Privacy Pools — the open source code security analysis

by Pessimistic

This report is public

May 15, 2024

Abstract

In this report, we consider the security of smart contracts of Privacy Pools - a contribution to open source. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Privacy Pools smart contracts. We described the audit process in the section below.

The audit showed one issue of medium severity: Discrepancy with documentation. Also, one low-severity issue was found, and several notes were added at the last section of the report.

After the initial audit, the codebase was updated. The project description became public and were updated - in that way, the Discrepancy with documentation issue of medium severity was fixed. The contributors fixed or commented all low severity issues and notes.

The number of tests increased. The overall quality of the project is good.

General recommendations

We have no futher recommendations.

Project overview

Project description

For the audit, we were provided with Privacy Pools fork project on a public GitHub repository, commit 9761382c982dc27f8cb4ff2a404bf1e4b95b7530.

The scope of the audit included:

- **contracts/PrivacyPool.sol**;
- **contracts/PricavyPoolFactory.sol**;
- **contracts/IncrementalMerkleTree.sol**;
- **circuits/withdraw_from_subset.circom**.

The documentation for the project included **FAQ.md** (sha1sum 48de0fe6f141e7323f7a8a437ceb9967784a58f0).

All 62 tests pass successfully. The code coverage is 100%.

The total LOC of audited sources is 781.

Codebase update #1

After the initial audit, the codebase was updated, and we were provided with commit b2d1ca6c92f6a0ca504e265429e9847718e62615.

The contributors fixed the medium severity issue and fixed or provided the comments for all low severity issues and notes. They also updated and made the **FAQ.md** file public by adding to the repository.

The number of tests increased to 64. All of them passed. The code coverage was 100%.

Audit process

We started the audit on April 1 and finished on April 12, 2024. The scope was limited. We did not check the ZKP math scheme inside this protocol and the **Poseidon** hash functionality. Our task was to check the solidity part, which is specified in the project description. We also looked at:

- The circom file, but our review here was limited as we mainly focused on the Solidity part of the codebase;
- The **ProofLib** contract to better understand the project.

We inspected the materials provided for the audit. Then, we contacted the repository contributor for an introduction to the project. After the discussion, we did some preliminary research and identified parts of the code and logic that needed additional attention during the audit. As a result of our questions, the contributors wrote and provided the detailed description of the project.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among others, we verified the following properties of the contracts:

- Whether the solidity and circom code corresponds to the provided description;
- Whether secrets are not revealed when dealing with contracts;
- Whether double-spending is not possible;
- Whether the precalculated hashes are correct in the Incremental Merkle Tree;
- Whether the path for withdrawal and subset proofs are the same.
- Whether the usage of the same commitment hash for different deposits is safe;

During the work, we stayed in touch with the repository contributor of the team and discussed confusing or suspicious parts of the code.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the repository contributor of the team in the text file.

We ran tests and calculated the code coverage.

During the audit, we did not find any major issues within the scope. However, we have added a few notes that should alert users and other projects on what to look for when integrating.

Important: we only reviewed a portion of solidity and one circom file that is used as a source to generate a verifier's code and proofs of deposits. In the private report, we have combined all verified issues found during manual auditing or detected by automated tools.

We made the rechek on April 23-27, 2024. We checked the contributors comments and the code updates. Also, we reran the tests and recalculated the code coverage.

Finally, we updated the report, including the N01 description.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Discrepancy with documentation (fixed)

Documentation is a critical part that helps to improve security and reduce risks. It should explicitly explain the purpose and behavior of the contracts, their interactions, and key design choices. During the audit, we found several discrepancies between the code and the provided documentation:

- The `Factory` section of **FAQ** mentioned that `The factory enables the creation of new privacy pool instances for a specified asset with a given power base 10, or a number of tokens`. However, it seems that it is not possible to specify any number of tokens in the current version (only powers of base 10);
- In the `Withdraw` section, there is a paragraph: `Fee: The fee works by providing an amount less than the amount being withdrawn, that can be paid to a fee recipient`; but in the code, the fee is less or equal, so it is possible to fully send the deposit to the relayer;
- In the `Withdrawal Process and On-Chain Verification` part, there is a statement, `Their withdrawal is part of an approved deposit subset, excluding known illicit funds`; but in fact, it is possible to generate a proof with any subset where your deposit is allowed (including any number of illegal deposits);
- In the `On-Chain Verification`, it is said that `It checks that the subset root provided by the user matches one of the approved subset roots, representing clean subsets free of illicit funds`. Still, there is no verification on-chain that the given `subsetRoot` does not contain illicit deposits. As a result, anyone can withdraw their deposit (but not everyone can submit subset proof without illicit deposits).

The contributors updated the description and made it public.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Duplicate getter (fixed)

`PoolGroups` storage variable is marked public, but there is hand-written `poolGroupByInput` getter. Consider changing the visibility of the variable from `public` to `private` or `internal` at line 24 in the **PrivacyPoolFactory** contract.

The `poolGroupByInput` getter has been removed from the current version of the code.

Notes

N01. Importance of preserving secrets save (commented)

It is crucial not to reveal a `secret`, as this can lead to loss of deposit.

It is also essential that the withdrawer does not disclose the relationship to `leafIndex` or `commitmentHash` (these parameters are stored in the Merkle tree, and anyone can see them) so as not to lose privacy when withdrawing.

> *Comment from the contributors: Acknowledged.*

N02. Poseidon hash implementations must be agreed (commented)

The protocol uses the Poseidon hash function to calculate different hashes, including nodes of Incremental Merkle trees on-chain. This function's address is set in the factory as `immutable` for all pools on-chain. The on-chain implementation must match the one used in Circom during the proof generation. Otherwise, the on-chain verifier would not accept generated proofs.

> *Comment from the contributors: Acknowledged. Consistency in the implementation of the Poseidon hash function across the protocol is essential to ensure the validity of the proofs generated.*

N03. Setup ceremony importance (commented)

The Powers of Tau setup ceremony plays a vital role in the protocol. The proper execution of the setup ceremony ensures the robustness and trustworthiness of the system. Generating input parameters for the circuit correctly is crucial to prevent the possibility of generating malicious ZK proofs later in the project.

> *Comment from the contributors: The importance of the Powers of Tau setup ceremony and its role in maintaining the protocol's security and trustworthiness is acknowledged. Whoever deploys the contracts must follow the best practices in conducting the setup ceremony to ensure the highest security standards.*

N04. The same verifier is used for multiple contracts (commented)

The code of **WithdrawFromSubsetVerifier** contract is reused for every deployed **PrivacyPool**. It would be more efficient to deploy the verifier separately and make calls to the single contract from all the pools, as the cost of pool deployments would be reduced.

> *Comment from the contributors: While deploying the verifier as its own contract could reduce deployment costs for pool instances, the high transaction volume in each pool (approximately 2^20 deposits and withdrawals) negates these savings. The current approach prioritizes minimizing transaction costs for withdrawals, which seems more beneficial in the long run.*

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Daria Korepanova, Senior Security Engineer
Evgeny Marchenko, Senior Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

May 15, 2024