Security assessment and code review

Initial Delivery: August 6, 2021
Updated Report: September 14-22, 2021

Prepared for:
Wei Jie Koh | Ethereum Foundation
Cory Dickson | Ethereum Foundation

Prepared by:
Er-Cheng Tang | HashCloak Inc
Karl Yu | HashCloak Inc
Mikerah Quintyne-Collins | HashCloak Inc

Table Of Contents

Executive Summary

Minimal Anti-Collusion Infrastructure (MACI) aims to provide a quadratic voting system that disincentives collusion behavior. It is deployed as Ethereum smart contracts where users can sign up and make anonymous votes. The work of processing these votes is delegated to a coordinator, who has the privilege to read the encrypted votes and has to prove its correct execution to the smart contract.

From July 5th, 2021 to August 2nd, 2021, the Ethereum Foundation's Applied ZKPs team engaged HashCloak for an audit of the MACI protocol. The audit was conducted with 3 auditors over 15 person weeks. The MACI codebase was assessed at commit 2db5f625b67a6b810bd851950d7a42c26189088b and the circomlib version that was used was assessed at commit 0e78dde8c813b95f4585b0613927e9c4269de500. From August 18, 2021 to September 22, 2021, we assisted the MACI team in resolving the issues brought up in this report and have updated the report with the status of each issue brought up. The latest commit with these changes is at 4bc217db24f573c1da5e24ad0caca596637c3dc8.

The following packages were in scope:

- `Circuits`
- `Contracts`
- `Core`
- `Crypto`
- `Domainobjs`

During the first week of the audit, we familiarized ourselves with the MACI codebase and specification. In the second week, we started the first phase of manually reviewing the contract. This phase lasted until the fourth week of the audit.

We found a variety of issues ranging from critical to informational.

Severity	Number of Findings
Critical	1
High	0
Medium	1
Low	1
Informational	3

Overview

The MACI protocol consists of three subsystems in different programming languages:

- Smart contracts: These smart contracts handle the management and on-chain voting aspects of the system. These provide developers with the tools to integrate MACI into their Ethereum smart contracts. These contracts are written in Solidity.
- Zero-knowledge proof circuits: In order to privately handle votes and enforce non-collusion without the aid of a trusted third party, MACI makes use of zero-knowledge proofs. The circuits for these zero-knowledge proofs are written in Circom.
- Business logic: This code manages the handling between the smart contracts and the circuit code. This is written in Typescript.

The main components of the smart contracts are:

- MACI.sol : Allow user sign-ups; deploy polls; compress state queue
- Poll.sol : Accept votes; verify state updates, ballot updates, and vote tally updates
- VkRegistry.sol : Register verification keys

The main components regarding secretly proving and publicly verifying are :

- ecdh.circom : Recover the ephemeral encryption key
- messageToCommand.circom : Decrypt and parse the message
- verifySignature.circom : Check for the authenticity of the message
- messageValidator.circom : Check for the overall validity of the message
- stateLeafAndBallotTransformer.circom : Partially account for a single message
- processMessages.circom : Update the commitment of all states and ballots
- tallyVotes.circom : Update the commitment of the vote tally

The main components for the business logic are:

- Core: Handles the core data structures and methods for handling MACI.
- Crypto: Implements and provides wrappers around the core cryptographic primitives used in MACI.
- DomainObjs: Contains domain objects for data structures used in MACI.

Findings

Initial Conditions Are Not Properly Enforced

Type: Critical
Files affected: tallyVotes.circom

In `ResultCommitmentVerifier`, the verification of `currentTallyCommitment` is skipped when processing the first batch, so as to allow accepting the value zero. Hence, there should be other checks confirming that the current tally is actually the initial tally in such a case, but this is absent from the current implementation. Otherwise, a malicious coordinator could actually start with an arbitrary tally and thus affect the tally result.

