yAudit Rate Limiting Nullifier (RLN) Review

Review Resources:

- codebase
- spces v2

Auditors:

- 0xKitetsu
- 0xnagu
- Antonio Viggiano
- curiousapple
- Chen Wen Kang
- Elpacos
- gafram
- Igor Line
- IwItea
- MarkuSchick
- nullity
- 0ba
- parsley
- Rajesh Kanna
- Vincent Owen
- Vishal Kulkarni
- whoismatthewmc
- zkoranges

Table of Contents

Review Summary

Rate Limiting Nullifier

RLN (Rate-Limiting Nullifier) is a zk-library/protocol that enables spam prevention mechanism in permissionless environments while maintaining user anonymity.

Users register an `identityCommitment` which is stored in a Merkle tree, along with a stake. When interacting with the protocol, the user generates a zero-knowledge proof that (a) their identity commitment is part of the membership Merkle tree and (b) they have not exceeded a preset rate limit of messages. Each message is effectively an `x` point, and sending it reveals the correponding `y` that is the evaluation of a Shamir-secret-sharded polynomial. They key ingredient that RLN provides is that `m+1` points are needed to reconstructed the polynomial, where `m` is the rate limit. Hence, anyone can reconstruct a user's secret and withdraw (slash) their stake if they exceed the limit by just 1 message.

The circuits of RLN were reviewed between 31st May and 14th June, 2023. The repository was not under active development during the review and the review was limited to commit 37073131b9.

In addition, the contracts of the rln-contracts repo, commit dfcdb2beff, were provided as a reference implementation on how the circuits would be integrated (by a dApp for example). Although these contracts were out of scope, we have also included some related findings in this report.

Scope

The scope of the review consisted of the following circuits at the specific commit:

- `rln.circom`
- `utils.circom`
- `withdraw.circom`

After the findings were presented to the RLN team, fixes were made and included in several PRs.

This code review is for identifying potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, RLN and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	N/A	Any access control is left to the systems of dApps that integrate RLN
Mathematics	Good	No heavy mathematical components were involved. The mathematics of the underlying proof system, Groth16, were not reviewed. RLN uses the latest version of its implementation (the circom compiler).
Complexity	Good	The code is easy to understand and closely follows the specification
Libraries	Good	Well known libraries such as circomlib were used whenever possible
Decentralization	Good	RLN can be integrated in highly decentralizead and permissionless environments
Cryptography	Good	The docs recommend generating proofs with Groth16 using a BN254 curve, which has a security level of `128 bits`. It makes use of the Poseidon hash function known for its efficiency, zk-friendliness, and resistance against various cryptanalytic attacks. However, it's essential to note that cryptographic algorithms and functions are always subject to ongoing analysis, and new attacks or weaknesses may be discovered in the future.
Code stability	Good	The code was reviewed at a specific commit. The code did not changed during the review. Moreover, it is not likely to change significantly with the addition of features or updates
Documentation	Good	RLN documentation comprises a specification RFC, a Github documentation website, a Github README, and a Hackmd presentation. It is recommended to (a) add clear warnings to integrators of RLN regarding input assumptions (e.g. the need for hashing inputs, and the danger of malicious frontends that may intentionally not do that) and (b) reduce the number of resources necessaries to fully understand the protocol
Monitoring	N/A	The protocol is intended to be integrated by other systems or dApps which will be responsible for the monitoring
Testing and verification	Average	The protocol contains only a few tests for the circuits. It is recommended to add more tests to increase the test coverage

Findings Explanation

Findings are broken down into sections by their respective Impact:

- Critical, High, Medium, Low Impact
    - These are findings that range from attacks that may cause loss of funds, a break in the soundness, zero-knowledge, completeness of the system, proof malleability, or cause any unintended consequences/actions that are outside the scope of the requirements
- Informational
    - Findings including Recommendations and best practices

Critical Findings

1. Critical - Risk of secret getting revealed if the input is zero

When the message `x` is given the value 0, the identity secret is revealed.

