

A background network diagram consisting of numerous white nodes connected by thin white lines, set against a light blue gradient. The nodes are scattered across the upper and middle portions of the page, creating a sense of connectivity and data flow.

ABDK CONSULTING

SMART CONTRACT
AUDIT

Railgun

Circum and Solidity

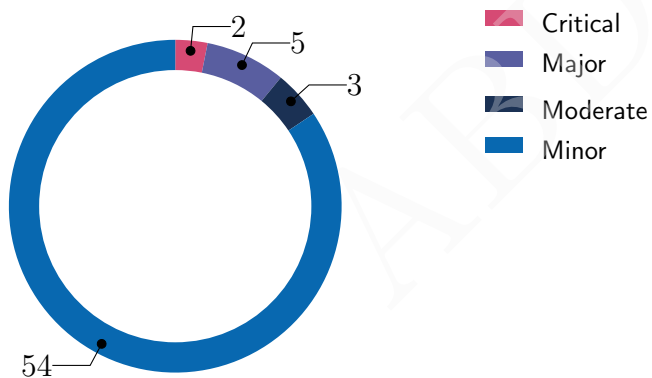


abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
18th June 2021

We've been asked to review Railgun smart contracts and circuits given in separate files. We have found 2 critical and 5 major issues, as well as many other issues with lower severity.



Findings

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

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

ID	Severity	Category	Status
CVF-58	Minor	Procedural	Opened
CVF-59	Critical	Flaw	Opened
CVF-60	Minor	Suboptimal	Opened
CVF-61	Major	Flaw	Opened
CVF-62	Minor	Suboptimal	Opened
CVF-63	Minor	Bad datatype	Opened
CVF-64	Minor	Suboptimal	Opened

ABDK

Contents

1	Document properties	8
2	Introduction	9
2.1	About ABDK	9
2.2	Disclaimer	9
2.3	Methodology	10
3	Detailed Results	11
3.1	CVF-1	11
3.2	CVF-2	11
3.3	CVF-3	12
3.4	CVF-4	12
3.5	CVF-5	12
3.6	CVF-6	13
3.7	CVF-7	13
3.8	CVF-8	13
3.9	CVF-9	14
3.10	CVF-10	14
3.11	CVF-11	14
3.12	CVF-12	15
3.13	CVF-13	15
3.14	CVF-14	16
3.15	CVF-15	17
3.16	CVF-16	17
3.17	CVF-17	18
3.18	CVF-18	18
3.19	CVF-19	18
3.20	CVF-20	19
3.21	CVF-21	19
3.22	CVF-22	19
3.23	CVF-23	20
3.24	CVF-24	20
3.25	CVF-25	20
3.26	CVF-26	21
3.27	CVF-27	21
3.28	CVF-28	21
3.29	CVF-29	22
3.30	CVF-30	22
3.31	CVF-31	22
3.32	CVF-32	23
3.33	CVF-33	23
3.34	CVF-34	23
3.35	CVF-35	24
3.36	CVF-36	24
3.37	CVF-37	25

3.38 CVF-38	25
3.39 CVF-39	25
3.40 CVF-40	26
3.41 CVF-41	26
3.42 CVF-42	26
3.43 CVF-43	27
3.44 CVF-44	27
3.45 CVF-45	28
3.46 CVF-46	28
3.47 CVF-47	28
3.48 CVF-48	29
3.49 CVF-49	29
3.50 CVF-50	29
3.51 CVF-51	30
3.52 CVF-52	30
3.53 CVF-53	30
3.54 CVF-54	31
3.55 CVF-55	31
3.56 CVF-56	32
3.57 CVF-57	32
3.58 CVF-58	32
3.59 CVF-59	33
3.60 CVF-60	33
3.61 CVF-61	33
3.62 CVF-62	34
3.63 CVF-63	34
3.64 CVF-64	35

1 Document properties

Version

Version	Date	Author	Description
0.1	June 17, 2021	D. Khovratovich	Initial Draft
0.2	June 17, 2021	D. Khovratovich	Minor revision
1.0	June 18, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

ABDK

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.

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 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
- **Category** Procedural
- **Status** Opened
- **Source** RailgunLogic.sol

Description This function always returns true.

Recommendation Consider removing the return value.

