RAILGUN_

RAILGUN_

SMART CONTRACT AUDIT

zokyo

April 20th, 2022 | v. 1.0

Security Audit Score

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

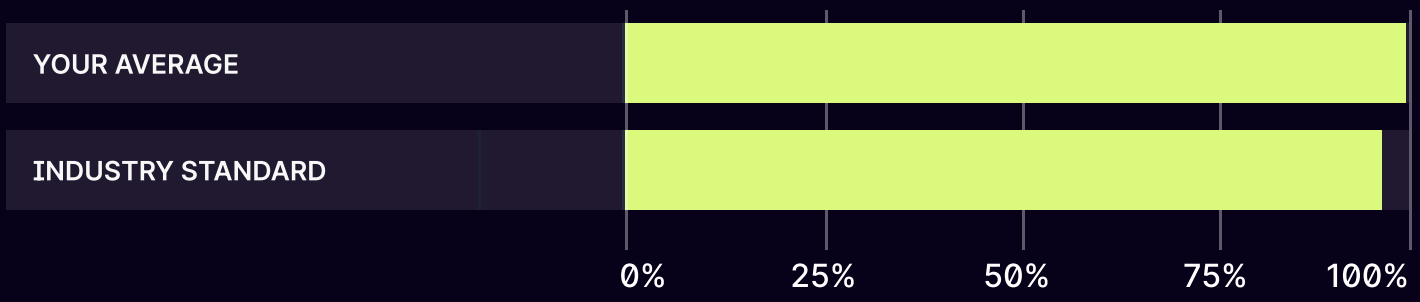
TECHNICAL SUMMARY

This document outlines the overall security of the Railgun smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Railgun smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code is significant for the ensurance of contracts security.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a security of the contract we at Zokyo recommend that the Railgun put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

Rhe Railgun smart contract's source code was taken from the repository provided by Railgun: https://github.com/railgun-privacy/contract

Audited commit: 62401e1a64b9af968d85b8f5347f1a72cc56862c

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- Globals.sol
- Commitments.sol
- Poseidon.sol
- RailgunLogic.sol
- Snark.sol
- Verifier.sol
- RokenBlacklist.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Ream has followed best practices and industry-standard techniques to verify the implementation of Railgun smart contracts. Ro do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.
04	Thorough, manual review of the codebase, line by line.

Executive Summary

Railgun set of contracts represents functionality for the private balances keeping empowered with Merkle trees and snarks verifications.

Audited contracts are well-structured, have very high code quality and are completely documented. There were no critical issues found. The functionality is sufficiently covered with unit-tests, which were also verified by auditors team. Auditors have carefully checked business logic, access control, verification process, tokens usage and other security factors. Few issues present in the report are connected to the Solidity best practices and do not influence contract security.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

INFORMATIONAL | UNRESOLVED

Variables lacks validation.
RailgunLogic.sol: function changeFee().
Percentage fee variables should be verified not to exceed 100%.

Recommendation:
Verify that new fee variables are less than “BASIS_POINTS”.

INFORMATIONAL | UNRESOLVED

Use enum instead of uint.
Globals.sol: line.
Variables “tokenType” from struct “TokenData”, “withdraw” from struct “BoundParams” are supposed to be assigned to a restricted range of numbers. To ensure that values of these variables are not out of bounds, a enum should be used as a set of constants, assigned to numbers, starting from 0.

Recommendation:
Define enum for each variable instead of using uint8.

	Globals.sol	Commitments.sol	Poseidon.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	RailgunLogic.sol	Snark.sol	Verifier.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	TokenBlacklist.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Railgun team

As part of our work assisting Railgun team in verifying the correctness of their contract code, our team has checked the complete set of unit tests prepared by the Predy team.

It needs to be mentioned, that the original code has a significant original coverage with testing scenarios provided by the Railgun team. All of them were also carefully checked by the auditors team to be consistent and meaningful.

Logic/Commitments

- ✓ Should calculate zero values (49ms)
- ✓ Should calculate empty root
- ✓ Should hash left/right pairs
- ✓ Should incrementally insert elements (2957ms)
- ✓ Should roll over to new tree

Logic/RailgunLogic

- ✓ Should change treasury
- ✓ Should change fee (49ms)
- ✓ Should calculate fee (1137ms)
- ✓ Should calculate token field
- ✓ Should hash note preimages (232ms)
- ✓ Should deposit ERC20 (762ms)
- ✓ Should transfer ERC20 (30167ms)

Logic/Snark

- ✓ Should reject invalid proofs (549ms)
- ✓ Should accept valid proofs (539ms)
- ✓ Should reject points outside scalar field

Logic/TokenBlacklist

- ✓ Should add and remove from blacklist (84ms)
- ✓ Should emit blacklist events (54ms)

Logic/Verifier

- ✓ Should set verifying key (68ms)
- ✓ Should hash bound parameters (92ms)
- ✓ Should verify dummy proofs (5251ms)
- ✓ Should verify proofs
- ✓ Should throw error if circuit artifacts don t exist

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	Uncovered Lines
logic\	83.66	46.43	88.46	86.5	
Commitments.sol	92.31	87.5	75	92.31	223,228,231
Globals.sol	100	100	100	100	
Poseidon.sol	100	100	0	100	
RailgunLogic.sol	71.57	32.14	100	76.29	... 386,390,396
Snark.sol	100	66.67	100	100	
TokenBlacklist.so	100	100	100	100	
Verifier.sol	95	50	100	95	139
All files	**83.66**	**46.43**	**88.46**	**86.5**	

We are grateful to have been given the opportunity to work with Railgun.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that Railgun put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo