ABDK CONSULTING

SMART CONTRACT AND CIRCUIT AUDIT

Mystiko

mystikonetwork

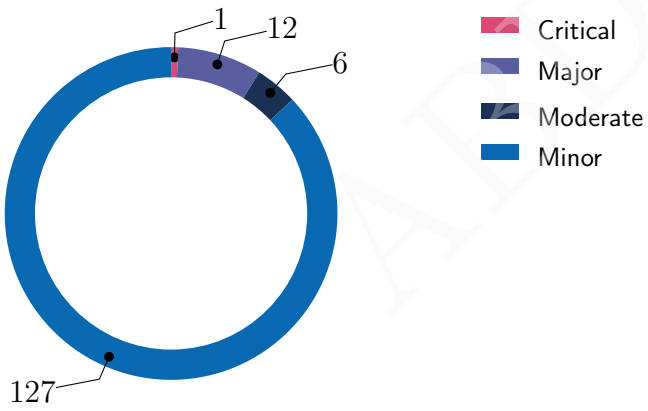
Solidity and Zokrates

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
1st August 2022

We've been asked to review 39 files in a Github repository. We found 1 critical, 12 major, and a few less important issues. All identified critical and major issues have been fixed.



Findings

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

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

ID	Severity	Category	Status
CVF-58	Moderate	Suboptimal	Info
CVF-59	Minor	Suboptimal	Fixed
CVF-60	Minor	Suboptimal	Fixed
CVF-61	Minor	Suboptimal	Fixed
CVF-62	Minor	Suboptimal	Info
CVF-63	Minor	Readability	Fixed
CVF-64	Minor	Suboptimal	Fixed
CVF-65	Minor	Suboptimal	Fixed
CVF-66	Major	Suboptimal	Fixed
CVF-67	Moderate	Suboptimal	Fixed
CVF-68	Minor	Suboptimal	Fixed
CVF-69	Moderate	Suboptimal	Fixed
CVF-70	Critical	Overflow/Underflow	Fixed
CVF-71	Minor	Suboptimal	Info
CVF-72	Minor	Suboptimal	Info
CVF-73	Minor	Suboptimal	Info
CVF-74	Minor	Documentation	Fixed
CVF-75	Minor	Documentation	Fixed
CVF-76	Minor	Procedural	Fixed
CVF-77	Minor	Procedural	Fixed
CVF-78	Minor	Procedural	Fixed
CVF-79	Minor	Bad datatype	Fixed
CVF-80	Major	Overflow/Underflow	Fixed
CVF-81	Minor	Suboptimal	Info
CVF-82	Minor	Suboptimal	Fixed
CVF-83	Minor	Suboptimal	Info
CVF-84	Minor	Procedural	Fixed
CVF-85	Minor	Procedural	Fixed
CVF-86	Minor	Bad datatype	Fixed
CVF-87	Minor	Bad datatype	Info

ID	Severity	Category	Status
CVF-88	Minor	Readability	Fixed
CVF-89	Major	Flaw	Fixed
CVF-90	Minor	Suboptimal	Fixed
CVF-91	Minor	Procedural	Fixed
CVF-92	Minor	Suboptimal	Fixed
CVF-93	Minor	Bad datatype	Info
CVF-94	Minor	Bad datatype	Fixed
CVF-95	Minor	Bad datatype	Info
CVF-96	Minor	Suboptimal	Fixed
CVF-97	Minor	Flaw	Fixed
CVF-98	Minor	Suboptimal	Fixed
CVF-99	Minor	Documentation	Fixed
CVF-100	Minor	Suboptimal	Info
CVF-101	Moderate	Procedural	Fixed
CVF-102	Minor	Bad naming	Fixed
CVF-103	Minor	Suboptimal	Fixed
CVF-104	Minor	Bad datatype	Fixed
CVF-105	Minor	Procedural	Fixed
CVF-106	Minor	Bad datatype	Fixed
CVF-107	Minor	Procedural	Fixed
CVF-108	Minor	Procedural	Fixed
CVF-109	Minor	Bad datatype	Fixed
CVF-110	Minor	Procedural	Fixed
CVF-111	Minor	Procedural	Fixed
CVF-112	Minor	Bad naming	Fixed
CVF-113	Minor	Bad naming	Info
CVF-114	Minor	Suboptimal	Info
CVF-115	Minor	Suboptimal	Fixed
CVF-116	Minor	Bad datatype	Fixed
CVF-117	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-118	Minor	Bad datatype	Fixed
CVF-119	Minor	Suboptimal	Fixed
CVF-120	Minor	Suboptimal	Fixed
CVF-121	Minor	Suboptimal	Fixed
CVF-122	Minor	Procedural	Fixed
CVF-123	Minor	Procedural	Fixed
CVF-124	Minor	Bad datatype	Fixed
CVF-125	Minor	Suboptimal	Fixed
CVF-126	Minor	Bad datatype	Fixed
CVF-127	Minor	Suboptimal	Info
CVF-128	Minor	Bad datatype	Fixed
CVF-129	Minor	Suboptimal	Info
CVF-130	Moderate	Procedural	Fixed
CVF-131	Major	Documentation	Fixed
CVF-132	Minor	Bad datatype	Fixed
CVF-133	Minor	Bad datatype	Fixed
CVF-134	Minor	Bad datatype	Fixed
CVF-135	Minor	Bad datatype	Fixed
CVF-136	Major	Suboptimal	Fixed
CVF-137	Minor	Bad datatype	Fixed
CVF-138	Minor	Bad datatype	Fixed
CVF-139	Minor	Bad datatype	Fixed
CVF-140	Minor	Bad datatype	Fixed
CVF-141	Minor	Bad datatype	Fixed
CVF-142	Minor	Bad datatype	Fixed
CVF-143	Minor	Procedural	Info
CVF-144	Minor	Bad naming	Info
CVF-145	Minor	Bad naming	Info
CVF-146	Minor	Bad naming	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	August 1, 2022	D. Khovratovich	Initial Draft
0.2	August 1, 2022	D. Khovratovich	Minor revision
1.0	August 1, 2022	D. Khovratovich	Release
1.1	August 1, 2022	D. Khovratovich	CVF-131 was removed
2.0	August 1, 2022	D. Khovratovich	Release
2.1	August 1, 2022	D. Khovratovich	Mystico > Mystiko
3.0	August 1, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.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 and circuit structure, critical/major bugs detection and issuing the general recommendations.
We have reviewed contracts and circuits at repository.

