MACI Security Audit

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

February, 2024

Table of Contents

Overview

Executive Summary

The MACI code base was audited with a focus on the smart contracts, typescript core, and Circom circuits. The contracts followed solidity best practices with great documentation. Three critical bugs were found, two within the Circom circuits and one in the smart contracts. All three of these have been fixed, but a new trusted setup ceremony is required before these fixes are active. 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: 3aa4f33aa7f4558f16da65b5a3fb93b282bd4fe5
Branch: audit-branch
Documentation: https://maci.pse.dev/docs/introduction/

The non quadratic voting circuits were included in a later commit: 86ba8548780049245482e5277cc47f4a8776e9e6. These circuits were also reviewed as part of this audit.

The following component was not included in this audit -

- Subsidy components

All other contracts 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 2 critical bugs were found within them.

Findings

Critical

[C1] Incorrect Matching between State Leaf and Message

Location
https://github.com/privacy-scaling-explorations/maci/blob/3aa4f33aa7f4558f16da65b5a3fb93b282bd4fe5/circuits/circom/processMessages.circom#L537-L559

Description
In the MACI protocol, each message submitted by a user has an index that states which state leaf it is meant to modify. However, if a message is invalid, which can happen for a variety of reasons, then it shouldn't modify any stateleaf. Since invalid messages are expected in the protocol, they should not cause the circuits to fail. In order to support this, the coordinator is expected to pass in the state leaf at index 0 for any invalid messages. So the circuit will mark the message as invalid, then check that the passed in state leaf exists at the 0 position in the Merkle tree.

The problem is that a message can be rendered invalid if it is compared to a state leaf index that does not match the message's state leaf index. So there is an issue where a coordinator sees an invalid message, and so compares it to the 0 state leaf. But by comparing it to the 0 state leaf, the message will be marked as invalid since it was likely intended for a different leaf.

The attack here is that a coordinator can censor any user message by simply comparing it to the 0 leaf, even if the message is valid. Since the message is likely intended for a different state leaf, the circuits will mark it as invalid and this vote will not be tallied.

Implemented Fix
The implemented fix was to constrain the comparison of any message to its intended state leaf. As long as the message index is less than the number of sign ups, then the compared state leaf must be at that index. This still allows messages with non-sensible (too large) state leaf indices to be marked as invalid, while requiring any valid index to match to the given state leaf.
https://github.com/privacy-scaling-explorations/maci/pull/1170

[C2] Incorrect Matching between Current Vote Weight and Vote Option

Location
https://github.com/privacy-scaling-explorations/maci/blob/3aa4f33aa7f4558f16da65b5a3fb93b282bd4fe5/circuits/circom/processMessages.circom#L595-L612

Description

This bug is very similar to [C1] but for the vote option index instead of the state leaf index. If a message is invalid, the coordinator is expected to pass in the currentVoteWeight the user has for vote option 0. If the message is valid, then the currentVoteWeight is expected to be the vote weight for the message's vote option index. However, a message can be marked invalid if the newVoteWeight is greater than the currentVoteWeight for the intended vote option plus any remaining vote weight the user has left.

The attack here is that the coordinator can censor votes by using currentVoteWeight for vote option 0, even when the message is intended for a different vote option. Then if currentVoteWeight plus the remaining balance is less than the newVoteWeight, the message will be marked invalid. So this attack only works when the vote weight for option 0 is insufficient to cover the new vote weight.

Implemented Fix
The implemented fix was to constrain the currentVoteWeight to its intended vote option. As long as the vote option index is less than the number of vote options, then the currentVoteWeight must be the correct amount for that option. This still allows messages with non-sensible (too large) vote option indices to be marked as invalid, while requiring any valid index to match to the given currentVoteWeight.
https://github.com/privacy-scaling-explorations/maci/pull/1170

[C3] Missing Check for Message Type

Location
https://github.com/privacy-scaling-explorations/maci/blob/6f64471e9f06b751297ea33d48d6f1e5b4a86fb9/contracts/contracts/Poll.sol#L168

Description
In the MACI protocol, there are two types of messages - voting/key change messages and topup messages. The message stores which type of message it is in its first index (msgs[0]), where 1 signals it is a voting/key change message, and 2 signals it is a topup message. In the smart contracts, a user can publish their own message with their own parameters via the publishMessage function in poll.sol. If they want to publish a topup message to increase their voting balance, they must use the topup function in poll.sol, which will first attempt to retrieve the new topup amount for the user. If that doesn't succeed, then the user can't submit a topup message.

The problem is that when calling publishMessage, the user can input the message type as type 2. So it will be regarded as a topup message in the circuits. This allows malicious users to topup their balance without actually having the credits on-chain. So any malicious user can have more voting power than intended.

Implemented Fix
The fix was to add a check in the publishMessage function to ensure that it is of type 1.
https://github.com/privacy-scaling-explorations/maci/pull/1201

Major

[M1] DDoS to prevent poll owner from generating mergedStateRoot

Location
https://github.com/privacy-scaling-explorations/maci/blob/3aa4f33aa7f4558f16da65b5a3fb93b282bd4fe5/contracts/contracts/Poll.sol#L188

Description
After voting is over, the poll owner is supposed to call the mergeMaciStateAqSubRoots function in Poll.sol repeatedly, until all of the subtrees are merged. Then the owner will call the function mergeMaciStateAq to create the mergedStateRoot. However, the last call may always be front-run by a signup function call by an unknowing or malicious user, such that subTreesMerged is no longer True.

