RAILGUN

AUDIT REPORT

Railgun Privacy Contract

The following audit report confirms:

✓A. Railgun has no way of taking funds
✓B. Railgun is effective at preserving privacy
✓C. All bugs have been successfully fixed in this latest version

About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit.

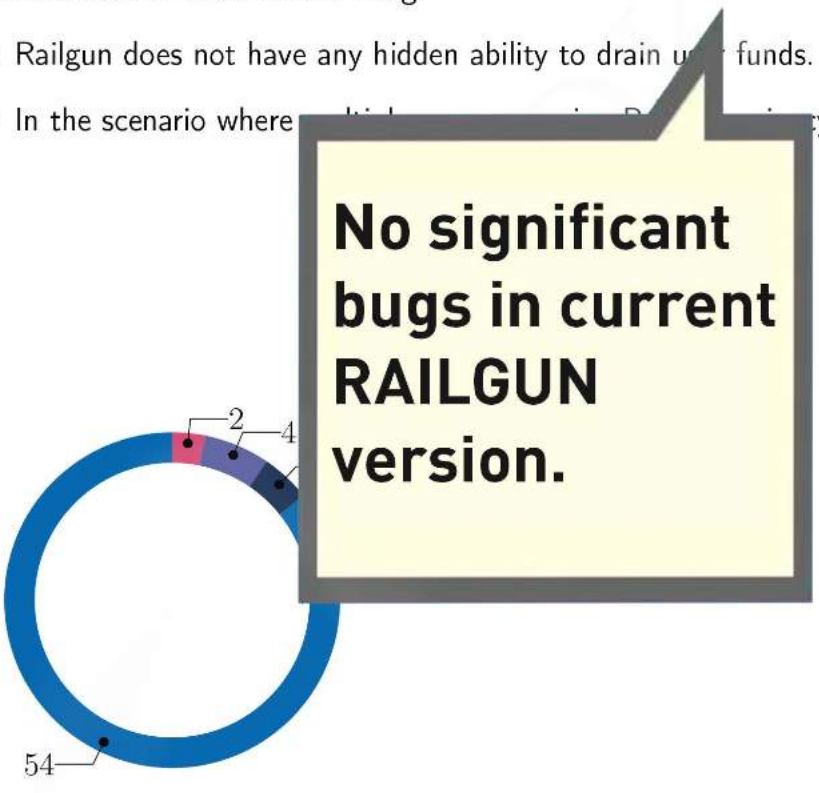
SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
2d July 2021

We were asked to review Railgun smart contracts and circuits. We evaluated the code for its utility as a tool to send Ethereum assets, receive Ethereum assets, and interact with other smart contracts, without revealing the origin address of the user. In the earlier versions we have found several major issues, but they were all fixed in the final version.

To the best of auditors knowledge:

- Railgun does not have any hidden ability to drain u[illegible] funds.
- In the scenario where [illegible]y is preserved for users.



SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
2d July 2021

We were asked to review Railgun smart contracts and circuits. We evaluated the code for its utility as a tool to send Ethereum assets, receive Ethereum assets, and interact with other smart contracts, without revealing the origin address of the user. In the earlier versions we have found several major issues, but they were all fixed in the final version.

To the best of auditors knowledge:

- Railgun does not have any hidden ability to drain user funds.
- In the scenario where multiple users are using Railgun, privacy is preserved for users.

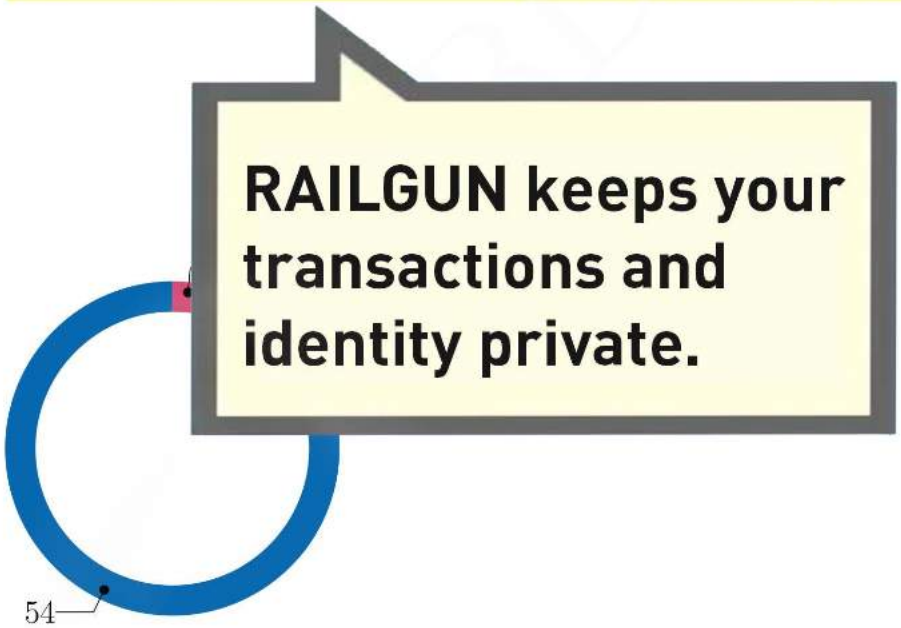
SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
2d July 2021

We were asked to review Railgun smart contracts and circuits. We evaluated the code for its utility as a tool to send Ethereum assets, receive Ethereum assets, and interact with other smart contracts, without revealing the origin address of the user. In the earlier versions we have found several major issues, but they were all fixed in the final version.