Solidity smart contracts:

- core/commitment/CommitmentPool.sol
- core/commitment/CommitmentPoolERC20.sol
- core/commitment/CommitmentPoolMain.sol
- core/deposit/base/MystikoV2Loop.sol
- core/deposit/loop/MystikoV2LoopERC20.sol
- core/deposit/loop/MystikoV2LoopMain.sol
- core/rule/Sanctions.sol
- interface/ICommitmentPool.sol
- interface/IHasher3.sol
- interface/IMystikoLoop.sol
- interface/IVerifier.sol
- libs/asset/AssetPool.sol
- libs/asset/ERC20AssetPool.sol
- libs/asset/IERC20Metadata.sol
- libs/asset/MainAssetPool.sol
- libs/verifiers/Rollup1Verifier.sol
- libs/verifiers/Transaction1x0Verifier.sol

Zokrates circuits:

- Commitment.zok
- JoinSplit.zok
- KeccakBatch.zok
- MerkleTree.zok
- MerkleTreeBatchUpdater.zok

- MerkleTreeBuilder.zok
- MerkleTreeUpdater.zok
- Ownership.zok
- Rollup1.zok
- Rollup4.zok
- Rollup16.zok
- Rollup64.zok
- Rollup256.zok
- SerialNumber.zok
- Sha256Batch.zok
- SignatureHash.zok
- Transaction1x0.zok
- Transaction1x1.zok
- Transaction1x2.zok
- Transaction2x0.zok
- Transaction2x1.zok
- Transaction2x2.zok

The fixes were provided in the 151ff80 commit.

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** Transaction1x0Verifier.sol

Recommendation This library should be moved to a separate file named "Transaction1x0Pairing.sol".

Listing 1:

```
3 library Transaction1x0Pairing {
```

3.2 CVF-2

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Transaction1x0Verifier.sol

Description This library looks identical to the "Rollup1Pairing" library defined in "Rollup1Verifier.sol".
Recommendation Consider defining the library in a shared place.

Listing 2:

```
3 library Transaction1x0Pairing {
```

3.3 CVF-3

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Transaction1x0Verifier.sol

Description In the yellow paper and ERC-1025 this value is known as "p" and "q" is used for another value: 2188(...)5617.
Recommendation Consider using the same notation as in the specification.

Listing 3:

```
33 uint256 q =
   ↪ 21888242871839275222246405745257275088696311157297823662689037894645226208583;
```

3.4 CVF-4

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Transaction1x0Verifier.sol

Recommendation This value should be a named constant.

Listing 4:

```
33 uint256 q =
    ↪ 21888242871839275222246405745257275088696311157297823662689
    ↪
037894645226208583;
```

3.5 CVF-5

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Transaction1x0Verifier.sol

Recommendation Should be "p.X % q" to guarantee that returned values are field element.

Listing 5:

```
35 return G1Point(p.X, q - (p.Y % q));
```

3.6 CVF-6

- **Severity** Major
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** Transaction1x0Verifier.sol

Description The latter argument overflows for p.Y==0.
Recommendation Consider taking both outputs modulo q in order to ensure they are field elements

Listing 6:

```
35 return G1Point(p.X, q - (p.Y % q));
```

3.7 CVF-7

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Transaction1x0Verifier.sol

Description The success flag is checked twice.
Recommendation Consider removing one of the checks.
Client Comment Keep this redundant check for failing fast.

Listing 7:

```
46    success := staticcall(sub(gas(), 2000), 6, input, 0xc0, r, 0
        ↪ x60)
      switch success
      case 0 {
        invalid()
50    }

52  require(success);

62    success := staticcall(sub(gas(), 2000), 7, input, 0x80, r, 0
        ↪ x60)
      switch success
      case 0 {
        invalid()
      }

68  require(success);

88    switch success
      case 0 {
90      invalid()
      }

93  require(success);
```

3.8 CVF-8

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Transaction1x0Verifier.sol

Recommendation The "revert" opcode would be more efficient as it doesn't waste all the remaining gas.

Listing 8:

```
49 invalid()

65 invalid()

90 invalid()
```

3.9 CVF-9

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Transaction1x0Verifier.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary. Also it would be possible to pass the content of such array to the precompile as is, without copying."
Client Comment The interface is more clear when defining parameters separately.

Listing 9:

```
71 function pairing(G1Point[] memory p1, G2Point[] memory p2)
   ↪ internal view returns (bool) {
```

3.10 CVF-10

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Transaction1x0Verifier.sol

Description The coordinates of the points are copied twice: once before calling the "pairing" function and another time inside "pairing".
Recommendation Consider refactoring to avoid double copying.

Listing 10:

```
105 p1[0] = a1;
    p1[1] = b1;
    p2[0] = a2;
    p2[1] = b2;
    return pairing(p1, p2);

122 p1[0] = a1;
    p1[1] = b1;
    p1[2] = c1;
    p2[0] = a2;
    p2[1] = b2;
    p2[2] = c2;
    return pairing(p1, p2);

143 p1[0] = a1;
    p1[1] = b1;
    p1[2] = c1;
    p1[3] = d1;
    p2[0] = a2;
    p2[1] = b2;
    p2[2] = c2;
150 p2[3] = d2;
    return pairing(p1, p2);
```

3.11 CVF-11

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Transaction1x0Verifier.sol

Description This contract looks very similar to the "Rollup1Verifier " contract.
Recommendation Consider extracting a shared abstract base contract to avoid code duplication.

Listing 11:

```
155 contract Transaction1x0Verifier {
```

3.12 CVF-12

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Transaction1x0Verifier.sol

Recommendation The return type should be "bool".

Listing 12:

```
236 function verify(uint256[] memory input, Proof memory proof)
    ↪ internal view returns (uint256) {
```

3.13 CVF-13

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Transaction1x0Verifier.sol

Recommendation This value should be a named constant.
Client Comment Local variables spend less gas.

Listing 13:

```
237 uint256 snark_scalar_field =
    ↪ 21888242871839275222246405745257275088548364400416034343698204186575808495617;
```

3.14 CVF-14

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Transaction1x0Verifier.sol

Recommendation Should be "else return" for readability.

Listing 14:

```
261 return 0;
```

3.15 CVF-15

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** Transaction1x0Verifier.sol

Description 'Proof' and 'input' are not checked to be curve points. It may be possible that invalid inputs result in a successful check as the curve operations do not check the inputs either.

Listing 15:

```
264 function verifyTx(Proof memory proof, uint256[] memory input)
    ↪ public view returns (bool r) {
```

3.16 CVF-16

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Transaction1x0Verifier.sol

Recommendation "This could be simplified as: return verify(input, proof) == 0;"

Listing 16:

```
266 if (verify(input, proof) == 0) {
      return true;
    } else {
      return false;
270 }
```

3.17 CVF-17

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

Description Declaring top-level structures in a file named after a contract makes it harder to navigate through the code.
Recommendation Consider either moving the structured into the contract or moving the structures to a separate file named "types.sol" or something like this.

Listing 17:

```
11 struct CommitmentLeaf {

16 struct WrappedVerifier {
```

3.18 CVF-18

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

Description The same constant is defined in the "MystickoV2Loop" contract.
Recommendation Consider defining once and a shared place.

