

# Saakuru Labs SKR Token Security Audit Report

# Contents

Contents	1
Executive Summary	2
Project Overview	2
Audit Scope	3
Audit Methodology	4
Findings Summary	6
Findings	7
Conclusion	12
Disclaimer	12

# Executive Summary

Title	Description
Client	Saakuru Labs
Project	skr-token
Platform	Ethereum
Language	Solidity
Repository	<a href="https://github.com/Saakuru-Open-Source/skr-token">https://github.com/Saakuru-Open-Source/skr-token</a>
Initial commit	2941019f95b3e246ee4976551087ff6af73df180, 801fde8ab53ad4cc860693caa081427eaab6020a
Final commit	086bc59773ac9e84ef8810ab24264d547b176c77
Timeline	April 16 2023 - April 18 2024

## Project Overview

The Saakuru token is a multi-purpose (utility & governance) token of the Saakuru Protocol with a burning mechanism that constantly reduces the total circulating supply.

# Audit Scope

File	Link
LERC20.sol	<a href="#"><u>LERC20.sol</u></a>
SKR.sol	<a href="#"><u>SKR.sol</u></a>
SKRBridged.sol	<a href="#"><u>SKRBridged.sol</u></a>

# Audit Methodology

## General Code Assessment

The code is reviewed for clarity, consistency, style, and whether it follows code best practices applicable to the particular programming language used, such as indentation, naming convention, commented code blocks, code duplication, confusing names, irrelevant or missing comments, etc. This part is aimed at understanding the overall code structure and protocol architecture. Also, it seeks to learn overall system architecture and business logic and how different parts of the code are related to each other.

## Code Logic Analysis

The code logic of particular functions is analyzed for correctness and efficiency. The code is checked for what it is intended for, the algorithms are optimal and valid, and the correct data types are used. The external libraries are checked for relevance and correspond to the tasks they solve in the code. This part is needed to understand the data structures used and the purposes for which they are used. At this stage, various public checklists are applied in order to ensure that logical flaws are detected.

## Entities and Dependencies Usage Analysis

The usages of various entities defined in the code are analyzed. This includes both: internal usage from other parts of the code as well as possible dependencies and integration usage. This part aims to understand and spot overall system architecture flaws and bugs in integrations with other protocols.

## Access Control Analysis





Access control measures are analyzed for those entities that can be accessed from outside. This part focuses on understanding user roles and permissions, as well as which assets should be protected and how.

## Use of checklists and auditor tools



Auditors can perform a more thorough check by using multiple public checklists to look at the code from different angles. Static analysis tools (Slither) help identify simple errors and highlight potentially hazardous areas. While using Echidna for fuzz testing will speed up the testing of many invariants, if necessary.

## Vulnerabilities





The audit is directed at identifying possible vulnerabilities in the project's code. The result of the audit is a report with a list of detected vulnerabilities ranked by severity level:



Severity	Description
 <b>Critical</b>	Vulnerabilities leading to the theft of assets, blocking access to funds, or any other loss of funds.
 <b>High</b>	Vulnerabilities that cause the contract to fail and that can only be fixed by modifying or completely replacing the contract code.
 <b>Medium</b>	Vulnerabilities breaking the intended contract logic but without loss of funds and need for contract replacement.
 <b>Low</b>	Minor bugs that can be taken into account in order to improve the overall quality of the code

After the stage of bug fixing by the Customer, the findings can be assigned the following statuses:

Status	Description
 <b>Fixed</b>	Recommended fixes have been made to the project code and no longer affect its security.
 <b>Acknowledged</b>	The Customer took into account the finding. However, the recommendations were not implemented since they did not affect the project's safety.

# Findings Summary

Severity	# of Findings
 Critical	0
 High	0
 Medium	0
 Low	5

ID	Severity	Title	Status
L-1	 Low	Pragma solidity version	Acknowledged
L-2	 Low	No checking of input variables	Fixed
L-3	 Low	No updating state for <code>recoveryAdminKeyHash</code>	Fixed
L-4	 Low	There are no limits and no setter for the <code>timelockPeriod</code> variable	Fixed
L-5	 Low	There are no limits for the <code>amount</code> variable	Acknowledged

# Findings

## Critical

Not Found

## High


Not Found

## Medium

Not Found

## Low

L-1

 Low

Pragma solidity version

Acknowledged

**Description**

[hardhat.config.ts#L36](#)  
[SKR.sol#L2](#)  
[LERC20.sol#L21](#)

The Solc version specified in the configuration is 0.8.2, which is outdated.

**Recommendation**

We recommend setting the latest stable version of the Solidity compiler and lock the pragma.



L-2

Low

No checking of input variables

Fixed

### Description

- `recipient:`  
[LERC20.sol#L205](#)  
Users can lose their funds if they enter their addresses incorrectly.
- `admin:`  
[LERC20.sol#L30](#)
- `recoveryAdmin:`  
[LERC20.sol#L31](#)

### Recommendation

We recommend adding null address checks.

### Client's commentary

Fixed in [801fde8a](#).

L-3

 LowNo updating state for `recoveryAdminKeyHash`

Fixed

**Description**[LERC20.sol#L111](#)

The initial state for `recoveryAdminKeyHash` is `""`.

The initial state for `recoveryAdminCandidate` is `address(0)`.

After the `acceptRecoveryAdminOwnership` function, we should reset the state for the `recoveryAdminCandidate` and `recoveryAdminKeyHash` variables.

**Recommendation**

We recommend adding `recoveryAdminKeyHash = ""` to the `acceptRecoveryAdminOwnership` function.

**Client's commentary**

Fixed in [801fde8a](#).

L-4

● Low

There are no limits and no setter for the `timelockPeriod` variable

Fixed

#### Description

[LERC20.sol#L34](#)

The `timelockPeriod` variable is set once and cannot be changed.

#### Recommendation

We recommend adding limits for the `timelockPeriod` variable for initialization and a setter for this function with an `onlyOwner` modifier.

#### Client's commentary

Fixed in [801fde8a](#).

L-5

 LowThere are no limits for the `amount` variable

Acknowledged

**Description**[SKRBridged.sol#L49](#)

The `mint` function doesn't have limits for the `amount` variable.

**Recommendation**

We recommend adding limits for the `amount` variable.

**Client's commentary**

This is a bridged token contract, binded to the SKR token deployed on the Saakuru Chain which has the limited supply of 1bIn tokens. As the cross-chain supply is dynamic, tokens will be minted and burned through the bridge.

<https://bscscan.com/address/0x21c1e74caadf990e237920d5515955a024031109#code>.

Standard practice for bridged tokens: <https://docs.openzeppelin.com/contracts/4.x/crosschain>.

## Conclusion

Altogether, the audit process has revealed 5 LOW severity findings.

## Disclaimer

The Stronghold audit makes no statements or warranties about the utility of the code, the safety of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug-free status. The audit documentation is for discussion purposes only.