To the best of auditors knowledge:

- Railgun does not have any hidden ability to drain user funds.
- In the scenario where multiple users are using Railgun, privacy is preserved for users.



ABDK CONSULTING

SMART CONTRACT AUDIT

Railgun

Circom and Solidity

abdk.consulting

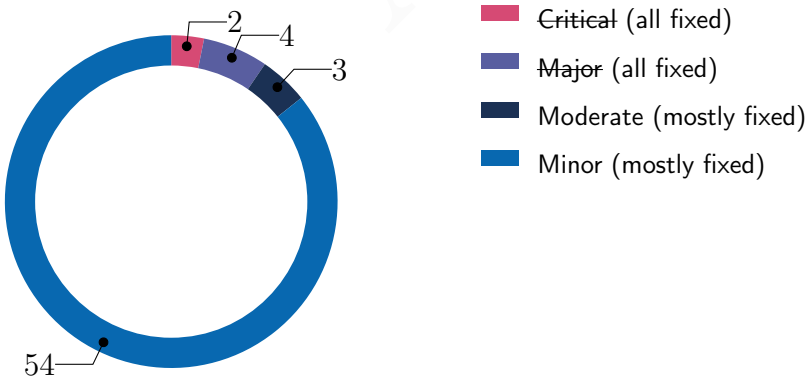
SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
2d July 2021

We were asked to review Railgun smart contracts and circuits. We evaluated the code for its utility as a tool to send Ethereum assets, receive Ethereum assets, and interact with other smart contracts, without revealing the origin address of the user. In the earlier versions we have found several major issues, but they were all fixed in the final version.

To the best of auditors knowledge:

- Railgun does not have any hidden ability to drain user funds.
- In the scenario where multiple users are using Railgun, privacy is preserved for users.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Fixed
CVF-2	Minor	Flaw	Fixed
CVF-3	Minor	Documentation	Fixed
CVF-4	Minor	Bad naming	Fixed
CVF-5	Minor	Bad datatype	Info
CVF-6	Minor	Suboptimal	Fixed
CVF-7	Minor	Bad datatype	Fixed
CVF-8	Minor	Suboptimal	Fixed
CVF-9	Major	Flaw	Fixed
CVF-10	Minor	Bad datatype	Info
CVF-11	Minor	Procedural	Fixed
CVF-12	Minor	Bad naming	Info
CVF-13	Minor	Flaw	Fixed
CVF-14	Minor	Readability	Info
CVF-15	Minor	Suboptimal	Info
CVF-16	Minor	Bad datatype	Info
CVF-17	Moderate	Suboptimal	Info
CVF-18	Minor	Flaw	Fixed
CVF-19	Minor	Suboptimal	Info
CVF-20	Critical	Flaw	Fixed
CVF-21	Major	Flaw	Fixed
CVF-22	Moderate	Flaw	Fixed
CVF-23	Minor	Flaw	Fixed
CVF-24	Minor	Suboptimal	Fixed
CVF-25	Minor	Readability	Fixed
CVF-26	Minor	Bad naming	Fixed
CVF-27	Minor	Readability	Fixed

ID	Severity	Category	Status
CVF-28	Minor	Procedural	Info
CVF-29	Minor	Bad naming	Info
CVF-30	Minor	Suboptimal	Info
CVF-31	Minor	Suboptimal	Info
CVF-32	Minor	Suboptimal	Fixed
CVF-33	Minor	Flaw	Fixed
CVF-34	Minor	Suboptimal	Fixed
CVF-35	Minor	Suboptimal	Fixed
CVF-36	Minor	Suboptimal	Fixed
CVF-37	Minor	Procedural	Info
CVF-38	Minor	Flaw	Info
CVF-39	Minor	Suboptimal	Info
CVF-40	Minor	Documentation	Fixed
CVF-41	Major	Flaw	Info
CVF-42	Minor	Suboptimal	Fixed
CVF-43	Minor	Unclear behavior	Fixed
CVF-44	Minor	Bad naming	Fixed
CVF-45	Minor	Bad datatype	Info
CVF-46	Minor	Bad datatype	Info
CVF-47	Minor	Suboptimal	Info
CVF-48	Minor	Flaw	Fixed
CVF-49	Minor	Suboptimal	Fixed
CVF-50	Moderate	Flaw	Info
CVF-51	Minor	Suboptimal	Fixed
CVF-52	Minor	Suboptimal	Fixed
CVF-53	Minor	Suboptimal	Fixed
CVF-54	Minor	Suboptimal	Fixed
CVF-55	Minor	Suboptimal	Fixed
CVF-56	Minor	Suboptimal	Fixed
CVF-57	Minor	Procedural	Fixed