Listing 18:

```
22 uint256 constant FIELD_SIZE =
    ↪ 21888242871839275222246405745257275088548364400416034343698
    ↪
204186575808495617;
```

3.19 CVF-19

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

Description There is not access level specifies for these variables, so internal access will be used by default.
Recommendation Consider explicitly specifying an access level.

Listing 19:

```
22 uint256 constant FIELD_SIZE =
     ↪ 21888242871839275222246405745257275088548364400416034343698204186575808495617;

24 mapping(uint32 => mapping(uint32 => WrappedVerifier))
     ↪ transactVerifiers;
   mapping(uint32 => WrappedVerifier) rollupVerifiers;

27 mapping(uint256 => bool) historicCommitments;
   mapping(uint256 => bool) spentSerialNumbers;

30 mapping(uint256 => CommitmentLeaf) commitmentQueue;
   uint256 commitmentQueueSize = 0;
   uint256 commitmentIncludedCount = 0;

34 uint256 immutable treeCapacity;
   mapping(uint32 => uint256) rootHistory;
   uint256 currentRoot;
   uint32 currentRootIndex = 0;
   uint32 immutable rootHistoryLength;

40 address operator;
   uint256 minRollupFee;
   mapping(address => bool) rollupWhitelist;
   mapping(address => bool) enqueueWhitelist;

45 bool verifierUpdateDisabled;
   bool rollupWhitelistDisabled;
```

3.20 CVF-20

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

Description The semantics of the keys for this mapping is unclear.
Recommendation Consider documenting.

Listing 20:

```
24 mapping(uint32 => mapping(uint32 => WrappedVerifier))
     ↪ transactVerifiers;
   mapping(uint32 => WrappedVerifier) rollupVerifiers;

30 mapping(uint256 => CommitmentLeaf) commitmentQueue;

35 mapping(uint32 => uint256) rootHistory;
```

3.21 CVF-21

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

Recommendation Using root=>bool mapping would make history search much cheaper.

Listing 21:

```
35 mapping(uint32 => uint256) rootHistory;
```

3.22 CVF-22

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

Description String error messages are suboptimal.
Recommendation Consider using named errors instead.

Listing 22:

```
49  require(msg.sender == operator, "only operator.");

54  require(rollupWhitelistDisabled || rollupWhitelist[msg.sender],
      ↪ "only whitelisted roller.");

59  require(enqueueWhitelist[msg.sender], "only whitelisted sender
      ↪ .");

73  require(_rootHistoryLength > 0, "_rootHistoryLength should be
      ↪ greater than 0");

87  require(_request.rollupFee >= minRollupFee, "rollup fee too few
      ↪ ");
    require(commitmentIncludedCount + commitmentQueueSize <
      ↪ treeCapacity, "tree is full");
    require(!historicCommitments[_request.commitment], "the
      ↪ commitment has been submitted");

101 require(!isKnownRoot(_request.newRoot), "newRoot is duplicated")
      ↪ ;

106   "invalid rollupSize"

108 require(commitmentIncludedCount % _request.rollupSize == 0, "
      ↪ invalid rollupSize at current state");

117   require(commitmentQueue[index].commitment != 0, "index out of
        ↪ bound");

126 require(_request.leafHash == expectedLeafHash, "invalid leafHash
      ↪ ");

133 require(verified, "invalid proof");
    (150, 159, 163, 167, 176, 183, 221, 228, 237, 243, 267)
```

3.23 CVF-23

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

Recommendation Events are usually named via nouns.

Listing 23:

```
63 event CommitmentQueued(

69 event CommitmentIncluded(uint256 indexed commitment);
70 event CommitmentSpent(uint256 indexed rootHash, uint256 indexed
   ↪ serialNumber);
```

3.24 CVF-24

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

Recommendation Shift would be more efficient.

Listing 24:

```
76 treeCapacity = 2**uint256(_treeHeight);
```

3.25 CVF-25

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

Description The maximum "_treeHeight" value that could be used here is 255, while the "_treeHeight" argument has type "uint32".
Recommendation Consider changing the type to "uint8".

Listing 25:

```
76 treeCapacity = 2**uint256(_treeHeight);
```

3.26 CVF-26

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

Description This function always returns true.
Recommendation Consider returning nothing.

Listing 26:

```
85 returns (bool)
```

3.27 CVF-27

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

Description The expression "rollupVerifiers[_request.rollupSize]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 27:

```
105     rollupVerifiers[_request.rollupSize].enabled,

132 bool verified = rollupVerifiers[_request.rollupSize].verifier.
    ↪ verifyTx(_request.proof, inputs);
```

3.28 CVF-28

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** CommitmentPool.sol

Description The value "commitmentIncludedCount" is read from the storage several times.
Recommendation Consider reading once and caching in a local variable.

Listing 28:

```
108 require(commitmentIncludedCount % _request.rollupSize == 0, "
    ↪ invalid rollupSize at current state");
    uint256 pathIndices = _pathIndices(commitmentIncludedCount,
    ↪ _request.rollupSize);

113   uint256 index = commitmentIncludedCount;
      index < commitmentIncludedCount + _request.rollupSize;

119   leaves[index - commitmentIncludedCount] = commitment;

135 commitmentIncludedCount = commitmentIncludedCount + _request.
    ↪ rollupSize;
```

3.29 CVF-29

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

Description The expression "commitmentIncludedCount + _request.rollupSize" is calculated on every loop iteration.
Recommendation Consider calculating once and reusing.

Listing 29:

```
114 index < commitmentIncludedCount + _request.rollupSize;
```

3.30 CVF-30

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

Description The expression "commitmentIncludedCount + _request.rollupSize" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 30:

```
114   index < commitmentIncludedCount + _request.rollupSize;

135 commitmentIncludedCount = commitmentIncludedCount + _request.
      ↪ rollupSize;
```

3.31 CVF-31

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

Description The expression "commitmentQueue[index]" is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 31:

```
117 require(commitmentQueue[index].commitment != 0, "index out of
      ↪ bound");
    uint256 commitment = commitmentQueue[index].commitment;

120 totalRollupFee = totalRollupFee + commitmentQueue[index].
      ↪ rollupFee;
    delete commitmentQueue[index];
```

3.32 CVF-32

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

Description The expression "commintemtQueue[index].commitment" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 32:

```
117 require(commitmentQueue[index].commitment != 0, "index out of
      ↪ bound");
    uint256 commitment = commitmentQueue[index].commitment;
```

3.33 CVF-33

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

Recommendation This line could be simplified using the "+=" operator.

Listing 33:

```
120 totalRollupFee = totalRollupFee + commitmentQueue[index].
    ↪ rollupFee;
```

3.34 CVF-34

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

Recommendation This line could be simplified using the "-=" operator.

Listing 34:

```
122 commitmentQueueSize = commitmentQueueSize - 1;
```

3.35 CVF-35

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** CommitmentPool.sol

Description The "commitmentQueueSize" variable is updated on every loop iteration.
Recommendation Consider updating once after the loop.