Technical Details

If the input x is 0 in `y <== identitySecret + a1 * x;`, it will reveal the `identitySecret`. `x` becomes 0 for value 0 and `21888242871839275222246405745257275088548364400416034343698204186575808495617` as mentioned in [Circom Docs]

Also, Circom 2.0.6 introduces two new prime numbers to work with :

- The order of the scalar field of `BLS12 - 381` is `52435875175126190479447740508185965837690552500527637822603658699938581184513`
- The goldilocks prime `18446744069414584321`, originally used in `plonky-2`

```
template RLN() {
    ...
    signal output y;
    ...
    y <== identitySecret + a1 * x;
    ...
}
```

Impact

The user may get their secret revealed without them violating the rate-limit rules.

Recommendation

1. Hash the random number instead of directly using it.
2. Have a `greater_than` constraint that checks whether `x` is greater than zero (quite inefficient compared to 1).
3. Impose `greater_than` constraint / hash the random number outside of circuit (relatively more efficient).
4. Constrain `a1*x` to be non-zero by adding `isZero(a1*x).out === 0`

Developer Response

> *"x is hash of the message, and it's hashed outside of the circuits. Probability of finding preimage of 0 for Poseidon hash is negligible, it's not critical bug; though it may be user error to use 0 value for x, but as it's public input - then it can be checked on the client side - no deal to make it in the circuit."*

Reported by Rajesh Kanna, OxKitetsu, Chen Wen Kang, Vincent Owen, Antonio Viggiano, Igor Line, Oba, Oxnagu

High Findings

None

Medium Findings

1. Medium - Unused public inputs optimized out by the circom compiler

The circuit `withdraw.circom` contains a potential issue related to public inputs being optimized out by the circom compiler.

Technical Details

The `withdraw.circom` circuit specifies `address` as a public input but does not use this input in any constraints. Consequently, the unused input could be pruned by the compiler,

making the zk-SNARK proof independent of this input. This may result in a potentially exploitative behavior.

Impact

Medium. Despite circom not generating constraints for unused inputs, snarkjs (which RLN uses) does generate these constraints. The protocol also tested `ark-circom`, which also adds the constraints ommitted by circom. However, if some zk-SNARK implementation does not include these constraints, it can lead to a potential loss of funds by users of the protocol.

Recommendation

As outlined in OxPARC's zk-bug-tracker, it is advised to add a dummy constraint using the unused signal address in the circuit. This change ensures that the zk-SNARK proof is dependent on the `address` input.

```
diff --git a/circuits/withdraw.circom b/circuits/withdraw.circom
index a2051ea..7a4b88d 100644
--- a/circuits/withdraw.circom
+++ b/circuits/withdraw.circom
@@ -6,6 +6,8 @@ template Withdraw() {
     signal input identitySecret;
     signal input address;

+    signal addressSquared <== address * address;
+
     signal output identityCommitment <== Poseidon(1)([identitySecret]);
 }
```

Developer Response

> *"Good find! We'll add the "dummy constraint" to the circuit."*

Reported by Antonio Viggiano, Igor Line, Oba, Chen Wen Kang, Vincent Owen, OxKitetsu, MarkuSchick, Elpacos, gafram, Rajesh Kanna, curiousapple, Oxnagu, lwltea, parsley, Vishal Kulkarni, whoismatthewmc, zkoranges

2. Medium - Effective stake at risk is only fees and not the complete stake as intended

RLN contracts require users to stake and register themselves before being part of the network.
The idea here is if a user misbehaves and spams the network, anyone can slash their stake and get some of it.
However, since networks are anonymous, users can slash themselves using another identity and only lose fees.

Technical Details

`N/A`

Impact

Users lose only part of the stake for their misbehavior

Recommendation

Consider using different incentive mechanism for slash.

