that the provided account matches the instruction parameter value and that the provided account is executable.

7.5 [Info] Changing the `fee_collector` to a different program will cause instructions to fail

File(s): `set_fee_collector.rs`

Description: The `set_fee_collector` instruction allows to set a different `fee_collector` program for Unlocker's fee processing capabilities, as per TokenTable's feedback from the previous audit:

>Added the ability to change the fee_collector for a project rather than adding verification in a80d3c31d. We may need to change this address at some point in the future. This function is only callable by an admin (read: one of our wallet accounts), so errors should not happen in setting these values, and we would be able to fix any errors if need be.

The problem arises when the Fee Collector `program_id` is changed and certain instructions which take the `fee_collector` program account are called. The Anchor implementation under the hood will validate the program account against the `program_id` of the `FeeCollector` which was placed there at compile time:

```
pub fee_collector: Option<Program<'info, FeeCollector>>,
```

Impact: It will not be possible to update the `fee_collector` program account without also updating the entire Unlocker program.

Recommendation(s): Remove the `fee_collector` account from Anchor's context structs and handle it manually within the instruction code.

Status: Fixed

Update from TokenTable: In `aa48e8ab3c30b65f0e90a3be35cdb81a7f7f9461`, `fee_collector` program account verification is handled manually. Anchor now expects an `UncheckedAccount<>`, and in all instructions where `fee_collector` is used, we verify that the provided account matches the expected unlocker's/airdrop's `fee_collector` but skip the account executable check, since this would have already been checked when the account was set.

7.6 [Info] Lack of `Option` wrapper on `fee` account

File(s): `claim.rs` `claim_cancelled_actual_tokens.rs`

Description: Both claiming instructions - `claim` and `claim_cancelled_actual_tokens` are invoked with few accounts related to fee collection:

- `authority_fee_ata`;
- `fee_collector_storage`;
- `fee_collector_vault`;
- `fee_collector`;
- `fee_token_mint`;
- `fee`;
- `fee_token_program`;

The fee collection mechanism is optional, hence, the design allows skipping them if they are unnecessary through the `Option` wrapper on the account type in the `instructions` contexts.

The `fee` account however is not:

```
/// CHECK: The account is checked in the FeeCollector, not here.
#[account(mut)]
pub fee: UncheckedAccount<'info>,
```

Impact: Expected difficulties in building the fee-less transactions as the `fee` account still needs to be provided to the instruction call.

Recommendation(s): Wrap the `fee` account type in `Option`.

Status: Fixed

Update from TokenTable: As of `78051afb53579a4e6558519000d6c35f510a5533`, the fee collection mechanism is no longer optional. `fee` is a required account and the documented structure here is needed to support the updated fee collection mechanism.

7.7 [Info] Redundant code

File(s): `utils.rs`, `collect_fee.rs`, `get_fee.rs`

Description: The protocol contains multiple pieces of redundant code.

1. When obtaining the special configuration `fee_bips` if `fee_bips` equals `BIPS_PRECISION`(10000) it will be set to 0;

```
pub fn collect_fee(
  ctx: Context<CollectFee>,
  fee_token: Pubkey,
  _project_id: String,
  token_transferred: u64
) -> Result<u64> {
    //...
    let mut fee_bips = ctx.accounts.fee.bips;

    if fee_bips == BIPS_PRECISION {
      fee_bips = 0;
    }
```

   However, this logic is redundant because `fee_bips` is already restricted to not exceed `MAX_FEE`(1000) during configuration.

2. In the `claim()` instruction of the Unlocker program it is imperative to validate the provided `fee_collector` program account if it matches the one held within the `unlocker` account. This is done directly within the `claim()` inside `claim.rs` file:

```
if ctx.accounts.unlocker.fee_collector != Pubkey::default() {
  require!(
    ctx.accounts.unlocker.fee_collector == ctx.accounts.fee_collector.key(),
    TokenTableError::InvalidFeeCollector
  );
}
```

   Later in the code, `_claim()` is called, which calls `_after_claim()`, which calls `_charge_fees()`. Inside it we can find the same unnecessary validation:

```
require!(
  fee_collector.key() == storage.fee_collector.key(),
  TokenTableError::UnsupportedOperation
);
```

   Where the `storage` is in fact the `unlocker` account. What is more the Airdrop program, does not contain such a redundant check in its `_charge_fees()` equivalent. It is recommended to remove the check from the `_charge_fees()` function.

Impact: Redundant code hinders readability and increases deployment costs.

Recommendation(s): It is recommended to optimize the redundant code.