Listing 35:

```
122 commitmentQueueSize = commitmentQueueSize - 1;
```

3.36 CVF-36

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

Description This check makes the "_request.leafHash" field redundant.
Recommendation Consider removing this field.
Client Comment Keep this redundant check for failing fast.

Listing 36:

```
126 require(_request.leafHash == expectedLeafHash, "invalid leafHash
    ↪ ");
```

3.37 CVF-37

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

Description It is not checked that "_request.newRoot" is a field element.

Listing 37:

```
129 inputs[1] = _request.newRoot;
```

3.38 CVF-38

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** CommitmentPool.sol

Description Reentrancy is possible here.
Recommendation Consider calling external contracts after updating the state.

Listing 38:

```
134 _processRollupFeeTransfer(totalRollupFee);
```

3.39 CVF-39

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

Recommendation This line could be simplified using the "+=" operator.

Listing 39:

```
135 commitmentIncludedCount = commitmentIncludedCount + _request.
    ↪ rollupSize;
```

3.40 CVF-40

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

Description The message value is not used in this function. It is unclear, why is ti payable.

Listing 40:

```
143 payable
```

3.41 CVF-41

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** CommitmentPool.sol

Description Overflow is possible here.
Recommendation Consider using safe conversion.

Listing 41:

```
147 uint32 numInputs = uint32(_request.serialNumbers.length);
    uint32 numOutputs = uint32(_request.outCommitments.length);
```

3.42 CVF-42

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

Description This check makes the "_request.sigPk" field redundant.
Recommendation Consider removing this field.
Client Comment Keep this redundant check for failing fast.

Listing 42:

```
159 require(_request.sigPk == bytes32(uint256(uint160(recoveredSigPk
    ↪ ))), "invalid signature");
```

3.43 CVF-43

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

Description It is not ensured that the input values assigned here are field elements.

Listing 43:

```
164 inputs[0] = _request.rootHash;

168   inputs[i + 1] = _request.serialNumbers[i];
      inputs[i + 1 + numInputs] = _request.sigHashes[i];

172 inputs[2 * numInputs + 2] = uint256(_request.publicAmount);
    inputs[2 * numInputs + 3] = uint256(_request.relayerFeeAmount);

178   inputs[2 * numInputs + 4 + i] = _request.outCommitments[i];
      inputs[2 * numInputs + numOutputs + 4 + i] = _request.
        ↪ outRollupFees[i];
```

3.44 CVF-44

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

Description The expression "_request.serialNumbers[i]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 44:

```
167 require(!spentSerialNumbers[_request.serialNumbers[i]], "the
    ↪ note has been spent");
    inputs[i + 1] = _request.serialNumbers[i];
```

3.45 CVF-45

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

Description The expression "1 + numInputs" is calculated on every loop iteration.
Recommendation Consider calculating once before the loop.

Listing 45:

```
169 inputs[i + 1 + numInputs] = _request.sigHashes[i];
```

3.46 CVF-46

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

Description The expression "2 * numInputs" is calculated several times.
Recommendation Consider calculating once before the loop.

Listing 46:

```
171 inputs[2 * numInputs + 1] = uint256(_request.sigPk);
    inputs[2 * numInputs + 2] = uint256(_request.publicAmount);
    inputs[2 * numInputs + 3] = uint256(_request.relayerFeeAmount);

178   inputs[2 * numInputs + 4 + i] = _request.outCommitments[i];
      inputs[2 * numInputs + numOutputs + 4 + i] = _request.
        ↪ outRollupFees[i];
```

3.47 CVF-47

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

Recommendation The conversions are redundant, as the corresponding fields are already "uint256".

Listing 47:

```
172 inputs[2 * numInputs + 2] = uint256(_request.publicAmount);
    inputs[2 * numInputs + 3] = uint256(_request.relayerFeeAmount);
```

3.48 CVF-48

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** CommitmentPool.sol

Description These parameters should undergo range checks.

Listing 48:

```
172 inputs[2 * numInputs + 2] = uint256(_request.publicAmount);
    inputs[2 * numInputs + 3] = uint256(_request.relayerFeeAmount);

179   inputs[2 * numInputs + numOutputs + 4 + i] = _request.
        ↪ outRollupFees[i];
```

3.49 CVF-49

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

Recommendation Consider making serial number dependent on the leaf index. This would allow using duplicate commitments.

Listing 49:

```
176 require(!historicCommitments[_request.outCommitments[i]], "
      ↪ duplicate commitment");
```

3.50 CVF-50

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

Description The names are confusing, as these functions set a particular flag value rather than toggle a flag.
Recommendation Consider renaming.

Listing 50:

```
208 function toggleRollupWhitelist(bool _state) external
      ↪ onlyOperator {

212 function toggleVerifierUpdate(bool _state) external onlyOperator
      ↪  {

275 function toggleSanctionCheck(bool _check) external onlyOperator
      ↪ {
```

3.51 CVF-51

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

Description These functions should emit some events.

Listing 51:

```
208 function toggleRollupWhitelist(bool _state) external
      ↪ onlyOperator {

212 function toggleVerifierUpdate(bool _state) external onlyOperator
      ↪  {

216 function enableTransactVerifier(

227 function disableTransactVerifier(uint32 _numInputs, uint32
      ↪ _numOutputs) external onlyOperator {

236 function enableRollupVerifier(uint32 _rollupSize, address
      ↪ _rollupVerifier) external onlyOperator {

242 function disableRollupVerifier(uint32 _rollupSize) external
      ↪ onlyOperator {

250 function addRollupWhitelist(address _roller) external
      ↪ onlyOperator {

254 function removeRollupWhitelist(address _roller) external
      ↪ onlyOperator {

258 function addEnqueueWhitelist(address _actor) external
      ↪ onlyOperator {

262 function removeEnqueueWhitelist(address _actor) external
      ↪ onlyOperator {

266 function setMinRollupFee(uint256 _minRollupFee) external
      ↪ onlyOperator {

271 function changeOperator(address _newOperator) external
      ↪ onlyOperator {

275 function toggleSanctionCheck(bool _check) external onlyOperator
      ↪ {

279 function updateSanctionContractAddress(address _sanction)
      ↪ external onlyOperator {
```

3.52 CVF-52

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

Recommendation The type of this argument should be "IVerifier".

Listing 52:

```
219 address _transactVerifier
```

3.53 CVF-53

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

Description These checks don't save gas.
Recommendation Consider removing them.

Listing 53:

```
231 if (transactVerifiers[_numInputs][_numOutputs].enabled) {
```

```
245 if (rollupVerifiers[_rollupSize].enabled) {
```

3.54 CVF-54

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

Recommendation The type of the "_rollingVerifier" argument should be "IVerifier".