Developer Response

> *"Acknowledged"*

Reported by curiousapple

Low Findings

1. Low - Inconsistency between contract and circuit on the number of bits for userMessageLimit

There is an inconsistency between the `RLN.sol` and `rln.circom` pertaining to the `userMessageLimit`. Specifically, the difference lies in the range of values that `messageLimit`, `messageId`, and `userMessageLimit` can take.

Technical Details

In `RLN`.sol, the calculation of `messageLimit` is based on the amount divided by `MINIMAL_DEPOSIT`. Given the current implementation, `messageLimit` has the potential to accept values up to `2**256 - 1`.

```
uint256 messageLimit = amount / MINIMAL_DEPOSIT;
```

On the other hand, in `rln.circom`, the `messageId` and `userMessageLimit` values are limited by the `RangeCheck(LIMIT_BIT_SIZE)` function to `2**16 - 1`.

```
template RLN(DEPTH, LIMIT_BIT_SIZE) {
...
    // messageId range check
    RangeCheck(LIMIT_BIT_SIZE)(messageId, userMessageLimit);
...
}
component main { public [x, externalNullifier] } = RLN(20, 16);
```

Although the contracts check that the identity commitment is lower than the size of the set, which is `2**20`, it is possible that the `DEPTH` and `LIMIT_BIT_SIZE` parameters of the circuit are modified, which would lead to unexpected outcomes if not addressed.

Impact

Low. On the current implementation, this discrepancy does not pose any issues to the protocol.

Recommendation

Short term, add a range check to the circuits to make sure they are constrained to the same range as the smart contract variables. Long term, make sure the input space of the contracts and the circuits are always in sync.

```
function register(uint256 identityCommitment, uint256 amount) external {
        ...
        uint256 messageLimit = amount / MINIMAL_DEPOSIT;
+       require( messageLimit <= type(uint16).max , "Max amount of message limit
is 65535");
        token.safeTransferFrom(msg.sender, address(this), amount);
        ...
    }
```

Developer Response

> *“Good find! We’ll add this check to the contract :)”*

Reported by nullity, Antonio Viggiano, Igor Line, Oba, curiousapple, Oxnagu, Vishal Kulkarni

2. Low - Difference between documents and implementation

The documentation mentions that the rate limit is between `1` and `userMessageLimit`

> *"Signaling will use other circuit, where your limit is private input, and the counter `k` is checked that it's in the range from 1 to userMessageLimit."*

Although the implementation is sound, the counter `k` should be documented as being between `0` to `userMessageLimit-1`
Testing the circuit allows for a `messageId` of `0` and does not allow for a `messageId` of `userMessageLimit`.

Technical Details

```
signal rangeCheck <== LessThan(LIMIT_BIT_SIZE)([messageId, limit]);
```

The above code will always return true for a `messageId` of `0`

Impact

Low. There is no impact except for clarity to potential developers/users.

Recommendation

The project may consider chaging the wording to be something like :

```
Signaling will use other circuit, where your limit is private input, and the
counter `k` is checked that it is within the range from 0 to userMessageLimit -1.
```

Developer Response

> *"Docs are out of date, we'll update them soon."*

Reported by lwltea, parsley, Vishal Kulkarni, whoismatthewmc, zkoranges

3. Low - Unnecessary import & inheritance of Ownable

Contract RLN inherits from Ownable but ownable functionality isn't actually used by the contract.

Technical Details

There are imports and inheritance for `Ownable` in the `RLN.sol` contract. Ownable is never used within the RLN contract.

Impact

Low.

Recommendation

Remove `Ownable` import and inheritance from `RLN.sol`

Developer Response

> *"Thanks, we should remove Ownable from the contract."*

Reported by curiousapple, lwltea

4. Low - Edge case in user registration

In the user registration the calculation `y = mx + c` can have unexpected side effects.

Technical Details

In the user registration spec, for `y = mx + c` where `m` and `x` can be multiplicative inverse of each other(less probability), then the equation becomes `y = 1 + c`. The attacker can easily subtract from the public `y` to get the hash of the secret key.

