MACI Security Audit

By Kyle Charbonnet (PSE Security), Yufei Li (PSE Security)

July, 2024

Table of Contents

Overview

Executive Summary

Since the last audit, MACI code has been updated that addressed issues found in the last audit, removed some functionalities, modularized certain parts of the code, as well as added some new functionalities.

These changes were audited with a focus on the smart contracts, typescript core, and Circom circuits. The contracts followed solidity best practices with great documentation. Only minor issues and gas optimization were found, while one major issue was found in an open PR, which was suggested to be reviewed as well. All issues have been addressed. Overall, the MACI codebase is complex, yet well documented and thoroughly reviewed.

Background

MACI is a zero-knowledge protocol that acts as a collusion resistant voting platform. The zero-knowledge proof generation and vote tallying process is all done by a single actor known as the coordinator. However, the coordinator cannot censor individual votes or tamper with the results. The voting process deters buying votes by allowing voters to secretly change their votes. Not even the voters themselves can prove to a briber who they voted for - provided the coordinator does not reveal their secret key. This functionality enables a wide range of use cases where vote bribing is popular.

The three main components of MACI are the smart contracts, the core typescript, and the Circom circuits. The smart contracts manage the voting data and zero-knowledge proof validation on-chain so that everyone can verify the voting inputs. The core typescript provides a simple interface for users to publish votes on-chain and so the coordinator can tally the voting results. The circuits are used to generate zero-knowledge proofs that the votes were counted correctly. The proof can then be verified through a verifier smart contract on-chain.

Coverage

Github Repo: https://github.com/privacy-scaling-explorations/maci
Commit Hash: 14e89baea1ede9bc9bad79fafb5c362d6a6a81e9
Branch: chore/audit-v2.0.0
Documentation: https://maci.pse.dev/docs/introduction/
Changes since last audit:
https://www.notion.so/pse-team/MACI-v2-0-0-Audit-fbd726ed4b704402a4dca5536921d4e9
The open PR that optimizes processMessage circuit:
https://github.com/privacy-scaling-explorations/maci/pull/1644/files
The previous version of the verifier supported only a single parameter, necessitating the use of packed values and SHA256 input hashes. With the latest version, which supports multiple input values, and due to reduced gas costs on-chain, input values are now directly utilized:
https://github.com/privacy-scaling-explorations/maci/pull/1700

Code in maci/contracts, scripts in maci/core, maci/crypto, maci/cli, maci/domainobjs, and circuits in maci/circuits were reviewed.

General Analysis

Category	Evaluation
Access Control	**Strong**. Access is limited as intended, mainly to the coordinator
Launch Risk	**Strong**. MACI does not manage assets. Additionally many dApps built using MACI will deploy their own MACI contract. They should

	consider launch controls if necessary.
Code Quality	**Strong**. Code follows best practices for solidity, typescript, and Circom. No unnecessary use of assembly. No confusing variable/function names. Good use of interfaces and inheritance.
Decentralization	**Moderate.** Coordinator can choose to never publish the voting results, thus halting the vote. However, they are not able to censor individual votes or publish incorrect voting results.
Events	**Strong**. Events are emitted after every important function call.
Dummy Proof	**Strong**. Contract function names are clear in their intent and hard to misuse. The code makes this complex protocol as simple as can be.
Complexity	**Moderate**. Contracts are short and simple. Typescript is easy to read and understand. However, the circuit logic is non-trivial and can be difficult to reason through.
Testing	**Strong**. Strong and well written unit and integration tests.
Documentation	**Strong**. NatSpec comments for all functions and good documentation on the website.
Cryptography	**Strong.** Follows best practices and utilizes SNARK safe cryptography when needed.
ZK Circuits	**Moderate**. The circuits are quite complex, and as described in one of the major issues, bugs can be easily introduced.

