Privacy Pools Security Audit

Report Version 1.0

April 30, 2024

Conducted by:

Dimo Dimov, ZK Engineer
George Hunter, Independent Security Researcher

Table of Contents

1 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

2 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

2.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

2.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

3 Executive summary

Overview

Project Name	Privacy Pools (fork)
Repository	https://github.com/0x132fabb5bc2fb61fc68bcfb5508841ddb11e9/pools-sol
Commit hash	440794ab1c0d738b9e88618150958075883e3378
Resolution	b2d1ca6c92f6a0ca504e265429e9847718e62615
Methods	Manual review, Static Analysis

contracts/IncrementalMerkleTree.sol
contracts/PrivacyPool.sol
contracts/PrivacyPoolFactory.sol
contracts/verifiers/ProofLib.sol
circuits/*

Issues Found

High risk	0
Medium risk	0
Low risk	1
Informational	1

4 Findings

4.1 Low

4.1.1 ExclusionProof missing bit length check for LessThan

Severity: *Low*

Context: exclusion_proof.circom#L39

Description: The idea of the *exclusion_proof.circom* template is that you will be given the root of a sorted merkle tree and a proof of inclusion of *lowerLeaf* and *upperLeaf* in the tree, which must be adjacent leaves in the tree. Therefore, by proving that *lowerLeaf* < *leaf* < *upperLeaf* you would prove that *leaf* is not part of the tree.

The *LessThan* template compares two inputs and outputs *1* if the first input is less than the second input. The problem is that it takes in the maximum number of bits for both inputs as a parameter, but does not actually check this constraint.

And leaves being just results of Poseidon hashes could be numbers larger than 252 bits, which could lead to a problem where an attacker can generate a proof for exclusion for a leaf that is actually part of the tree. Or where an honest prover cannot prove his leaf is part of the tree even if it actually is not.

Recommendation: Mitigating this issue would require working with commitments trimmed to the 252nd bit in the entire protocol as well as adding the constraint for the leaves' size in the template:

```
component n2b1 = Num2Bits(252);
n2b1.in <== lowerLeaf;

component n2b2 = Num2Bits(252);
n2b2.in <== leaf;

component n2b2 = Num2Bits(252);
n2b2.in <== upperLeaf;
```

Resolution: Resolved. Developer Response: This code section was deemed out of scope as the implementation utilizing this logic was found to be unnecessary. Subsequently, it has been removed from our project. This action ensures that the potential vulnerability does not affect the security of our protocol.

4.2 Informational

4.2.1 Optimizations, code-style suggestions and non-critical issues

Severity: *Informational*

Context: contracts, circuits

Description: The contracts contain one or more optimizations, code-style suggestions and non-critical issues. In an effort to keep the report size reasonable, we enumerate these below:

1. The *poseidon* storage variable in PrivacyPoolFactory can be marked as _immutable.
2. Consider using the more recent compiler version to utilize the latest optimizations and bug fixes.

3. There's no need to invoke the *ReentrancyGuard()* constructor in PrivacyPool on line 80.
4. Consider saving gas by moving the assets transfer from *_deposit* to *deposit* and *depositMany* since *depositMany* only needs to execute the transfer once for all separate deposits, rather than calling *.safeTransferFrom* for each commitment.
5. It should be noted that fee-on-transfer and rebasing tokens are not supported by the pools.
6. Consider implementing a check in *_verifyWithdrawal* that *proof.refund* is not greater than 0 if the asset is the native token.

Recommendation: Consider resolving the above suggestions.

Resolution: Resolved. The above recommendations have been successfully implemented.