Impact

Low.

Recommendation

In `rln.circom`, add a check to make sure `m` and `x` are not multiplicative inverses of each other.

Developer Response

> *"The probability of that is negligible, and prover can make this check before proving."*

Reported by Vishal Kulkarni

5. Low - Parameter validation missing in rln.circom

The code doesn't check if `DEPTH` and `LIMIT_BIT_SIZE` are within expected ranges or conform to specific requirements. If these are passed in as invalid values, it could lead to unexpected behavior or vulnerabilities.

Technical Details

```
template RLN(DEPTH, LIMIT_BIT_SIZE) {
```

Missing range checks for `DEPTH` and `LIMIT_BIT_SIZE`

Impact

Low.

Recommendation

Add checks to ensure that the values of `DEPTH` and `LIMIT_BIT_SIZE` are within expected ranges.

Developer Response

> *"`DEPTH` and `LIMIT_BIT_SIZE` are not the inputs of the circuit, but compile-time metaparameters. We don't need to range check them. It's just constant values that describe behavior of the circuit as well as code itself."*

Reported by 0xnagu, zkoranges

6. Low - Check if the `identityCommitment` is less than the SCALAR FIELD

As mentioned in 0xparc's bug tracker, the inputs have to be less than the Snark's scalar field.

Technical Details

The prime field in circom is ~ `2**254` whereas solidity supports 256-bit integers. There is a possibility of users registering twice, once using the `identityCommitment` & another time

using `A` such that `A mod p = identityCommitment` where `p` is the prime field used in circom. Hence, the user registers twice using the same `identitySecret`.

Impact

Low.

Recommendation

Add check to ensure that user doesn't register twice with the same secret.

```
require(identityCommitment < snark_scalar_field);
```

Developer Response

> *"`identityCommitment` cannot be 256 bit, as it's the result of Poseidon hash-function that's defined over p (prime field of Circom/bn254)"*

Reported by nullity

7. Low - Specification uses incorrect definition of identity commitment

The V2 Specification uses the `identity_secret` to compute the `identity_commitment` instead of the `identity_secret_hash`. The `identity_secret` is already used by the Semaphore circuits and should not get revealed in a Slashing event.

Technical Details

RLN stays compatible with Semaphore circuits by deriving the secret ("`identity_secret_hash`") as the hash of the semaphore secrets `identity_nullifier` and `identity_trapdoor`.

RLN V2 improves upon the V1 Protocol by allowing the setting of different message rate-limit.
Hence, the definition of the user identity changes from the V1's definition:

```
    identity_secret: [identity_nullifier, identity_trapdoor],
    identity_secret_hash: poseidonHash(identity_secret),
    identity_commitment: poseidonHash([identity_secret_hash])
+   rate_commitment: poseidonHash([identity_commitment, userMessageLimit])
```

The RLN-Diff flow wrongfully derives the `identity_commitment` from the `identity_secret` directly instead of the `identity_secret_hash`.

Impact

Low.

Recommendation

In the short term, modify the following part of the V2 Specification :

```
Registration

-id_commitment in 32/RLN-V1 is equal to poseidonHash=(identity_secret).
+id_commitment in 32/RLN-V1 is equal to poseidonHash=(identity_secret_hash).
The goal of RLN-Diff is to set different rate-limits for different users. It
follows that id_commitment must somehow depend on the user_message_limit
parameter, where 0 <= user_message_limit <= message_limit. There are few ways to
do that:
1. Sending identity_secret_hash = poseidonHash(identity_secret, userMessageLimit)
and zk proof that user_message_limit is valid (is in the right range). This
approach requires zkSNARK verification, which is an expensive operation on the
blockchain.
-2. Sending the same identity_secret_hash as in 32/RLN-V1
(poseidonHash(identity_secret))
+2. Sending the same identity_commitment as in 32/RLN-V1
(poseidonHash(identity_secret_hash))
and a user_message_limit publicly to a server or smart-contract where
-rate_commitment = poseidonHash(identity_secret_hash, userMessageLimit) is
calculated.
+rate_commitment = poseidonHash(identity_commitment, userMessageLimit) is
calculated.
The leaves in the membership Merkle tree would be the rate_commitments of the
users. This approach requires additional hashing in the Circuit, but it
eliminates the need for zk proof verification for the registration.
```