ID	Severity	Category	Status
CVF-58	Critical	Flaw	Fixed
CVF-59	Minor	Suboptimal	Fixed
CVF-60	Major	Flaw	Fixed
CVF-61	Minor	Suboptimal	Info
CVF-62	Minor	Bad datatype	Fixed
CVF-63	Minor	Suboptimal	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	June 17, 2021	D. Khovratovich and M. Vladimirov	Initial Draft
0.2	June 17, 2021	D. Khovratovich	Minor revision
1.0	June 18, 2021	D. Khovratovich and M. Vladimirov	Release
1.1	July 1st, 2021	D. Khovratovich	Add client comment
2.0	July 2d, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

dmitry@abdkconsulting.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

We have audited two repositories: contract at commit 6281cb and files:

- Commitments.sol;
- RailgunLogic.sol;
- Snark.sol;
- TokenWhitelist.sol;
- Types.sol;
- Verifier.sol.

as well as circuits at commit 2c3c31 and files:

- base/HashInputs.circom;
- base/MerkleTree.circom;
- JoinSplit.circom;
- Large.circom;
- Small.circom.

The fixes were given in the release v0.0.1.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RailgunLogic.sol

Description This function always returns true.
Recommendation Consider removing the return value.
Client Comment Success returns where functions either succeed or revert have been removed.

Listing 1:

```
78  function changeTreasury(address payable _treasury) public
      ↪ onlyOwner returns (bool success) {

94  function changeFee(uint256 _fee) public onlyOwner returns (bool
      ↪ success) {

122 function transact(

227 function generateDeposit(
```

3.2 CVF-2

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** RailgunLogic.sol

Description This function emits an event even if the state has not changed.
Client Comment Wrapped in if statement to prevent emitting events when no state changes are made.

Listing 2:

```
78  function changeTreasury(address payable _treasury) public
      ↪ onlyOwner returns (bool success) {

94  function changeFee(uint256 _fee) public onlyOwner returns (bool
      ↪ success) {
```

3.3 CVF-3

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** RailgunLogic.sol

Description This argument goes first in the function signature, but not in the documentation comment.
Recommendation Consider describing arguments in the same order they are declared.
Client Comment Reordered parameters in documentation.

Listing 3:

```
111 * @param _proof - snark proof
```

3.4 CVF-4

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** RailgunLogic.sol

Recommendation The name is confusing. Should be ‘tokenAddress‘ or just ‘token‘
Client Comment Variable naming changed to ’_tokenField’ to maintain consistancy with naming in circom

Listing 4:

```
130 address _outputTokenField,
```

3.5 CVF-5

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** RailgunLogic.sol

Recommendation This argument should have type “IERC20”.
Client Comment Preferring the primitive value for interfaces.

Listing 5:

```
130 address _outputTokenField,
```

3.6 CVF-6

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RailgunLogic.sol

Recommendation This could be checked simply as: require (_adaptIDcontract == address (0) || _adaptIDcontract == msg.sender);
Client Comment Changed to || in require statement.

Listing 6:

```
146 // If _adaptIDcontract is not zero check that it matches the
    ↪ caller
if (_adaptIDcontract != address(0)) {
  require(_adaptIDcontract == msg.sender, "AdaptID doesn't match
    ↪  caller contract");
}
```

3.7 CVF-7

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RailgunLogic.sol

Recommendation The maximum deposit and withdraw amounts should be defined as named constants.
Client Comment Changed to named constant.

Listing 7:

```
155 require(_depositAmount < 2**120, "RailgunLogic: depositAmount
    ↪ too high");
require(_withdrawAmount < 2**120, "RailgunLogic: withdrawAmount
    ↪ too high");
```

3.8 CVF-8

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RailgunLogic.sol

Recommendation This check is cheap and can be made in the beginning of the function.
Client Comment Check moved to start of function.

Listing 8:

```
197 TokenWhitelist.tokenWhitelist[_outputTokenField],
```

3.9 CVF-9

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** RailgunLogic.sol

Recommendation There must be a 2^120 range check for the amount and field element checks for public keys and serials.
Client Comment Added require statements.

Listing 9:

```
228 uint256[2] calldata _pubkey,
    uint256 _serial,
230 uint256 _amount,
    address _token
```

3.10 CVF-10

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** RailgunLogic.sol

Recommendation This argument should have type "IERC20".
Client Comment Preferring the primitive value for interfaces.

Listing 10:

```
231 address _token
```

3.11 CVF-11

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Verifier.sol

Description This constant was already defined in to "Snark" library.
Recommendation Consider using is from there to avoid code duplication.
Client Comment Moved to types file and imported where needed.