Findings

Findings On New Commit - July 2024

Major

[M1] **from the open PR instead of the audit ready codebase*

Misuse of computedIsStateLeafIndexValid instead of computedIsVoteOptionIndexValid

Location

https://github.com/privacy-scaling-explorations/maci/blob/52c9711929ee28f7cf5395f18629d6a180714474/circuits/circom/core/qv/processMessages.circom#L457

Description

```
      var cmdVoteOptionIndexMux = Mux1()([0, cmdVoteOptionIndex],
computedIsStateLeafIndexValid);
```

`cmdVoteOptionIndexMux’s` value should be based on `computedIsVoteOptionIndexValid` not `computedIsStateLeafIndexValid`. With this misuse, presumably `cmdVoteOptionIndex` larger than `maxVoteOptions` will be viewed as valid, thus leading to unforeseen consequences down the line.

Implemented Fix

Replace `computedIsStateLeafIndexValid` with `computedIsVoteOptionIndexValid`
https://github.com/privacy-scaling-explorations/maci/blob/9766bbfcecd41bd0f39c1422978d959705b1b500/circuits/circom/core/qv/processMessages.circom#L415

[M2] Missing coordPubKey checking while processing messages can result in coordinator censoring all or selected batches

Location

The code below were removed from the new commit:
https://github.com/privacy-scaling-explorations/maci/blob/be7a6598b7aeab2e11717bea509c697d61b556db/circuits/circom/core/non-qv/processMessages.circom#L135-L149
https://github.com/privacy-scaling-explorations/maci/blob/be7a6598b7aeab2e11717bea509c697d61b556db/contracts/contracts/MessageProcessor.sol#L246

Description

The previous version of the verifier supported only a single parameter, necessitating the use of packed values and SHA256 input hashes. With the latest version, which supports multiple input values, and due to reduced gas costs on-chain, input values are now directly utilized. Comparing the input variables between two commit:
Previous:

```
      // pack the values
      uint256 packedVals = genProcessMessagesPackedVals(
      _currentMessageBatchIndex,
      _numSignUps,
      _numMessages,
      _messageTreeSubDepth,
      _voteOptionTreeDepth
      );
      (uint256 deployTime, uint256 duration) =
poll.getDeployTimeAndDuration();

      // generate the circuit only public input
      uint256[] memory input = new uint256[](7);
      input[0] = packedVals;
      input[1] = coordinatorPubKeyHash;
      input[2] = _messageRoot;
      input[3] = _currentSbCommitment;
      input[4] = _newSbCommitment;
      input[5] = deployTime + duration;
      input[6] = actualStateTreeDepth;
      inputHash = sha256Hash(input);
   }
```

and the new version:

```
       publicInputs = new uint256[](8);
      publicInputs[0] = numSignUps;
      publicInputs[1] = deployTime + duration;
      publicInputs[2] = messageAq.getMainRoot(messageTreeDepth);
      publicInputs[3] = poll.actualStateTreeDepth();
      publicInputs[4] = batchEndIndex;
      publicInputs[5] = _currentMessageBatchIndex;
      publicInputs[6] = (sbCommitment == 0 ? poll.currentSbCommitment() :
sbCommitment);
      publicInputs[7] = _newSbCommitment;
```

We see coordinatorPubKeyHash is missing. Because the coordinator pub key hash is not part of the public inputs anymore, the coordinator can run the circuit with an invalid private key (not associated with the public key on chain) and have a tally of 0 validated successfully - or even worse, he can pass the wrong private key for some batches, and the correct one for other batches.

The fix is to have the pub key hash as public input, still accept the private key as private input, derive public key from private, hash it and check that it matches the public input on-chain.

Implemented fix
https://github.com/privacy-scaling-explorations/maci/blob/9766bbfcecd41bd0f39c1422978d959705b1b500/contracts/contracts/MessageProcessor.sol#L159-L168
https://github.com/privacy-scaling-explorations/maci/blob/9766bbfcecd41bd0f39c1422978d959705b1b500/circuits/circom/core/non-qv/processMessages.circom#L187-L189

[R1] Optimization: Extra Code

Location

https://github.com/privacy-scaling-explorations/maci/blob/14e89baea1ede9bc9bad79fafb5c362d6a6a81e9/circuits/circom/utils/messageToCommand.circom#L48-L57

Description

There are multiple instances, where in .circom files, arrays can be used directly in assignment, instead of assigning each element in a loop.