In the long term, rename the variable `identity_secret` in the circuit to avoid further confusion with a variable of the same name derived from Semaphore.

Developer Response

> *"Spec/docs is out of date, we'll update it soon. Thanks for your review."*

Reported by MarkuSchick

Informational Findings

1. Informational - Mismatch between specification and implementation for x value

Mismatch between specification and implementation regarding the `x` message value.

Technical Details

In 58/RLN-V2, the specification states that `x` is the signal hashed. However, in the `rln.circom` circuit, `x` is the message without hash.

Impact

Informational.

Recommendation

Update the implementation to hash the `x` value according to the specification.

Developer Response

> *"x is hash of the message, and it's hashed outside of the circuits. Correctness can be checked on the client-side as it's public input."*

Reported by Antonio Viggiano, Igor Line, Oba

2. Informational - Misleading Naming Convention for Utility Circuits

The `withdraw.circom` does not involve the generation of a proof to withdraw funds from the contract. Instead, it is a utility circuit that is used to generate the identity commitment from a private identity hash. The naming of the circuit is misleading.

Technical Details

`N/A`

Impact

Informational.

Recommendation

Rename the circuit to `identity_commitment.circom` to better reflect its purpose.

Developer Response

> *"None"*

Reported by Chen Wen Kang, Vincent Owen

3. Informational - Edge case in Shamir Secret Sharing computation

In Shamir Secret Sharing computation, the coefficient of the random value `x` may be a multiplicative inverse and the secret maybe revealed. However, the probability of having multiplicative inverse is $1/p$ where p is prime order, which is negligible.

Technical Details

`N/A`

Impact

The user may get his secret revealed but the probability is negligible.

Recommendation

This finding is just to highlight the potential of issue. If the secret sharing is computed using higher degree polynomials, this is not negligible.

Developer Response

> *"As you said - probability of that is negligible"*

Reported by Rajesh Kanna

4. Informational - Possibility of Spamming

By sending **the same** message continuously, users can still spam the network. It is true that if the users change their msg and go above the msg limit, they expose their secret, however, they can still send the same messages continuously.

Technical Details

`N/A`

Impact

Very low. Nodes will filter out repeated messages and drop the peer, as is the standard practice in any p2p messaging system. Such feature is baked into p2p libraries.

Recommendation

`N/A`

Developer Response

> *"You cannot spam more than once. As soon as user exceed the limit, they can be slashed immediately (at least on the node level)"*

Reported by curiousapple

5. Informational - Penalty enforced on the user doesn't scale with the magnitude of their spam

There is no difference between a penalty for spamming once or spamming x1000 times. Hence the cost of attack does not scale with the impact
All attacker needs to do is do a minimum deposit and then he/she can spam the network with any number of messages at once, and all they are penalized for is a minimum deposit.

Technical Details

`N/A`

Impact

This incentive structure doesn't take into account the degree of spamming.