Listing 11:

```
24 uint256 private constant SNARK_SCALAR_FIELD =
   ↪ 21888242871839275222246405745257275088548364400416034343698204186575
   ↪
```

3.12 CVF-12

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** Verifier.sol

Description The terms “small“ and “large” in names are too generic.
Recommendation Consider using more descriptive names, such as "2to3" and "10to3".
Client Comment "Small" and "large" naming has been used to keep consistancy with public messaging and documentation.

Listing 12:

```
27 VerifyingKey private vKeySmall;
   VerifyingKey private vKeyLarge;
```

3.13 CVF-13

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** Verifier.sol

Description There is no check for the lengths of these arrays, while it seems that the only valid length is 3.
Recommendation Consider adding explicit checks.
Client Comment Check was performed at top of verifyProof function so doesn't need to be re-preformed here. Check removed and array changed to fixed length for clarity.

Listing 13:

```
60  Commitment[] calldata _commitmentsOut

165 Commitment[] calldata _commitmentsOut
```

3.14 CVF-14

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** Verifier.sol

Recommendation Array literal would make the code more readable and less error prone.
Client Comment Array literal hasn't been used as it causes a stack overflow. Sub-optimal code patterns are used in Verifier.sol to work around stack size limitation.

Listing 14:

```
63  uint256[2] memory adaptIDhashPreimage;
    adaptIDhashPreimage[0] = uint256(uint160(_adaptIDcontract));
    adaptIDhashPreimage[1] = _adaptIDparameters;

70  uint256[24] memory cipherTextHashPreimage;
    // Commitment 0
    cipherTextHashPreimage[0] = _commitmentsOut[0].senderPubKey[0];
    cipherTextHashPreimage[1] = _commitmentsOut[0].senderPubKey[1];
    cipherTextHashPreimage[2] = _commitmentsOut[0].ciphertext[0];
    cipherTextHashPreimage[3] = _commitmentsOut[0].ciphertext[1];
    cipherTextHashPreimage[4] = _commitmentsOut[0].ciphertext[2];
    cipherTextHashPreimage[5] = _commitmentsOut[0].ciphertext[3];
    cipherTextHashPreimage[6] = _commitmentsOut[0].ciphertext[4];
    cipherTextHashPreimage[7] = _commitmentsOut[0].ciphertext[5];
80  // Commitment 1
    cipherTextHashPreimage[8] = _commitmentsOut[1].senderPubKey[0];
    cipherTextHashPreimage[9] = _commitmentsOut[1].senderPubKey[1];
    cipherTextHashPreimage[10] = _commitmentsOut[1].ciphertext[0];
    cipherTextHashPreimage[11] = _commitmentsOut[1].ciphertext[1];
    cipherTextHashPreimage[12] = _commitmentsOut[1].ciphertext[2];
    cipherTextHashPreimage[13] = _commitmentsOut[1].ciphertext[3];
    cipherTextHashPreimage[14] = _commitmentsOut[1].ciphertext[4];
    cipherTextHashPreimage[15] = _commitmentsOut[1].ciphertext[5];
    // Commitment 2
90  cipherTextHashPreimage[16] = _commitmentsOut[2].senderPubKey[0];
    cipherTextHashPreimage[17] = _commitmentsOut[2].senderPubKey[1];
    cipherTextHashPreimage[18] = _commitmentsOut[2].ciphertext[0];
    cipherTextHashPreimage[19] = _commitmentsOut[2].ciphertext[1];
    cipherTextHashPreimage[20] = _commitmentsOut[2].ciphertext[2];
    cipherTextHashPreimage[21] = _commitmentsOut[2].ciphertext[3];
    cipherTextHashPreimage[22] = _commitmentsOut[2].ciphertext[4];
    cipherTextHashPreimage[23] = _commitmentsOut[2].ciphertext[5];
    + 101, 168, 175, 207
```

3.15 CVF-15

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Verifier.sol

Description These arrays are redundant.
Recommendation Just pass the values to the "abi.encodePacked" function.
Client Comment Passing values directly to 'abi.encodePacked' causes stack overflow. Sub-optimal code patterns are used in Verifier.sol to work around stack size limitation.

Listing 15:

```
 63 uint256[2] memory adaptIDhashPreimage;
 70 uint256[24] memory cipherTextHashPreimage;
101 uint256[12] memory inputsHashPreimage;
168 uint256[2] memory adaptIDhashPreimage;
175 uint256[24] memory cipherTextHashPreimage;
207 uint256[18] memory inputsHashPreimage;
```

3.16 CVF-16

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Verifier.sol

Recommendation The array lengths should be named constants.
Client Comment Array length is specific to the array being defined so named constants have not been used.