Status: Fixed

Update from TokenTable:

Redundant code for checking `fee_collector` removed in `78051afb53579a4e6558519000d6c35f510a5533`,
validating fees removed in `8c443f729b4eefd491832bcac71d996c317f8252`,

7.8 [Info] The fee_collector constraint prevents the unlocker from claiming without a fee

File(s): `claim_cancelled_actual_tokens.rs`, `claim.rs`

Description: The following code indicates that when `unlocker.fee_collector` is set to `pubkey::default()`, the claim will not incur any fees.

```
// Fee collector
if ctx.accounts.unlocker.fee_collector != Pubkey::default() {
  require!(
    ctx.accounts.unlocker.fee_collector == ctx.accounts.fee_collector.key(),
    TokenTableError::InvalidFeeCollector
  );
}
```

```
pub fn _charge_fees<'info>(...) -> Result<u64> {
  let mut fee_collected: u64 = 0;
  if storage.fee_collector != Pubkey::default() {
    //...
  }

  Ok(fee_collected)
}
```

However, the `fee_collector` constraint in the `claim` and `claim_cancelled_actual_tokens` instructions causes the `unlocker.fee_collector` to fail the constraint check if it is set to `pubkey::default()`, thus preventing the fee-less claim from passing.

```
#[account(constraint = fee_collector.key() == unlocker.fee_collector.key())]
pub fee_collector: Program<'info, FeeCollector>,
```

Impact: The code's intended fee-less claim cannot be achieved.

Recommendation(s): Remove the `fee_collector` constraint.

Status: Fixed

Update from TokenTable: As of `78051afb53579a4e6558519000d6c35f510a5533`, the fee collection mechanism is no longer optional. In `aa48e8ab3c30b65f0e90a3be35cdb81a7f7f9461`, `fee_collector` is an `UncheckedAccount<>` with manual account verification checks.

Findings Raised During Fix Review Phase

[Medium] The `set_default_fee_collector` instruction cannot be executed

File(s): `set_fee_collector.rs`

Description: The `set_default_fee_collector` instruction is used to modify the `default_fee_collector`. Since it requires `config.admin` for permission validation, the `config` account should have already been initialized when calling the instruction. However, due to the incorrect assignment of the `init` attribute to the `config` account in the `ctx`, the `set_default_fee_collector` instruction fails to execute successfully.

```
#[derive(Accounts)]
#[instruction(_default_fee_collector: Pubkey)]
pub struct SetDefaultFeeCollector<'info> {
  #[account(
    init,
    seeds = [b"config".as_ref()],
    bump,
    payer = authority,
    space = 8 + Config::INIT_SPACE
  )]
  pub config: Account<'info, Config>,
  //...
}
```

Impact: The default_fee_collector cannot be successfully set

Recommendation(s): It is recommended to remove the init attribute from the config account in the ctx.

Status: Fixed

Update from TokenTable: Removed `init` attribute from the config account in the Anchor context in e2cf5fbc8802845c56d0e0ab48c874c0000ce015 and added `mut` attribute in 8edb2ab7e2a63c37258b78f365bce2d43db3403f.

[Info] Redundant check

File(s): claim.rs, claim_cancelled_actual_tokens.rs

Description: The claim and claim_cancelled_actual_tokens instructions contain redundant checks for fee_collector. The fee_collector is checked in the ctx and then checked again in the execution logic.

```
  /// CHECK: Checked in the function call.
#[account(constraint = fee_collector.key() == airdrop.fee_collector.key())]
pub fee_collector: UncheckedAccount<'info>,

// ...

pub fn claim_cancelled_actual_tokens(...) -> Result<()> {
  // Fee collector
  require!(
    ctx.accounts.unlocker.fee_collector == ctx.accounts.fee_collector.key(),
    TokenTableError::InvalidFeeCollector
  );

pub fn claim(...) -> Result<()> {
  // Fee collector
  require!(
    ctx.accounts.unlocker.fee_collector == ctx.accounts.fee_collector.key(),
    TokenTableError::InvalidFeeCollector
  );
```

Impact: Redundant checks increase the execution overhead of the transaction call.

Recommendation(s): It is recommended to remove the redundant checks.

Status: Fixed

Update from TokenTable: Redundant checks removed in 1aed8dad5fca73dd7e7b3d2a666c939a39a37be6.

8 Evaluation of Provided Documentation

The TokenTable team provided documentation in two forms:

- **Official Documentation Website:** The official documentation contains the protocol's design and implementation details, providing an overview of the protocol's purpose for both users and auditors. Unfortunately, the current state of the documentation website does not contain a version for Solana contracts.
- **Natspec Comments:** The code includes comments for key processes to help understand the logic. However, most functions lack comments, and expanding documentation coverage would enhance the overall comprehensibility of the code.