Impact: A malicious coordinator can affect the tally result, violating correct execution.
Suggestion: Add constraints to the current tally in case of the first batch. Another option is to avoid skipping verification, and to initialize the tally commitment in Poll.sol with a valid and expected commitment.

Status: This issue is has been partially resolved as of commit e61c684e2b61350a8adf44686184a3fe1cd9d05a, 4bc217db24f573c1da5e24ad0caca596637c3dc8 and 4892ada0dc7bcc2f7710aea917e309f1553317c5 as only constraints have been added to the tallyVotes circuit but not hasn't been enforced in Poll.sol. Due to the contract bytecode size limit, the development team has decided against also initializing the tally commit in Poll.sol. As such, an issue has been opened to keep track of this for future iterations of MACI.

Upper Bounds Are Not Properly Enforced

Type: Medium Severity
Files affected: MACI.sol

Accounting for the usage of the accumulation queue, the number of signups should be at most `STATE_TREE_ARITY ** stateTreeDepth`. However, the `signUp` function does not make limitations according to this upper bound; it uses a looser bound `2 ** 50`

instead. In case the former upper bound is reached, the contract will not be able to merge the accumulation queue, which hinders the execution of further steps.

Impact: The contract will become useless once the number of signups exceeds the limit.
Suggestion: Fix the upper bound on `numSignUps` (in line 193).
Status: This issue has been fixed in commit [58739097aa15816e6d0313eb70d72127a36823bf]

Indexing Error

Type: Low Severity
Files affected: Poll.sol

For the processing of messages, it is expected to process all messages with the reversed order. In the current implementation, the initial indexing might miss a few messages that are positioned at the end of the list. The code starting from line 481:

```
currentMessageBatchIndex = (numMessages / messageBatchSize) * messageBatchSize;
if (currentMessageBatchIndex > 0) { currentMessageBatchIndex -= messageBatchSize; }
```

When `numMessages` is larger than and not divisible by `messageBatchSize`, the starting index would be `messageBatchSize` less than the intended index.

Impact: Some messages might be skipped unintentionally during the processing stage.
Suggestion: Fix the calculation of the initial indexing.
Status: This issue has been fixed in commits [021ec436456b84a24c0ce320d82bf860dc4be4b3] and [2693c23c90134ae7c51e4587d9f08507fd3acb8a].

Vulnerable NPM packages

Type: Informational
Files affected: package.json

When running npm audit, there are several packages that require an update due to having a regex denial of service vulnerability. These packages mainly concern the Lerna package for managing monorepos and do not affect the security of the MACI-specific portions of the typescript code.

Impact: Regex denial of service attacks on the lerna npm package may slow down or hang a typescript program.
Suggestion: Update all dependencies to their newest versions and replace ones that haven't been updated.
Status: This issue has been fixed in commit 97957abd6bf9b6743943e191c9217b89a93ecd21.

Inconsistency Between Comments And Implementations

Type: Informational
Files affected: Poll.sol

Prior to the declaration of the variable `tallyCommitment`, the comments in lines 399-405 indicated that it should be initialized as a commitment to a series of zeros. However, both the contract code and the related circom code require its initialization to take the value zero, so that the comment does not apply.

Impact: The inconsistency may cause confusion to developers and code reviewers.
Suggestion: Make the comments in accordance with the implementation.
Status: This issue has been fixed in commit 9b728fc08de32bca4b759fa292a4e78278510667.

Incomplete Documentation for Typescript business logic

Type: Informational
Files affected: README.md for `core`, `crypto` and `domainobjs` packages

In the READMEs for `core`, `crypto` and `domainobjs` packages, there is a little bit of documentation explaining the package. But this documentation is incomplete

Impact: Makes it harder for users of MACI to understand the relevant code portions they may want to use for their applications.
Suggestion: Complete the documentation
Status: Extra documentation for the aforementioned modules has been added in commit 022da6f06d53be90a63d3bb89c2873d5103d1b90. We would also like to note that documentation is an ongoing process and as such, the MACI team should add documentation whenever a major change has occurred in the codebase.