Listing 16:

```
 70 uint256[24] memory cipherTextHashPreimage;
101 uint256[12] memory inputsHashPreimage;
175 uint256[24] memory cipherTextHashPreimage;
207 uint256[18] memory inputsHashPreimage;
```

3.17 CVF-17

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** Verifier.sol

Recommendation These values are not actually used in the circuit, so it will be more efficient to hash them into a single value first and pass the output to the circuit.
Client Comment Performance penalty on the circuit side has been chosen as an acceptable trade off for slightly lower gas use on-chain.

Listing 17:

```
102 inputsHashPreimage[0] = adaptIDhash % SNARK_SCALAR_FIELD;

106 inputsHashPreimage[4] = uint256(uint160(_outputEthAddress));

113 inputsHashPreimage[11] = cipherTextHash % SNARK_SCALAR_FIELD;
```

3.18 CVF-18

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** Verifier.sol

Recommendation This comment is misleading as the code does not check this condition.
Client Comment Comment adjusted to clarify where verification is performed.

Listing 18:

```
141 * @param _adaptIDcontract - contract address to this proof to (
    ↪ ignored if set to 0)
```

3.19 CVF-19

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Verifier.sol

Description The code of this function largely duplicates that of 'hashSmallInputs'.
Recommendation Consider using a single function where the input length is just a parameter.
Client Comment Code is duplicated as the added switching statements cause stack overflow. Sub-optimal code patterns are used in Verifier.sol to work around stack size limitation.

Listing 19:

```
153 function hashLargeInputs(
```

3.20 CVF-20

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** Verifier.sol

Description Hashes of second and third commitments are missing.
Client Comment Commitment hashes added.

Listing 20:

```
224 inputsHashPreimage[16] = _commitmentsOut[0].hash;
    inputsHashPreimage[17] = cipherTextHash % SNARK_SCALAR_FIELD;
```

3.21 CVF-21

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** Verifier.sol

Recommendation It should be checked that these parameters are valid field elements.
Client Comment The 'depositAmount' and the 'withdrawAmount' are checked to be $< 2^{120}$ in earlier functions (transact and generateDeposit in RailgunLogic.sol). Checks for nullifiers and commitment hashes added.

Listing 21:

```
277 uint256 _depositAmount,
    uint256 _withdrawAmount,

282 uint256[] calldata _nullifiers,
    uint256 _merkleRoot,

285 Commitment[] calldata _commitmentsOut
```

3.22 CVF-22

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Verifier.sol

Recommendation This function should check that the verification key is a set of valid curve points.
Client Comment VKey points are directly passed to EIP-196 and EIP-197 which have an internal check against invalid points. Also, we added check in the negate function.

Listing 22:

```
343 function setVKeySmall(VerifyingKey calldata _vKey) public
    ↪ onlyOwner returns (bool success) {

381 function setVKeyLarge(VerifyingKey calldata _vKey) public
    ↪ onlyOwner returns (bool success) {
```

3.23 CVF-23

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** Verifier.sol

Description These functions always returns true.
Recommendation Consider removing the returned values.
Client Comment Return values removed.

Listing 23:

```
372 return true;

410 return true;
```

3.24 CVF-24

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** JoinSplit.circom

Recommendation Using ‘pathIndices‘ instead of serial numbers would simplify requirements as the former are unique by definition.
Client Comment Replaced by 'treeNumber' and 'pathIndices' since there will be multiple trees.

Listing 24:

```
75 hasherNullifier[i].inputs[1] <== serialsIn[i];
```

3.25 CVF-25

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** JoinSplit.circom

Recommendation There are logical gates in circomlib that could make boolean calculations more readable and less error-prone.
Client Comment Fixed by using 'ForceEqualIfEnabled' gate.

Listing 25:

```
99      (merkle[i].root - merkleRoot)*(1-isDummyInput[i].out) === 0;

137 outputTokenField === tokenField * (1-isShieldedTransaction.out);
```

3.26 CVF-26

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** MerkleTree.circom

Recommendation 'MerklePath' or 'MerkleProof' would be a better name
Client Comment Changed.

Listing 26:

```
5 MerkleTree(n_levels) {
```

3.27 CVF-27

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** MerkleTree.circom

Description There are multiplexor templates in circomlib.
Recommendation Consider using them to make the code more readable.
Client Comment Used Switcher.

Listing 27:

```
20 hashers[i].inputs[0] <== index.out[i]*(pathElements[i] -
     ↪ levelHash) + levelHash;
hashers[i].inputs[1] <== index.out[i]*(levelHash - pathElements[
     ↪ i]) + pathElements[i];
```

3.28 CVF-28

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Commitments.sol

Description We didn't review this file.
Client Comment File is interface definition and contains no deployed code therefore was not included in Audit scope.

Listing 28:

```
11 { PoseidonT3, PoseidonT6 } from "./Poseidon.sol";
```

3.29 CVF-29

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** Commitments.sol

Description The names are similar and it is hard to get what is the difference between these two events.
Recommendation Consider using more specific names.
Client Comment Events refer to the same conceptual event reached through different code paths/some different fields. Names left as is.