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

```
> anchor build
   Compiling fee-collector v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fee-collector)
   Compiling merkle-token-distributor-solana v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/merkle-token-distributor-solana)
    Finished `release` profile [optimized] target(s) in 3.75s
   Compiling fee-collector v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fee-collector)
   Compiling merkle-token-distributor-solana v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/merkle-token-distributor-solana)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 3.29s
   Compiling fee-collector v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fee-collector)
    Finished `release` profile [optimized] target(s) in 1.32s
   Compiling fee-collector v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fee-collector)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.93s
   Compiling fee-collector v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fee-collector)
   Compiling unlocker-v2-solana v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/unlocker-v2-solana)
    Finished `release` profile [optimized] target(s) in 3.43s
   Compiling fee-collector v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fee-collector)
   Compiling unlocker-v2-solana v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/unlocker-v2-solana)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 2.33s
   Compiling fee-collector v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fee-collector)
   Compiling fungible-token-distributor-solana v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fungible-token-distributor-solana)
    Finished `release` profile [optimized] target(s) in 2.79s
   Compiling fee-collector v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fee-collector)
   Compiling fungible-token-distributor-solana v0.1.0
    ↪  (/tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/programs/fungible-token-distributor-solana)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 1.71s
```

9.2 Tests Output

Fee Collector's test output:

```
yarn run v1.22.22
$ /tmp/011-TokenTable-Solana-UnlockerV2/node_modules/.bin/ts-mocha -p ./tsconfig.json -t 1000000
 ↪  tests/_fee_collector.ts


  fee-collector init
     Is initialized! (461ms)
     Set Default Fee (475ms)
     Collect Fee (lamports) (464ms)
     Collect Fee (SPL) (945ms)
     Withdraw (fail - not owner) (471ms)
     Withdraw (succeed) (460ms)


  6 passing (5s)

  Done in 6.73s.
```

Unlocker's test output:

```
yarn run v1.22.22
$ /tmp/011-TokenTable-Solana-UnlockerV2-FollowUp-Merkle/code/unlocker/node_modules/.bin/ts-mocha -p ./tsconfig.json -t
↪  1000000 'tests/**/*.ts' test_merkle


 fee-collector init
custom
  Fungible
    Core
      should initialize correctly (241.246958ms)
      base params set (40.85375ms)
      fee collector set (42.731041ms)
      successful claim (54.477125ms)
      verify (37.703834ms)
    Core (417.749ms)
  Fungible (417.855625ms)
custom (418.01575ms)
token-table-unlocker-v2-solana
  Unlocker
    Core
      should initialize correctly (33.888042ms)
      should allow transfer of program admin (38.471708ms)
      should let program admin change fee collector (37.721583ms)
      should let project owner transfer ownership (33.146333ms)
      should create a preset and enforce permissions (38.317792ms)
      should create an actual and enforce permissions, no skipping (46.346375ms)
      should forbit creating new actual if create is disabled (34.366916ms)
      should manage transfering an actual (43.826458ms)
      should let founder withdraw deposit and enforce permissions (47.706792ms)
      should calculate the correct claimable amount
        no skip
          key timestamps (45.984042ms)
          random timestamps (121.61775ms)
        no skip (167.745459ms)
          Random amount skipped: 6382
        random skip
          key timestamps (50.323417ms)
          Random amount skipped: 4356
          random timestamps (105.707208ms)
        random skip (156.171042ms)
      should calculate the correct claimable amount (324.010542ms)
      should let investor claim the correct amount (64.714708ms)
      should let founders or cancelables cancel and refund the correct amount (54.197792ms)
      should let investor claim the correct amount (two projects) (63.509791ms)
      should let delegate claim the correct amount (69.9045ms)
      Fee Collector (lamports) (49.109917ms)
      Fee Collector (SPL) (52.100458ms)
    Core (1037.428333ms)
  Unlocker (1037.522666ms)
token-table-unlocker-v2-solana (1037.592958ms)
    Is initialized! (975ms)
    Transfer ownership (1948ms)
    Set Default Fee (489ms)
    Collect Fee (lamports) (497ms)
    Collect Fee (SPL) (986ms)
    Withdraw (fail - not owner) (495ms)
    Withdraw (succeed) (482ms)
```

9.3 Notes about Test suite

The TokenTable team delivered a rather comprehensive test suite, showcasing a well-structured approach to ensuring the protocol's correctness and resilience. Key suggestions of the test suite include:

- **Missing Functionality:** There are certain instructions whose validation is not currently included in the test suite, e.g. `disable_cancel`, `disable_withdraw`. CODESPECT recommends adding them to the test suite. Additionally new functionality and new instructions should be included in the test suite.
- **Edge Cases:** Beyond basic operations, the test suite should cover some basic edge cases specific to values which can be provided by the users of the protocol. One example would be adding a test case to verify the smallest possible time difference between consecutive claims.
- **Fee Collector:** The Fee Collector program tests should include tests for all fee options, i.e. BIPS, fixed fee, and default fee.

Overall, the test suite reflects a mature development process and significantly enhances the reliability of the protocol.

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

CODESPECT also recommends explicitly adding tests involving both token programs, Token and Token2022, to ensure seamless integration with both programs.