# Tornado Cash Anonymity Mining Audit

Igor Gulamov, 15 October 2020

## Introduction

ZeroPool conducted the audit of Tornado.cash anonymity mining smart contracts and zkSNARK circuits.

This review was performed by an independent reviewer under fixed rate.

## Scope

Source code is stored in https://github.com/tornadocash/tornado-anonymity-mining. Most issues are corresponding to **820bd83** commit.

circuits excluding MerkleTree.circom,
contracts excluding IVerifier.sol, RewardVerifier.sol, TreeUpdateVerifier.sol, WithdrawVerifier.sol.

## Issues

We found no critical issues. Major issues are fixed or do not provide vulnerabilities in the case if the contract is used with the governance.

### Major Miner.sol#L296-L309

We recommend you avoid multiple SLOADs as you can. SLOAD is expensive and in the future it will not become cheaper.

**Comment**

Fixed in https://github.com/tornadocash/tornado-anonymity-mining/commit/e0008b3ed46dbf127b452d1d086235f0fe2dfcb8.

### Major RewardSwap.sol#L88

```
function setPoolWeight(uint256 newWeight) external onlyMiner {
   poolWeight = newWeight;
}
```

`poolWeight` manipulation should be limited. Otherwise, the operator manages TORN distribution over a wide range.

**Comment**

Won’t fix. This constant can only be changed by governance, and it’s hard to determine correct limits for it in advance.

## Medium MerkleTreeUpdater.circom#L9

```
template MerkleTreeUpdater(n, zeroLeaf) {
    signal input oldRoot;
    signal input newRoot;
    signal input leaf;
    signal input pathIndices;
    signal private input pathElements[n];

    component treeBefore = MerkleTree(n);
    for(var i = 0; i < n; i++) {
        treeBefore.pathElements[i] <== pathElements[i];
    }
    treeBefore.pathIndices <== pathIndices;
    treeBefore.leaf <== zeroLeaf;
    treeBefore.root === oldRoot;

    component treeAfter = MerkleTree(n);
    for(var i = 0; i < n; i++) {
        treeAfter.pathElements[i] <== pathElements[i];
    }
    treeAfter.pathIndices <== pathIndices;
    treeAfter.leaf <== leaf;
    treeAfter.root === newRoot;
}
```

Unoptimized code. Numeric `pathIndexes` are bitified twice: at each `MerkleTree` component. It is better to bitify it once.

**Comment**

Fixed in https://github.com/tornadocash/tornado-anonymity-mining/commit/7487ac8b09dcfc78ecc166cff5208435010cec8e.

## Minor Withdraw.circom#L8-L9

Gas cost for withdrawals will be reduced in the case if the sum of the `amount` and `fee` is stored in one signal and the `fee` or `amount` is stored in `extData`.

**Comment**

Fixed in https://github.com/tornadocash/tornado-anonymity-mining/commit/459fa79321b1b48eefc3cb85f82733528d5e56d3.

## Minor Miner.sol#L226-L232

Percent fee makes calculations more transparent and reduces gas costs.

Both ways are the same in terms of front-run risk distribution (the exponential decay multiplier is used for relayer's and user's rewards).

**Comment**

Won't fix. We have a flat fee in reward method anyway, so it's more consistent to do it the same way in withdraw.

## Minor Miner.sol#L110

We recommend you use a fixed-sized array for verifiers.

**Comment**

Fixed in https://github.com/tornadocash/tornado-anonymity-mining/commit/49ce4e9375509c3bc866d32fbf6345361a85b9a0.

## Minor Miner.sol#L289

`Keccak252` should be internal for better optimizations.

**Comment**

fixed in https://github.com/tornadocash/tornado-anonymity-mining/commit/c8865315c50f3a0cabdd4110a6c45ceba4d4b809

## Minor Miner.sol#L42

```
struct TreeLeaf {
   address instance;
   bytes32 hash;
   uint256 block;
   uint256 index;
 }
```

Keyword `block` is reserved for `block` object. We recommend you to rename the field of the struct.

**Comment**

Won’t fix.

## Minor Reward.circom#L74-L77

Component

```
component checkRoot = ForceEqualIfEnabled();
checkRoot.in[0] <== inputRoot;
checkRoot.in[1] <== inputTree.root;
checkRoot.enabled <== inputAmount;
```

could be rewritten as

```
(inputTree.root-inputRoot)*inputAmount===0
```

**Comment**

Won’t fix. In our opinion it’s more readable as is.

## Severity Terms

### Minor

Minor issues are generally subjective in nature, or potentially deal with topics like “best practices” or “readability”. Minor issues in general will not indicate an actual problem or bug in code.

The maintainers should use their own judgment as to whether addressing these issues improves the codebase.

### Medium

Medium issues are generally objective in nature but do not represent actual bugs or security problems.

These issues should be addressed unless there is a clear reason not to.

### Major

Major issues will be things like bugs or security vulnerabilities. These issues may not be directly exploitable, or may require a certain condition to arise in order to be exploited.

Left unaddressed these issues are highly likely to cause problems with the operation of the contract or lead to a situation which allows the system to be exploited in some way.

### Critical

Critical issues are directly exploitable bugs or security vulnerabilities.

Left unaddressed these issues are highly likely or guaranteed to cause major problems or potentially a full failure in the operations of the contract.