Listing 29:

```
28 event NewCommitment(

37 event NewGeneratedCommitment(
```

3.30 CVF-30

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Commitments.sol

Recommendation A public key can be compressed to a single 256-bit value.
Client Comment X,Y representation is used to avoid needing to decompress values to build hash preimages.

Listing 30:

```
33 uint256[2] senderPubKey

41 uint256[2] pubkey,
```

3.31 CVF-31

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Commitments.sol

Recommendation Why this value is so little? Bigger batches should save more gas.
Client Comment The 'MAX_BATCH_SIZE' equal to the largest number of outputsthat a single transaction is capable of inserting. Named constant is provided in anticipation of a future upgrade containing a batch transaction method.

Listing 31:

```
57 uint256 internal constant MAX_BATCH_SIZE = 3;
```

3.32 CVF-32

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Commitments.sol

Recommendation The "abi.encodePacked" invocation is redundant here. Just use keccak256 ("Railgun").
Client Comment The 'encodePacked' call removed.

Listing 32:

```
60 uint256 private constant ZERO_VALUE = uint256(keccak256(abi.
   ↪ encodePacked("Railgun"))) % SNARK_SCALAR_FIELD;
```

3.33 CVF-33

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** Commitments.sol

Description The variable name and the comment are different.
Recommendation Consider changing the comment to something like this: // The next leaf index, which is the same as the number of inserted leaves
Client Comment Comment updated for clarity.

Listing 33:

```
62 // The number of inserted leaves
   uint256 private nextLeafIndex = 0;
```

3.34 CVF-34

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Commitments.sol

Recommendation This code could be simplified to: currentZero = ZERO_VALUE; for (i = 0; i < TREE_DEPTH; i++) { zeroValues [i] = currentZero; currentZero = hash (currentZero, currentZero); } merkleRoot = currentZero;
Client Comment Loop re-ordered as recommended.

Listing 34:

```
107 // Calculate zero values
    zeros[0] = ZERO_VALUE;

110 // Store the current zero value for the level we just calculated
    ↪  it for
    uint256 currentZero = ZERO_VALUE;

    // Loop through each level
    for (uint256 i = 1; i < TREE_DEPTH; i++) {
      // Calculate the zero value for this level
      currentZero = hashLeftRight(currentZero, currentZero);

      // Push it to zeros array
      zeros[i] = currentZero;
120 }

    // Calculate merkle root
    merkleRoot = hashLeftRight(currentZero, currentZero);
```

3.35 CVF-35

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Commitments.sol

Recommendation Here just stored variable "merkleRoot" is read from the storage twice. While Solidity compiler could be smart enough to optimize this, it would be better to cache the value in a local variable and reuse.
Client Comment Used chained assignment/reused 'currentZero' to assing in single action without reading.

Listing 35:

```
124 rootHistory[merkleRoot] = true;

127 newTreeRoot = merkleRoot;
```

3.36 CVF-36

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Commitments.sol

Recommendation This could be written as: return PoseidonT3.poseidon([_left, _right]);
Client Comment Changed to passing array literal to function.

Listing 36:

```
137 uint256[2] memory input = [
      _left,
      _right
140 ];
    return PoseidonT3.poseidon(input);
```

3.37 CVF-37

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Commitments.sol

Description There is no range check for the '_count'.
Recommendation Consider adding an explicit check. Also. for '_count == 0' the function could return earlier.
Client Comment Count is not a publically set variable, range check not needed.

Listing 37:

```
153 function insertLeaves(uint256[MAX_BATCH_SIZE] memory _leafHashes
      ↪ , uint256 _count) private {
```

3.38 CVF-38

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** Commitments.sol

Recommendation This loop should iterate on the next level cells and process a pair of current level elements on each iteration (the very first and the very last iteration could process one current level element in case of odd indexes).
Client Comment This results in at most 2 extra hashes being performed with current max batch size. Leaving code as-is, will revisit for optimization if a deployment with a larger batch size is needed.

Listing 38:

```
189 for (uint256 insertionElement = 0; insertionElement < _count;
      ↪ insertionElement++) {
```

3.39 CVF-39

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Commitments.sol

Description Here the storage is used to pass values between loop iterations, which is suboptimal.
Recommendation Consider using a local variable instead.
Client Comment SLOAD/SSTORE gas cost reduces to 100 after first access to the storage location in a transaction. Copying array to memory and then copying resulting array to storage results in larger gas cost. (See: EIP 2929)

Listing 39:

```
200 filledSubTrees[level] = _leafHashes[insertionElement];

203 left = filledSubTrees[level];
```

3.40 CVF-40

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Commitments.sol

Recommendation There shoulb be"hash", instead of "has".
Client Comment Has changed to hash.

