Semaphore and Mixer Smart Contracts. Audit Summary

Mikhail Vladimirov and Dmitry Khovratovich

8th March 2020

This document is the audit of Semaphore and Mixer smart contract set performed by ABDK Consulting.

1. Introduction

We've been asked to review the Semaphore codebase in GitHub repos (Semaphore, Mixer) in September 2019. We reviewed selected Solidity contracts; the other ones are taken from OpenZeppelin codebase (and are deemed secure), are used for testing only, or are outdated.

After a few interactions with the authors, all major and critical issues were resolved.

2. Phase One

In the first phase we have identified a number of issues in all contracts including Verifier.sol (an autogenerated contract for SNARK proof verification). We recommended substantial modification to all contracts.

In **MerkleTreeLib.sol** the critical issues were

- Only first 256 trees could be accessed.
- Leaf insertion method overwrote old leafs.
- Some range checks for inputs and paths were missing.

In **Semaphore.sol** the critical issues were

- Arbitrary value could be used as a `zero` in the pre-filled Merkle tree, for example a valid commitment, which could lead to money loss.
- The identity update procedure could be screwed if a wrong path is passed.
- Range checks for user inputs were missing.

In **Mixer.sol** the critical issues were

- It was not properly tied to a Semaphore instance so that if another contract had access to the instance, not even in parallel, the Mixer user could lose deposits.
- Range checks for user inputs were missing.

Also contract design was suboptimal: a lot of code duplication for ETH and Token cases. We recommended separating the common code to an abstract contract.

Verifier.sol, an autogenerated contract for ZK SNARKs, had a malleability bug so that multiple valid proofs can be created from a single one.

3. Phase Two

All major and critical issues from Phase One were fixed by December 2019. We made a review of the new code (it was in the private repos) and identified the following new critical issues:

- In **MerkleTreeLib.sol** the leaf counter was not always updated when needed, causing data loss. Also some range checks in public functions were missing.
- Semaphore and Mixer contract are still critically dependent on each other so we strongly recommended atomic deployment and guarantee that Mixer is the only owner of the Semaphore instance, otherwise data and money loss were possible.
- In **Mixer.sol**, a proof could be crafted with another nullifier so that deposits could be withdrawn twice.

4. Phase Three

All critical issues from Phase Two were fixed by February 2020. Also Mixer.sol was finally divided into abstract, ETH-specific, and ERC20-specific code. We made a review of the changes and new code (it was in private repos), and new critical issues were identified:

- In **IncrementalMerkleTree.sol** an overflow was possible when depth is 32 because certain variables were 32-bit. In the same method the always zero value is returned as a leaf index.

All these were fixed.

5. Remaining issues

We have identified the following non-critical issues that have not been fixed. They do not affect security.

- The `MIMCSponge` function's analogue in Circom code is called `MIMCFeistel`.

- The `preBroadcastCheck`() does not make exactly the same checks as `isValidSignalAndProof`() but there is still duplicated code.
- The proof data type can be made `bytes[]` to make API more portable.
- The `signal` body is not always logged, so clients should store some information internally.
- A malicious user can trick a relayer to broadcast a valid proof which results in sending Ether to a contract that does not accept it, so that the gas spent would be lost. This might not be detected on a trial execution. Relayers should be careful with recipients that are contracts.
- `getIdentityCommitments` method might be expensive.
- The code may work incorrectly with tokens that charge transaction fees also in tokens, so that after `transfer`() fewer tokens are added to the balance than requested.