Listing 54:

```
236 function enableRollupVerifier(uint32 _rollupSize, address
    ↪ _rollupVerifier) external onlyOperator {
```

3.55 CVF-55

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

Recommendation It should be ensured that "_rollupSize" is a power of 2. A cheap way to do this is to require _rollupSize & _rollupSize - 1 == 0.

Listing 55:

```
236 function enableRollupVerifier(uint32 _rollupSize, address
    ↪ _rollupVerifier) external onlyOperator {
```

3.56 CVF-56

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

Recommendation The argument type should be "SanctionsList".

Listing 56:

```
279 function updateSanctionContractAddress(address _sanction)
    ↪ external onlyOperator {
```

3.57 CVF-57

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

Recommendation These functions wouldn't be necessary if the corresponding mappings would be declared as public.

Listing 57:

```
283 function isHistoricCommitment(uint256 _commitment) public view
    ↪ returns (bool) {

287 function isSpentSerialNumber(uint256 _serialNumber) public view
    ↪ returns (bool) {
```

3.58 CVF-58

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

Description This functions returns true for a zero root in case the root history has unfilled slots.
Recommendation Consider explicitly returning false for a zero root.
Client Comment New root is guranteed not to be zero root.

Listing 58:

```
291 function isKnownRoot(uint256 root) public view returns (bool) {
```

3.59 CVF-59

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

Recommendation It would be more efficient to split this loop into two ordinary "for" loops: one from the current index to zero, and another from the history length to the current index.

Listing 59:

```
293 do {
      if (root == rootHistory[i]) {
        return true;
      }
      if (i == 0) {
        i = rootHistoryLength;
      }
300   i--;
    } while (i != currentRootIndex);
```

3.60 CVF-60

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

Recommendation These functions wouldn't be necessary if the corresponding variables would be declared as public.

Listing 60:

```
305 function getTreeCapacity() public view returns (uint256) {

309 function getRootHistoryLength() public view returns (uint32) {

313 function isVerifierUpdateDisabled() public view returns (bool) {

317 function isRollupWhitelistDisabled() public view returns (bool)
    ↪ {

321 function getMinRollupFee() public view returns (uint256) {

325 function getCommitmentIncludedCount() public view returns (
    ↪ uint256) {
```

3.61 CVF-61

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

Recommendation This line could be simplified using the "+=" operator.

Listing 61:

```
336 commitmentQueueSize = commitmentQueueSize + 1;
```

3.62 CVF-62

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

Recommendation This function could be optimized by using a binary decision tree rather than a linear one: if (_nth < 16) { if (_nth < 8) { if (_nth < 4) { if (_nth < 2) { if (_nth == 0) return ...; else return ...; } else { ... etc ...
Client Comment Only called by the constructor function.

Listing 62:

```
340 function _zeros(uint32 _nth) internal pure returns (uint256) {
```

3.63 CVF-63

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** CommitmentPool.sol

Recommendation Should be "else return" for readability.

Listing 63:

```
408 return 0;
```

3.64 CVF-64

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

Description The function silently returns zero on unsupported input.
Recommendation Consider reverting in such a case.

Listing 64:

```
408 return 0;
```

3.65 CVF-65

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

Recommendation This function uses linear approach to count bits in a number. Bisection approach would be more efficient: function _pathIndices(uint256 _fullPath, uint32 _rollupSize) public pure returns (uint256) { if (_rollupSize >= 0x10000) { _rollupSize »= 16; _fullPath »= 16; } if (_rollupSize >= 0x100) { _rollupSize »= 8; _fullPath »= 8; } if (_rollupSize >= 0x10) { _rollupSize »= 4; _fullPath »= 4; } if (_rollupSize >= 0x4) { _rollupSize »= 2; _fullPath »= 2; } if (_rollupSize >= 0x2) { /* _rollupSize »= 1; */ _fullPath »= 1; } return _fullPath; }"

Listing 65:

```
411 function _pathIndices(uint256 _fullPath, uint32 _rollupSize)
      ↪ internal pure returns (uint256) {
    _rollupSize >>= 1;
    while (_rollupSize != 0) {
      _fullPath >>= 1;
      _rollupSize >>= 1;
    }
    return _fullPath;
  }
```

3.66 CVF-66

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** CommitmentPool.sol

Description A new bytes array is allocated on each iteration and the contents of the previous array is copies into it, thus memory usage and gas consumption is O(n^2).
Recommendation Consider appending to a single array. This would require using assembly.

Listing 66:

```
423 requestBytes = abi.encodePacked(requestBytes, _request.
    ↪ outEncryptedNotes[i]);
```

3.67 CVF-67

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** CommitmentPool.sol

Description The boundaries between individual encrypted notes are not preserved when hashing.
Recommendation Consider preserving them somehow.

Listing 67:

```
423 requestBytes = abi.encodePacked(requestBytes, _request.
    ↪ outEncryptedNotes[i]);
```

3.68 CVF-68

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

Description String are inefficient.
Recommendation Consider returning a enum constant instead.

Listing 68:

```
16 function assetType() public view virtual returns (string memory)
    ↪ ;
```

3.69 CVF-69

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** JoinSplit.zok

Description The order of arguments in these functions is different which is error prone.
Recommendation Consider using the same order.

Listing 69:

```
47 def checkOwnerships<N>(\
     field[N] secretKeys, \
     field[N] publicKeys, \

52     bool match = ownershipChecker(publicKeys[i], secretKeys[i],
         ↪ context)
```

3.70 CVF-70

- **Severity** Critical
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** JoinSplit.zok

Description Overflow is possible here.
Recommendation Consider checking that "inAmount[i]", "outAmount[i]", "rollupFeeAmount[i]", "publicAmount", and "relayerFeeAmount" are in a certain safe range, or preventing overflow is some other way. Note that the assert (x>0) only verifies that the value is non-zero.

Listing 70:

```
77   inTotal = inTotal + inAmount[i]

81   assert(outAmount[i] > 0)
     outTotal = outTotal + outAmount[i] + rollupFeeAmounts[i]

84 outTotal = outTotal + publicAmount + relayerFeeAmount
```

3.71 CVF-71

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** JoinSplit.zok

Recommendation This should be asserted to be a curve point.
Client Comment This two numbers are copied from https://eips.ethereum.org/EIPS/eip-2494 which is supposed to be on curve.

Listing 71:

```
112 ownershipParams.Gu =
    ↪ 529961924064155128163486558351829703028287447219077289408652
    ↪
1144482721001553
ownershipParams.Gv =
    ↪ 169501507984606577179586255678218345503016631616247077872228
    ↪
15936182638968203
```

3.72 CVF-72

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

Recommendation It would be more efficient to have a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.
Client Comment The interface is more clear when defining parameters separately.

Listing 72:

```
24 uint256[] serialNumbers;
   uint256[] sigHashes;
```

3.73 CVF-73

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

Recommendation If would be more efficient to have a single array of struct with three fields, rather than three parallel arrays. This would also make the length checks unnecessary.
Client Comment The interface is more clear when defining parameters separately.