Listing 40:

```
211 // Calculate the has for the next level
```

3.41 CVF-41

- **Severity** Major
- **Status** Info
- **Category** Flaw
- **Source** Commitments.sol

Recommendation In case the current level insertion index is even and the element is no the last one, this hash will be overwritten at the next loop iteration and thus will never be used.
Client Comment This results in at most 2 extra hashes being performed with current max batch size. Leaving code as-is, will revisit for optimization if a deployment with a larger batch size is needed.

Listing 41:

```
212 _leafHashes[nextLevelHashIndex] = hashLeftRight(left, right);
```

3.42 CVF-42

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Commitments.sol

Recommendation This duplicated code should be extracted to a utility function.
Client Comment Extracted to 'newTree()' function.

Listing 42:

```
243 // Restore merkleRoot to newTreeRoot
    merkleRoot = newTreeRoot;

    // Existing values in filledSubtrees will never be used so
       ↪ overwriting them is unnecessary

    // Reset next leaf index to 0
    nextLeafIndex = 0;
250
    // Increment tree number
    treeNumber++;

295 // Restore merkleRoot to newTreeRoot
    merkleRoot = newTreeRoot;

    // Existing values in filledSubtrees will never be used so
       ↪ overwriting them is unnecessary

300 // Reset next leaf index to 0
    nextLeafIndex = 0;

    // Increment tree number
    treeNumber++;
```

3.43 CVF-43

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Commitments.sol

Description It is not checked that the values are in field. Probably not an issue.
Client Comment These are now checked at 'generateDeposit()' in RailgunLogic.sol.

Listing 43:

```
309 _pubkey[0],
310 _pubkey[1],
    _serial,
    _amount,
```

3.44 CVF-44

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** TokenWhitelist.sol

Description Events are usually named via nouns. The names are too complicated. The name "RemoveFromTokenUnwhitelist" is grammatically incorrect.
Recommendation Consider renaming to just "Addition" and "Removal" or "Listing" and "Delisting".
Client Comment Changed to "TokenListing" and "TokenDelisting".

Listing 44:

```
21 event AddToTokenWhitelist(address indexed token);
   event RemoveFromTokenUnwhitelist(address indexed token);
```

3.45 CVF-45

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** TokenWhitelist.sol

Recommendation The key type should be "IERC20".
Client Comment Preferring the primitive value.

Listing 45:

```
28 mapping(address => bool) public tokenWhitelist;
```

3.46 CVF-46

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** TokenWhitelist.sol

Recommendation The type of the "_tokens" argument should be "IERC20 [] calldata".
Client Comment Preferring the primitive value for interfaces.

Listing 46:

```
36 function initializeTokenWhitelist(address[] calldata _tokens)
     ↪ internal initializer {

49 function addToWhitelist(address[] calldata _tokens) public
     ↪ onlyOwner returns (bool success) {

76 function removeFromWhitelist(address[] calldata _tokens)
     ↪ external onlyOwner returns (bool success) {
```

3.47 CVF-47

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** TokenWhitelist.sol

Description This function is guarded by the "onlyOwner" modifier which effectively means that the "initializeTokenWhitelist" function is also callable only by the owner of the smart contract. This could cause problems in some scenarios.
Recommendation Consider moving the logic of the "addToWhitelist" into an unprotected internal function, and calling this function from both, "initializeTokenWhitelist" and "addToWhitelist" functions.
Client Comment The order of initialization allows the initializer to act as owner on initialization so this pattern isn't an issue for us.

Listing 47:

```
38 addToWhitelist(_tokens);
```

3.48 CVF-48

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** TokenWhitelist.sol

Description This function always returns true.
Recommendation Consider removing the return values.
Client Comment Return values removed.

Listing 48:

```
49 function addToWhitelist(address[] calldata _tokens) public
     ↪ onlyOwner returns (bool success) {

76 function removeFromWhitelist(address[] calldata _tokens)
     ↪ external onlyOwner returns (bool success) {
```

3.49 CVF-49

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Snark.sol

Recommendation This library should be moved to a separate file named "Pairing.sol" or should be merged with the "Snark" library.
Client Comment Pairing library merged with snark library.

Listing 49:

```
7 Pairing {
```

3.50 CVF-50

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** Snark.sol

Description This produces an invalid point in case p.y % PRIME_Q is zero.
Client Comment if the result of negate is invalid point then, pairing function will revert.

Listing 50:

```
21 return G1Point(p.x, PRIME_Q - (p.y % PRIME_Q));
```

3.51 CVF-51

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Snark.sol

Description The input size (0xc0) and the output size (0x60) are too big.
Recommendation The 0x80 and 0x40 respectively would be enough.
Client Comment Allocation sizes changed.

Listing 51:

```
44 success := staticcall(sub(gas(), 2000), 6, input, 0xc0, result,
   ↪ 0x60)
```

3.52 CVF-52

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Snark.sol

Recommendation The "success" flag could be checked outside the assembly block.
Client Comment Removed in-assembly checks.

Listing 52:

```
46 switch success case 0 { invalid() }

73 switch success case 0 { invalid() }
```

3.53 CVF-53

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Snark.sol

Recommendation This was already checked inside the assembly block, so this check could never fail.
Client Comment Removed in-assembly checks.