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
- **Category** Flaw
- **Status** Opened
- **Source** RailgunLogic.sol

Description This function emits an event even if the state has not changed.

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
- **Category** Documentation
- **Status** Opened
- **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.

Listing 3:

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

3.4 CVF-4

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

Recommendation The name is confusing. Should be 'tokenAddress' or just 'token'

Listing 4:

```
130 address _outputTokenField ,
```

3.5 CVF-5

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

Recommendation This argument should have type "IERC20".

Listing 5:

```
130 address _outputTokenField ,
```

3.6 CVF-6

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

Recommendation This could be checked simply as: `require (_adaptIDcontract == address(0) || _adaptIDcontract == msg.sender);`

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
- **Category** Bad datatype
- **Status** Opened
- **Source** RailgunLogic.sol

Recommendation The maximum deposit and withdraw amounts should be defined as named constants.

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
- **Category** Suboptimal
- **Status** Opened
- **Source** RailgunLogic.sol

Recommendation This check is cheap and can be made in the beginning of the function.

Listing 8:

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

3.9 CVF-9

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

Recommendation There must be a 2¹²⁰ range check for the amount and field element checks for public keys and serials.

Listing 9:

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

3.10 CVF-10

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

Recommendation This argument should have type "IERC20".

Listing 10:

```
231 address _token
```

3.11 CVF-11

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

Description This constant was already defined in to "Snark" library.

Recommendation Consider using is from there to avoid code duplication.

Listing 11:

```
24 uint256 private constant SNARK_SCALAR_FIELD =
    ↪ 21888242871839275222246405745257275088548364400416034343698
    ↪
    204186575808495617;
```

3.12 CVF-12

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **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".

Listing 12:

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

3.13 CVF-13

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **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.

Listing 13:

```
60 Commitment [] calldata _commitmentsOut  
165 Commitment [] calldata _commitmentsOut
```

3.14 CVF-14

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

Recommendation Array literal would make the code more readable and less error prone.

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
- **Category** Suboptimal
- **Status** Opened
- **Source** Verifier.sol

Description These arrays are redundant.

Recommendation Just pass the values to the "abi.encodePacked" function.

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
- **Category** Bad datatype
- **Status** Opened
- **Source** Verifier.sol

Recommendation The array lengths should be named constants.

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
- **Category** Suboptimal
- **Status** Opened
- **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.

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
- **Category** Flaw
- **Status** Opened
- **Source** Verifier.sol

Recommendation This comment is misleading as the code does not check this condition.

Listing 18:

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

3.19 CVF-19

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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.

Listing 19:

```
153 function hashLargeInputs(
```

3.20 CVF-20

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

Description Hashes of second and third commitments are missing.

Listing 20:

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

3.21 CVF-21

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

Recommendation It should be checked that these parameters are valid field elements.

Listing 21:

```
277 uint256 _depositAmount,  
uint256 _withdrawAmount,  
  
282 uint256[] calldata _nullifiers,  
uint256 _merkleRoot,  
  
285 Commitment[] calldata _commitmentsOut
```

3.22 CVF-22

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

Recommendation It should be checked that the length of this array equals the number of nullifiers

Listing 22:

```
285 Commitment[] calldata _commitmentsOut
```

3.23 CVF-23

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

Recommendation This function should check that the verification key is a set of valid curve points.

Listing 23:

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

3.24 CVF-24

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

Description These functions always returns true.

Recommendation Consider removing the returned values.

Listing 24:

```
372 return true;  
  
410 return true;
```

3.25 CVF-25

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

Recommendation Using 'pathIndices' instead of serial numbers would simplify requirements as the former are unique by definition.

Listing 25:

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

3.26 CVF-26

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

Recommendation There are logical gates in circomlib that could make boolean calculations more readable and less error-prone.

Listing 26:

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

3.27 CVF-27

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

Recommendation 'MerklePath' or 'MerkleProof' would be a better name

Listing 27:

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

3.28 CVF-28

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

Description There are multiplexor templates in circomlib.

Recommendation Consider using them to make the code more readable.

Listing 28:

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

3.29 CVF-29

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

Description We didn't review this file.

Listing 29:

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

3.30 CVF-30

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **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.