Listing 73:

```
29 uint256[] outCommitments;
30 uint256[] outRollupFees;

33 bytes[] outEncryptedNotes;
```

3.74 CVF-74

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

Description The semantics of the returned value is unclear.
Recommendation Consider documenting.

Listing 74:

```
36 function enqueue(CommitmentRequest memory _request, address
   ↪ _executor) external returns (bool);
```

3.75 CVF-75

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** SerialNumber.zok

Recommendation Consider commenting it is a nullifier key to explain the two-step format.

Listing 75:

```
4 field nk = poseidon([secretKey])
```

3.76 CVF-76

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Rollup1Verifier.sol

Recommendation This library should be moved to a separate file named "Rollup1Pairing.sol".

Listing 76:

```
3 library Rollup1Pairing {
```

3.77 CVF-77

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Rollup1Verifier.sol

Description This library looks identical to the "Transaction1x0Pairing " library defined in "Transaction1x0Veridier.sol".
Recommendation Consider defining the library in a shared place.

Listing 77:

```
3 library Rollup1Pairing {
```

3.78 CVF-78

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Rollup1Verifier.sol

Description In the yellow paper and ERC-1025 this value is known as "p" and "q" is used for another value: 2188(...)5617.
Recommendation Consider using the same notation as in the specification.

Listing 78:

```
32 uint256 q =
    ↪ 21888242871839275222246405745257275088696311157297823662689
    ↪
037894645226208583;
```

3.79 CVF-79

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Rollup1Verifier.sol

Recommendation This value should be a named constant.

Listing 79:

```
32 uint256 q =
    ↪ 21888242871839275222246405745257275088696311157297823662689037894645226208583;
```

3.80 CVF-80

- **Severity** Major
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** Rollup1Verifier.sol

Description The latter argument overflows for p.Y==0.
Recommendation Consider taking both outputs modulo q in order to ensure they are field elements

Listing 80:

```
34 return G1Point(p.X, q - (p.Y % q));
```

3.81 CVF-81

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Rollup1Verifier.sol

Description The success flag is checked twice.
Recommendation Consider removing one of the checks.
Client Comment Keep this redundant check for failing fast.

Listing 81:

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

51  require(success);

62    switch success
      case 0 {
        invalid()
      }

67  require(success);

87    switch success
      case 0 {
        invalid()
90    }

92  require(success);
```

3.82 CVF-82

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Rollup1Verifier.sol

Recommendation The "revert" opcode would be more efficient as it doesn't waste all the remaining gas.

Listing 82:

```
48  invalid()

64  invalid()

89  invalid()
```

3.83 CVF-83

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Rollup1Verifier.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary. Also it would be possible to pass the content of such array to the precompile as is, without copying.
Client Comment The interface is more clear when defining parameters separately.

Listing 83:

```
70 function pairing(G1Point[] memory p1, G2Point[] memory p2)
   ↪ internal view returns (bool) {
```

3.84 CVF-84

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Rollup1Verifier.sol

Description The coordinates of the points are copied twice: once before calling the "pairing" function and another time inside "pairing".
Recommendation Consider refactoring to avoid double copying.

Listing 84:

```
104 p1[0] = a1;
    p1[1] = b1;
    p2[0] = a2;
    p2[1] = b2;
    return pairing(p1, p2);

121 p1[0] = a1;
    p1[1] = b1;
    p1[2] = c1;
    p2[0] = a2;
    p2[1] = b2;
    p2[2] = c2;
    return pairing(p1, p2);

142 p1[0] = a1;
    p1[1] = b1;
    p1[2] = c1;
    p1[3] = d1;
    p2[0] = a2;
    p2[1] = b2;
    p2[2] = c2;
    p2[3] = d2;
150 return pairing(p1, p2);
```

3.85 CVF-85

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Rollup1Verifier.sol

Description This contract looks very similar to the "Transaction1x0Verifier" contract.
Recommendation Consider extracting a shared abstract base contract to avoid code duplication.

Listing 85:

```
154 contract Rollup1Verifier {
```

3.86 CVF-86

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Rollup1Verifier.sol

Recommendation The return type should be "bool".

Listing 86:

```
227 function verify(uint256[] memory input, Proof memory proof)
    ↪ internal view returns (uint256) {
```

3.87 CVF-87

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Rollup1Verifier.sol

Recommendation This value should be a named constant.
Client Comment Local variables spend less gas.

Listing 87:

```
228 uint256 snark_scalar_field =
    ↪ 21888242871839275222246405745257275088548364400416034343698204186575808495617;
```

3.88 CVF-88

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Rollup1Verifier.sol

Recommendation Should be "else return" for readability.

Listing 88:

```
249 return 0;
```

3.89 CVF-89

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** Rollup1Verifier.sol

Description 'Proof' and 'input' are not checked to be curve points. It may be possible that invalid inputs result in a successful check as the curve operations do not check the inputs either.

Listing 89:

```
252 function verifyTx(Proof memory proof, uint256[] memory input)
    ↪ public view returns (bool r) {
```

3.90 CVF-90

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Rollup1Verifier.sol

Recommendation This could be simplified as: return verify(input, proof) == 0;

Listing 90:

```
254 if (verify(input, proof) == 0) {
      return true;
    } else {
      return false;
    }
```

3.91 CVF-91

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** MystikoV2Loop.sol

Description The same constant is defined in the "CommitmentPool" contract.
Recommendation Consider defining once and a shared place.

Listing 91:

```
12 uint256 constant FIELD_SIZE =
    ↪ 21888242871839275222246405745257275088548364400416034343698
    ↪
204186575808495617;
```

3.92 CVF-92

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MystikoV2Loop.sol

Description There is no access level specified for these variables, so internal access will be used by default.
Recommendation Consider explicitly specifying an access level.

Listing 92:

```
12 uint256 constant FIELD_SIZE =
     ↪ 21888242871839275222246405745257275088548364400416034343698
     ↪
   204186575808495617;

14 IHasher3 hasher3;

16 address associatedCommitmentPool;
   uint256 minAmount;

19 address operator;

21 bool depositsDisabled;
```

3.93 CVF-93

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MystikoV2Loop.sol

Recommendation The type of this variable should be "ICommitmentPool".

Listing 93:

```
16 address associatedCommitmentPool;
```

3.94 CVF-94

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** MystikoV2Loop.sol

Recommendation The argument type should be "IHasher3".

Listing 94:

```
28 constructor(address _hasher3) {
```

3.95 CVF-95

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MystikoV2Loop.sol

Recommendation The argument type should be "ICommitmentPool".

Listing 95:

```
33 function setAssociatedCommitmentPool(address
     ↪ _commitmentPoolAddress) external onlyOperator {
```

3.96 CVF-96

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MystikoV2Loop.sol

Description These functions should emit some events.

Listing 96:

```
37 function setMinAmount(uint256 _minAmount) external onlyOperator
     ↪ {

51 function deposit(DepositRequest memory _request) external
     ↪ payable override {

80 function toggleDeposits(bool _state) external onlyOperator {

84 function changeOperator(address _newOperator) external
     ↪ onlyOperator {

88 function toggleSanctionCheck(bool _check) external onlyOperator
     ↪ {

92 function updateSanctionContractAddress(address _sanction)
     ↪ external onlyOperator {
```

3.97 CVF-97

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** MystikoV2Loop.sol

Description It is not checked than this value is a field element.
Recommendation Consider adding such check.

Listing 97:

```
44 uint128 _randomS
```

3.98 CVF-98

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MystikoV2Loop.sol

Recommendation The conversion to "uint256" is redundant, as "uint128" is automatically converted to "uint256".

Listing 98:

```
48 return hasher3.poseidon([_hashK, _amount, uint256(_randomS)]);
```

3.99 CVF-99

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** MystikoV2Loop.sol

Recommendation Should be ‘too small‘.

Listing 99:

```
53 require(_request.amount >= minAmount, "amount too few");
```

3.100 CVF-100

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MystikoV2Loop.sol

Description This check makes the "_request.commitment" field redundant.
Recommendation Consider removing the field.
Client Comment Keep this redundant check for failing fast.

Listing 100:

```
55 require(_request.commitment == calculatedCommitment, "commitment
   ↪ hash incorrect");
```

3.101 CVF-101

- **Severity** Moderate
- **Status** Fixed
- **Category** Procedural
- **Source** MystikoV2Loop.sol

Description Here an external contract is called before updating the state. This could be used for reentrancy attacks.
Recommendation Consider calling the external contract after updating the state.

Listing 101:

```
67 _processDepositTransfer(associatedCommitmentPool, _amount +
   ↪ _rollupFee, 0);
```

3.102 CVF-102

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** MystikoV2Loop.sol

Description Despite the names, these functions don't toggle flags but rather set them to certain values.
Recommendation Consider renaming.

Listing 102:

```
80 function toggleDeposits(bool _state) external onlyOperator {

88 function toggleSanctionCheck(bool _check) external onlyOperator
   ↪ {
```

3.103 CVF-103

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MystikoV2Loop.sol

Recommendation These functions wouldn't be necessary if the corresponding variables would be declared as "public".

Listing 103:

```
100 function getMinAmount() public view returns (uint256) {

104 function isDepositsDisabled() public view returns (bool) {
```

3.104 CVF-104

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** MystikoV2LoopMain.sol

Recommendation The argument type should be "Ihasher3".

Listing 104:

```
7 constructor(address _hasher3) MystikoV2Loop(_hasher3) {}
```

3.105 CVF-105

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** MystikoV2LoopMain.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 105:

```
7 constructor(address _hasher3) MystikoV2Loop(_hasher3) {}
```

3.106 CVF-106

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** MystikoV2LoopERC20.sol

Recommendation The type of the arguments should be "IHasher3" and "IERC20" respectively.

Listing 106:

```
7 constructor(address _hasher3, address _token) MystikoV2Loop(
    ↪ _hasher3) ERC20AssetPool(_token) {}
```

3.107 CVF-107

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** MystikoV2LoopERC20.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 107:

```
7 constructor(address _hasher3, address _token) MystikoV2Loop(
    ↪ _hasher3) ERC20AssetPool(_token) {}
```

3.108 CVF-108

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

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 108:

```
9 {}

11 receive() external payable {}
```

3.109 CVF-109

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** CommitmentPoolERC20.sol

Recommendation The type of this argument should be "IERC20".

Listing 109:

```
10 address _token
```

3.110 CVF-110

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** CommitmentPoolERC20.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 110:

```
11 ) CommitmentPool(_treeHeight, _rootHistoryLength) ERC20AssetPool
     ↪ (_token) {}
```

3.111 CVF-111

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

Recommendation This interface should be moved to a separate file named "SanctionsList.sol".

Listing 111:

```
3 interface SanctionsList {
```

3.112 CVF-112

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

Description Unlike other interfaces in this project, the name of this interface doesn't have the "I" prefix.
Recommendation Consider adding the prefix for consistency.

Listing 112:

```
3 interface SanctionsList {
```

3.113 CVF-113

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

Description Despite the name, this interface doesn't look like a list, but rather like a set or even oracle.
Recommendation Consider renaming.
Client Comment The interface name is defined by Chainalysis oracle.

Listing 113:

```
3 interface SanctionsList {
```

3.114 CVF-114

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

Description Hardcoding mainnet addresses is a bad practice, as it makes it harder to test contracts.
Recommendation Consider passing the sanctions contract address as a constructor argument and storing in an immutable variable.
Client Comment Hardcoding for sanction contract address, because it is same address for all chain, and we also support sanction contract address change.

Listing 114:

```
8 address sanctionsContract = 0
    ↪ x40C57923924B5c5c5455c48D93317139ADDaC8fb;
```

3.115 CVF-115

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

Description There are no access levels specified for these variables, not internal access will be used by default.
Recommendation Consider explicitly specifying an access level.

Listing 115:

```
8 address sanctionsContract = 0
    ↪ x40C57923924B5c5c5455c48D93317139ADDaC8fb;
  bool sanctionCheckDisabled = false;
```

3.116 CVF-116

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

Recommendation The type of this variable should be "SanctionsList".

Listing 116:

```
8  address sanctionsContract = 0
       ↪ x40C57923924B5c5c5455c48D93317139ADDaC8fb;
```

3.117 CVF-117

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

Recommendation These functions wouldn't be necessary if the corresponding variables would be declared as public.

Listing 117:

```
11 function isSanctionCheckDisabled() public view returns (bool) {

15 function getSanctionsContract() public view returns (address) {
```

3.118 CVF-118

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

Recommendation The return type should be "SanctionsList".

Listing 118:

```
15 function getSanctionsContract() public view returns (address) {
```

3.119 CVF-119

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

Recommendation This function could be simplified as: return !sanctionsListDisabled && SanctionsList (sanctionsContract).isSanctioned(_addr);

Listing 119:

```
19 function isSanctioned(address _addr) internal returns (bool) {
```

3.120 CVF-120

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

Description String error messages are suboptimal.
Recommendation Consider using named errors instead.

Listing 120:

```
11 require(msg.value == amount + bridgeFee, "insufficient token");

13 require(success, "amount transfer failed");

18 require(success, "executor fee transfer failed");

23 require(success, "rollup fee transfer failed");

27 require(msg.value == 0, "no mainnet token allowed");

29 require(success, "withdraw failed");
```

3.121 CVF-121

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

Description There is no complimentary check in "ERC20AssetPool".
Recommendation Consider removing this check or moving it somewhere else.

Listing 121:

```
27 require(msg.value == 0, "no mainnet token allowed");
```

3.122 CVF-122

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IERC20Metadata.sol

Recommendation Consider moving this interface to the "interface" directory.

Listing 122:

```
5 interface IERC20Metadata is IERC20 {
```

3.123 CVF-123

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** ERC20AssetPool.sol

Description No access level specified for this variable.
Recommendation Consider explicitly specifying an access level.

Listing 123:

```
9 IERC20Metadata asset;
```

3.124 CVF-124

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20AssetPool.sol

Recommendation The argument type should be "IERC20".

Listing 124:

```
11 constructor(address _assetAddress) {
```

3.125 CVF-125

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20AssetPool.sol

Description String error messages are suboptimal.
Recommendation Consider using named errors instead.

Listing 125:

```
20 require(msg.value == bridgeFee, "bridge fee mismatch");
```

3.126 CVF-126

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** ERC20AssetPool.sol

Recommendation This value should be a named constant.

Listing 126:

```
37 return "erc20";
```

3.127 CVF-127

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Commitment.zok

Description Calling two width-4 Poseidons is suboptimal.
Recommendation Consider calling a single width-6 Poseidon which takes 5 inputs.
Client Comment This is an expected behaviour, because we need to use hashK to verify the commitment in the smart contract.

Listing 127:

```
 8 field hashK = poseidon([pubKey, randomP, randomR])
   return poseidon([hashK, amount, randomS])
```

3.128 CVF-128

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Rollup1.zok

Recommendation The value "20" should be a named constant.

Listing 128:

```
 8 private field[20] pathElements, \
10 assert(MerkleTreeBatchUpdater::<20, 20, 1>(\
```

3.129 CVF-129

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Sha256Batch.zok

Description This function is overcomplicated by handling both even and odd cases of N in the same code without if statements.
Recommendation Consider refactoring to handle either both cases in the same function with an if-statement or just using two functions.
Client Comment This function is not used any more.

Listing 129:

```
5 def main<N>(field[N] items) -> field:
```

3.130 CVF-130

- **Severity** Moderate
- **Status** Fixed
- **Category** Procedural
- **Source** KeccakBatch.zok

Description This library may have padding issues.
Recommendation Consider checking its code against Keccak test vectors.
Client Comment The padding issue has been fixed in Zokrates 0.7.14, we have upgraded our Zokrates to 0.7.14.

Listing 130:

```
1 import "hashes/keccak/256bit.zok" as keccak
```

3.131 CVF-131

- **Severity** Major
- **Status** Fixed
- **Category** Documentation
- **Source** MerkleTreeBatchUpdater.zok

Recommendation It should be commented how these numbers are obtained to guarantee they are not malicious

Listing 131:

```
6 450606924168002311076418960365866471059232703941254714774574507842
    ↪
4755206435,
```

3.132 CVF-132

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Transaction1x2.zok

Recommendation The value 20 should be a named constant.

Listing 132:

```
19 private field[1][20] inPathElements, \
20 private bool[1][20] inPathIndices, \

26 assert(JoinSplit::<20, 1, 2>(\
```

3.133 CVF-133

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Rollup4.zok

Recommendation The value 20 should be a named constant and the value 18 should be derived from that constant.

Listing 133:

```
8  private field[18] pathElements, \

10 assert(MerkleTreeBatchUpdater::<20, 18, 4>(\
```

3.134 CVF-134

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Rollup64.zok

Recommendation The value 20 should be a named constant and the value 14 should be derived from that constant.

Listing 134:

```
8  private field[14] pathElements, \

10 assert(MerkleTreeBatchUpdater::<20, 14, 64>(\
```

3.135 CVF-135

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Transaction2x0.zok

Recommendation The value 20 should be a named constant.

Listing 135:

```
19 private field[2][20] inPathElements, \
20 private bool[2][20] inPathIndices, \

26 assert(JoinSplit::<20, 2, 0>(\
```

3.136 CVF-136

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** MerkleTreeBuilder.zok

Description This allocates a two-dimensional array of (H + 1) * N elements, while only 2 * N - 1 elements are actually used.
Recommendation Consider allocating a linear array of 2 * N - 1 elements.

Listing 136:

```
5 field[H + 1][N] layers = [[0; N]; H + 1]
```

3.137 CVF-137

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Transaction1x1.zok

Recommendation The value 20 should be a named constant.

Listing 137:

```
19 private field[1][20] inPathElements, \
20 private bool[1][20] inPathIndices, \

26 assert(JoinSplit::<20, 1, 1>(\
```

3.138 CVF-138

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Transaction1x0.zok

Recommendation The value 20 should be a named constant.

Listing 138:

```
19 private field[1][20] inPathElements, \
20 private bool[1][20] inPathIndices, \

26 assert(JoinSplit::<20, 1, 0>(\
```

3.139 CVF-139

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Transaction2x2.zok

Recommendation The value 20 should be a named constant.

Listing 139:

```
19 private field[2][20] inPathElements, \
20 private bool[2][20] inPathIndices, \

26 assert(JoinSplit::<20, 2, 2>(\
```

3.140 CVF-140

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Transaction2x1.zok

Recommendation The value 20 should be a named constant.

Listing 140:

```
19 private field[2][20] inPathElements, \
20 private bool[2][20] inPathIndices, \

26 assert(JoinSplit::<20, 2, 1>(\
```

3.141 CVF-141

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Rollup256.zok

Recommendation The value 20 should be a named constant and the value 12 should be derived from that constant.

Listing 141:

```
8  private field[12] pathElements, \
10 assert(MerkleTreeBatchUpdater::<20, 12, 256>(\
```

3.142 CVF-142

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Rollup16.zok

Recommendation The value 20 should be a named constant and the value 16 should be derived from that constant.

Listing 142:

```
8  private field[16] pathElements, \
10 assert(MerkleTreeBatchUpdater::<20, 16, 16>(\
```

3.143 CVF-143

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

Description These structs are not used in the interface.
Recommendation Consider moving them to where they are used.
Client Comment Used in verifyTx.

Listing 143:

```
4  struct G1Point {
9  struct G2Point {
```

3.144 CVF-144

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

Description Names of structure fields usually start with a lower case letter.
Recommendation Consider renaming.

Listing 144:

```
5  uint256 X;
   uint256 Y;

10 uint256[2] X;
   uint256[2] Y;
```

3.145 CVF-145

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

Description The proof structure is specific to a particular ZK schema.
Recommendation Consider naming the structure after said schema.

Listing 145:

```
14 struct Proof {
```

3.146 CVF-146

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IHasher3.sol

Description While the interface name is quite generic, the function name is specific to the "poseidon" hash.
Recommendation Consider renaming to "hash3" or just "hash".

Listing 146:

```
4  function poseidon(uint256[3] memory data) external pure returns
   ↪ (uint256);
```