Recommendation

`N/A`

Developer Response

> *"You cannot spam more than once. As soon as user exceed the limit, they can be slashed immediately (at least on the node level)"*

Reported by curiousapple

Final remarks

- The RLN Protocol assumes that :
  - Poseidon hash fucntion is collision-resistant, resistant to differential, algebraic, and interpolation attacks.
  - The trusted setup to instantiate the Groth16 parameters will be done correctly and all relevant artifacts kept safe.
- The Merkle tree used for membership proof is assumed to be secure against second-preimage attacks.
- Social engineering attacks are still a valid way to break the system. An attacker can obtain the `identitySecret` signal of a user by using methods such as social engineering, phishing attacks, or exploiting vulnerabilities in the user's system.
- The security of the circuit depends on the secrecy of the `identitySecret` signal, which is assumed to be kept secret by the user.
- The security of the circuit depends on the security of the cryptographic primitives used for range checks and SSS share calculations.
- Overall, the code demonstrates good implementation of mathematical operations and basic functionality. However, it could benefit from more extensive documentation and additional testing.

Automated program analysis tools

Over the course of the audit, in addition to a manual review of the code, we applied different automated analysis tools and evaluated their output.

1. circomspect
2. Ecne

Note:: Ecne can output false positive. It relies on static analysis but does not call an SMT solver to concretely find the potential attack vector.

Results

The relevant findings from each tool were already included in the report.

Below is the full output of each tool.

1. Circomspect

[circomspect : A static analyzer and linter for the Circom zero-knowledge DSL]

```
circomspect: analyzing template 'RLN'
note: Field element arithmetic could overflow, which may produce unexpected
results.
   ┌─ /home/testadmin/Desktop/zk/rln/circom-rln/circuits/rln.circom:34:11
   │
34 │     y <== identitySecret + a1 * x;
   │           ^^^^^^^^^^^^^^^^^^^^^^ Field element arithmetic here.
   │
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#field
-element-arithmetic.

circomspect: 1 issue found.

circomspect: analyzing template 'RangeCheck'
note: Comparisons with field elements greater than `p/2` may produce unexpected
results.
   ┌─ /home/testadmin/Desktop/zk/rln/circom-rln/circuits/utils.circom:37:12
   │
37 │     assert(LIMIT_BIT_SIZE < 253);
   │            ^^^^^^^^^^^^^^^^^^^ Field element comparison here.
   │
   = Field elements are always normalized to the interval `(-p/2, p/2]` before
they are compared.
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#field
-element-comparison.

warning: Intermediate signals should typically occur in at least two separate
constraints.
   ┌─ /home/testadmin/Desktop/zk/rln/circom-rln/circuits/utils.circom:42:5
   │
```

```
42 |     signal bitCheck[LIMIT_BIT_SIZE] <== Num2Bits(LIMIT_BIT_SIZE)(messageId);
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
   |     |
   |     The intermediate signal array `bitCheck` is declared here.
   |     The intermediate signals in `bitCheck` are constrained here.
   |
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#under
-constrained-signal.

warning: Using `Num2Bits` to convert field elements to bits may lead to aliasing
issues.
   ┌─ /home/testadmin/Desktop/zk/rln/circom-rln/circuits/utils.circom:42:41
   │
42 |     signal bitCheck[LIMIT_BIT_SIZE] <== Num2Bits(LIMIT_BIT_SIZE)(messageId);
   |                                         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Circomlib template `Num2Bits` instantiated here.
   │
   = Consider using `Num2Bits_strict` if the input size may be >= than the prime
size.
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#non-
strict-binary-conversion.

circomspect: analyzing template 'MerkleTreeInclusionProof'
circomspect: 3 issues found.

circomspect: analyzing template 'Withdraw'
warning: The signal `address` is not used by the template.
  ┌─ /home/testadmin/Desktop/zk/rln/circom-rln/circuits/withdraw.circom:7:5
  │
7 |     signal input address;
  |     ^^^^^^^^^^^^^^^^^^^ This signal is unused and could be removed.
  │
  = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unuse
d-variable-or-parameter.

circomspect: 1 issue found.
~~~
```

2. Ecne

The Circom circuits were compiled to non-optimized R1CS constraints system and then they were tested for `Weak Verification` to check for any bad constraints or underconstraints. All the circuits passed the Ecne tests without any mis- or under-constraints. This verifies that R1CS equations of the given circuits uniquely produce outputs given specific inputs.