Listing 53:

```
51 require(success, "Pairing: Add Failed");

78 require(success, "Pairing: Scalar Multiplication Failed");
```

3.54 CVF-54

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Snark.sol

Description The input size (0x80) and the output size (0x60) are too big.
Recommendation The 0x60 and 0x40 respectively would be enough.
Client Comment Allocation sizes changed.

Listing 54:

```
71 success := staticcall(sub(gas(), 2000), 7, input, 0x80, r, 0x60)
```

3.55 CVF-55

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Snark.sol

Recommendation Here the input values are copied twice, which is suboptimal. Just create a static array of 24 element and use 24 plain assignments without any loops.
Client Comment Changed to static array.

Listing 55:

```
 99 G1Point[4] memory p1 = [_a1, _b1, _c1, _d1];
100 G2Point[4] memory p2 = [_a2, _b2, _c2, _d2];

104 for (uint256 i = 0; i < 4; i++) {
      uint256 j = i * 6;
      input[j + 0] = p1[i].x;
      input[j + 1] = p1[i].y;
      input[j + 2] = p2[i].x[0];
      input[j + 3] = p2[i].x[1];
110   input[j + 4] = p2[i].y[0];
      input[j + 5] = p2[i].y[1];
    }
```

3.56 CVF-56

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Snark.sol

Recommendation Statically sized array would be more efficient here.
Client Comment Changed to static array.

Listing 56:

```
102 uint256[] memory input = new uint256[](PAIRING_INPUT_SIZE);
```

3.57 CVF-57

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Snark.sol

Recommendation The value calculated here is actually a constant and should be precomputed.
Client Comment Changed to precomputed constant.

Listing 57:

```
123 mul(PAIRING_INPUT_SIZE, 0x20),
```

3.58 CVF-58

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** Snark.sol

Recommendation The function always returns true. It should return (out[0] != 0).
Client Comment Changed to 'return out[0] != 0'.

Listing 58:

```
135 return true;
```

3.59 CVF-59

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Snark.sol

Description This constant was already defined in the "Pairing" library.
Recommendation Consider defining it only once and reusing. Libraries are able to use constants defined in other libraries.
Client Comment Merged libraries so this is only declared once now.

Listing 59:

```
141 uint256 private constant PRIME_Q =
    ↪ 21888242871839275222246405745257275088696311157297823662689037894645226208583
    ↪
```

3.60 CVF-60

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** Snark.sol

Recommendation The function does not verify that the proof elements are actually points on a curve. This may lead to false positives on invalid inputs.
Client Comment proof elements are passed to EIP-196 and EIP-197 functions which revert in case of invalid points. Checks have been added to negate function.

Listing 60:

```
150 SnarkProof memory _proof,
```

3.61 CVF-61

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Snark.sol

Description These long string literals increase the byte code size. The only important parts in them are the variable reverences.
Recommendation Consider removing unimportant parts.
Client Comment String literals made slightly shorter by using shorthand (gte) however on the whole slightly larger bytecode deemed acceptable tradeoff.

Listing 61:

```
157 require(_proof.a.x < PRIME_Q, "Snark: Point a.x is greater than
    ↪ PRIME_Q");
    require(_proof.a.y < PRIME_Q, "Snark: Point a.y is greater than
    ↪ PRIME_Q");

160 require(_proof.b.x[0] < PRIME_Q, "Snark: Point b[0].x is greater
    ↪  than PRIME_Q");
    require(_proof.b.y[0] < PRIME_Q, "Snark: Point b[0].y is greater
    ↪  than PRIME_Q");

163 require(_proof.b.x[1] < PRIME_Q, "Snark: Point b[1].x is greater
    ↪  than PRIME_Q");
    require(_proof.b.y[1] < PRIME_Q, "Snark: Point b[1].y is greater
    ↪  than PRIME_Q");

166 require(_proof.c.x < PRIME_Q, "Snark: Point c.x is greater than
    ↪ PRIME_Q");
    require(_proof.c.y < PRIME_Q, "Snark: Point c.y is greater than
    ↪ PRIME_Q");
```

3.62 CVF-62

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Types.sol

Recommendation There should be named constants for the number of ciphertext words, as well for the indexes of particular fields in the ciphertext.
Client Comment The 'CIPHERTEXT_WORDS' constants added.

Listing 62:

```
8 uint256[6] ciphertext; // Ciphertext order: iv, recipient pubkey
  ↪  (2 x uint256), serial, amount, token
```

3.63 CVF-63

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** HashInputs.circom

Recommendation The current API makes the caller know the exact formula for SIZE. Consider just passing SIZE as a single parameter.
Client Comment Changed to one parameter size as recommended.

Listing 63:

```
4 HashInputs(nInputs, mOutputs){

6     var SIZE = 7 +nInputs + mOutputs
      signal input in[SIZE];
```