As an example, the following block

```
   var computedDecryptor[DECRYPTED_LENGTH] =
PoseidonDecryptWithoutCheck(MSG_LENGTH)(
        [
            message[0], message[1], message[2], message[3],
            message[4], message[5], message[6], message[7],
            message[8], message[9]
        ],
        0,
        computedEcdh
    );
```

can be simplified to:

```
       var computedDecryptor[DECRYPTED_LENGTH] =
PoseidonDecryptWithoutCheck(MSG_LENGTH)(
        message,
        0,
        computedEcdh
    );
```

Implemented fix

https://github.com/privacy-scaling-explorations/maci/blob/ad04340972f3d35a8d4915227d15c5502af7fe63/circuits/circom/utils/messageToCommand.circom#L48-L52

[R2] Remove dead code

Location

https://github.com/privacy-scaling-explorations/maci/blob/14e89baea1ede9bc9bad79fafb5c362d6a6a81e9/contracts/contracts/Tally.sol#L254-L257

Description

`tally` is defined and assigned in the function `verifySpentVoiceCredits`, but never used.

Implemented fix

https://github.com/privacy-scaling-explorations/maci/blob/ad04340972f3d35a8d4915227d15c5502af7fe63/contracts/contracts/Tally.sol#L257

[G1] Gas Optimization

Location

https://github.com/privacy-scaling-explorations/maci/blob/14e89baea1ede9bc9bad79fafb5c362d6a6a81e9/contracts/contracts/trees/LazyIMT.sol#L156

Description

The code below is calculating exp at each step, which is quite gas heavy,

```
while (uint40(2) ** uint40(depth) < numberOfLeaves) {
  depth++;
}
```

Right shift is much more gas efficient for this scenario:

```
while ((numberOfLeaves >> depth) > 1) {
  depth++;

}
```

Implemented fix

https://github.com/privacy-scaling-explorations/maci/blob/8f951b0fcc73212d99d3ddc962fbaa9788471fb6/contracts/contracts/trees/LazyIMT.sol#L156

Fix Log

Issue	Severity	Status
[M1]	Major	Fixed. https://github.com/privacy-scaling-explorations/maci/blob/9766bbfcecd41bd0f39c1422978d959705b1b500/circuits/circom/core/qv/processMessages.circom#L415
[M2]	Major	Fixed. https://github.com/privacy-scaling-explorations/maci/blob/9766bbfcecd41bd0f39c1422978d959705b1b500/contracts/contracts/MessageProcessor.sol#L159-L168

		https://github.com/privacy-scaling-explorations/maci/blob/9766bbfcecd41bd0f39c1422978d959705b1b500/circuits/circom/core/non-qv/processMessages.circom#L187-L189
[R1]	Recommendation	Fixed. https://github.com/privacy-scaling-explorations/maci/blob/ad04340972f3d35a8d4915227d15c5502af7fe63/circuits/circom/utils/messageToCommand.circom#L48-L52
[R2]	Recommendation	Fixed. https://github.com/privacy-scaling-explorations/maci/blob/ad04340972f3d35a8d4915227d15c5502af7fe63/contracts/contracts/Tally.sol#L257
[G1]	Gas Optimization	Fixed. https://github.com/privacy-scaling-explorations/maci/blob/8f951b0fcc73212d99d3ddc962fbaa9788471fb6/contracts/contracts/trees/LazyIMT.sol#L156

Vulnerability Classifications

Severity Categories	
Severity	**Description**
Recommendation	Information not relevant to security, but may be helpful for efficiency, costs, etc.
Warning	The issue does not pose an immediate security threat, but may be a lack of following best practices or more easily lead to the future introductions of bugs.
Minor	The code does not work as intended. Impact to the system and users is minimal if present at all.
Major	The issue can lead to **moderate** financial, reputation, availability, or privacy damage. Or the issue can lead to substantial damage under extreme and unlikely circumstances.
Critical	The issue can lead to **substantial** financial, reputation, availability, or privacy damage.