Listing 30:

```
28 event NewCommitment(  
37 event NewGeneratedCommitment(  
38     uint256[2] senderPubKey,  
39     uint256[2] pubkey,
```

3.31 CVF-31

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

Recommendation A public key can be compressed to a single 256-bit value.

Listing 31:

```
33 uint256 [2] senderPubKey  
41 uint256 [2] pubkey ,
```

3.32 CVF-32

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

Recommendation Why this value is so little? Bigger batches should save more gas.

Listing 32:

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

3.33 CVF-33

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

Recommendation The "abi.encodePacked" invocation is redundant here. Just use keccak256 ("Railgun").

Listing 33:

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

3.34 CVF-34

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **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

Listing 34:

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

3.35 CVF-35

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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;`

Listing 35:

```
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);
```

3.36 CVF-36

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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.

Listing 36:

```
124 rootHistory [merkleRoot] = true;

127 newTreeRoot = merkleRoot;
```


3.37 CVF-37

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

Recommendation This could be written as: `return PoseidonT3.poseidon([_left, _right]);`

Listing 37:

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

3.38 CVF-38

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **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.

Listing 38:

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

3.39 CVF-39

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **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).

Listing 39:

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

3.40 CVF-40

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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.

Listing 40:

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

3.41 CVF-41

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

Recommendation There should be "hash", instead of "has".

Listing 41:

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

3.42 CVF-42

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

Recommendation In case the current level insertion index is even and the element is not the last one, this hash will be overwritten at the next loop iteration and thus will never be used.

Listing 42:

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

3.43 CVF-43

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

Recommendation This duplicated code should be extracted to a utility function.

Listing 43:

```
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++;
```

3.44 CVF-44

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

Description It is not checked that the values are in field. Probably not an issue.

Listing 44:

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

3.45 CVF-45

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **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”.

Listing 45:

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

3.46 CVF-46

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

Recommendation The key type should be “IERC20”.

Listing 46:

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

3.47 CVF-47

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

Recommendation The type of the “_tokens” argument should be “IERC20 [] calldata”.

Listing 47:

```
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) {
```

3.48 CVF-48

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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.

Listing 48:

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

3.49 CVF-49

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

Description This function always returns true.

Recommendation Consider removing the return values.

Listing 49:

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

3.50 CVF-50

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

Recommendation This library should be moved to a separate file named “Pairing.sol” or should be merged with the “Snark” library.

Listing 50:

```
7 Pairing {
```

3.51 CVF-51

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

Description This produces an invalid point in case $p.y \% \text{PRIME_Q}$ is zero.

Listing 51:

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

3.52 CVF-52

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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.

Listing 52:

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

3.53 CVF-53

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

Recommendation The “success” flag could be checked outside the assembly block.

Listing 53:

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

3.54 CVF-54

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

Recommendation This was already checked inside the assembly block, so this check could never fail.

Listing 54:

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

3.55 CVF-55

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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.

Listing 55:

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

3.56 CVF-56

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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.

Listing 56:

```
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];
}
```

3.57 CVF-57

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

Recommendation Statically sized array would be more efficient here.

Listing 57:

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

3.58 CVF-58

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

Recommendation The value calculated here is actually a constant and should be precomputed.

Listing 58:

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


3.59 CVF-59

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

Recommendation The function always returns true. It should return `(out[0] != 0)`.

Listing 59:

```
135 return true;
```

3.60 CVF-60

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **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.

Listing 60:

```
141 uint256 private constant PRIME_Q =  
    ↪ 218882428718392752222464057452572750886963111572978236626890378946452  
    ↪
```

3.61 CVF-61

- **Severity** Major
- **Category** Flaw
- **Status** Opened
- **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.

Listing 61:

```
150 SnarkProof memory _proof,
```

3.62 CVF-62

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

Description These long string literals increase the byte code size. The only important parts in them are the variable references.

Recommendation Consider removing unimportant parts.

Listing 62:

```
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");
```

3.63 CVF-63

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **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.

Listing 63:

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

3.64 CVF-64

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

Recommendation The current API makes the caller know the exact formula for SIZE. Consider just passing SIZE as a single parameter.

Listing 64:

```
4 HashInputs(nInputs , mOutputs){
6     var SIZE = 7 +nInputs + mOutputs
    signal input in[SIZE];
```