If the signup gate keeper maintains a limited registration quota, then this is a gas-grief attack, as mergeMaciStateAqSubRoots is more expensive than a signup. However, if there is no limit on registration, the poll may be DDoSed to prevent tallying.

Implemented Fix
The fix involved adding a check to ensure that all subtrees have been merged before allowing any additional sign-ups.
https://github.com/privacy-scaling-explorations/maci/pull/1218

Minor

[N1] Unchecked output of LessEqThan Circuit

Location
https://github.com/privacy-scaling-explorations/maci/blob/3aa4f33aa7f4558f16da65b5a3fb93b282bd4fe5/circuits/circom/processMessages.circom#L398

Description
The validCreditBalance circuit is meant to ensure that the user's previous balance is less than or equal to the new credit balance after a topup. If the topup amount is too large, it can cause an overflow, so this circuit would then output 0. However, the output of this circuit is never checked.

Since the topup amount has to be larger than 252 bits, this event is extremely unlikely, and thus this issue is minor.

Implemented Fix
The fix used the output of the validCreditBalance circuit to determine which value, the previous balance or new balance, to update the user's balance to. Also documentation was included to ensure that any MACI users are expected to limit total credits given to users at 32 bits.

https://github.com/privacy-scaling-explorations/maci/pull/1225
https://maci.pse.dev/docs/topup

[N2] Inaccurate Number of Valid State Indices

Location
https://github.com/privacy-scaling-explorations/maci/blob/3aa4f33aa7f4558f16da65b5a3fb93b282bd4fe5/circuits/circom/messageValidator.circom#L19

Description
The state tree should contain numSignUps valid leaves. However, the circuit counts the index=numSignUps as a valid index where the state leaf indices are 0 based. So the max valid index should be numSignUps - 1. Since the state leaf at numSignUps is a 0 leaf, the user or coordinator can't manipulate the voting process with this issue. Therefore this issue is minor.

Implemented Fix
https://github.com/privacy-scaling-explorations/maci/pull/1200

Warning

[W1] Missing SNARK Field Size Check

Location
https://github.com/privacy-scaling-explorations/maci/blob/4658f3628df173e25aac200e2629db1080a7745e/contracts/contracts/Poll.sol#L94

Description
The coordinator public key given to the Poll contract is not checked to be within the SNARK field size. Thus, if the coordinator gives a public key greater than the field size, unexpected behavior could occur. However, this behavior would still not let the coordinator to manipulate the voting process, so this is simply a warning.

Suggested Solution
Add a check:

```
if (_coordinatorPubKey.x >= SNARK_SCALAR_FIELD || _coordinatorPubKey.y >= SNARK_SCALAR_FIELD) {
  revert MaciPubKeyLargerThanSnarkFieldSize();
}
```

Implemented fix
https://github.com/privacy-scaling-explorations/maci/pull/1204

[W2] Bribery is Still Possible

Description
MACI is aimed to minimize the incentive of bribery, by introducing reverse order message processing. However, if the briber demands the private key from the bribed, the best scenario will be a race to see who submitted the last vote. Considering that the briber is likely to be more technically sophisticated, there is a large probability that the briber will win this race. However, this bribing method is costly for a large attack, since there will be a race with each individual bribed user. Therefore this is simply a warning.

Possible Solution
By introducing a new password parameter in the message, MACI has a chance to minimize the incentive for private key bribery.

To vote, a user needs to include a password in the message. Only the password that was sent with the first message is the valid one. To make valid votes afterwards, the user would need to continue using the same password. Users could send messages with other passwords, but they will be invalid.

Now, if a user is sending the first message from a different ETH address than the one used for signup, no one other than the coordinator will be able to associate the message with the user's MACI keys.

When a briber asks for the private key and password, the user could happily provide the wrong password along with the correct MACI and ephemeral private keys that are associated with the wrong passwords in transactions.

Equipped with private keys, the briber could fetch all messages sent to the poll and decrypt all of them to find out the one which decrypts properly using the correct private key. They then would see the MACI public key of the user being included in the plaintext message. However, at that point, they still could not be sure that the first message is the actual first message because even if it had nonce set to 1, it could be the second message and not be valid, so the password might not be the correct one.

Recommendation

[R1] Optimization: Extra Code

Location
https://github.com/privacy-scaling-explorations/maci/blob/4658f3628df173e25aac200e2629db1080a7745e/contracts/contracts/MessageProcessor.sol#L97-L101

Description
The code block can be simplified to:

```
uint256 r = numMessages % messageBatchSize;
currentMessageBatchIndex = numMessages;
```

Implemented fix
https://github.com/privacy-scaling-explorations/maci/pull/1072/files#diff-f39275978242a9b1a7f74784b2e64eaa347acd73c535c35d92f46246c7618c15

[R2] Optimization: Unnecessary Assignment

Location
https://github.com/privacy-scaling-explorations/maci/blob/776f9ece4191c765e59168b3214418e461af909c/contracts/contracts/utilities/Utilities.sol#L42-L47

Description
The values dat[2] through dat[10] default to 0, so no need to assign them explicitly.

Implemented fix
https://github.com/privacy-scaling-explorations/maci/pull/1204/files#diff-d63b651b32439543231abca52b1794fc2605e677a569e898e50d271297f62ccd

[R3] Change inequality sign

Location
https://github.com/privacy-scaling-explorations/maci/blob/4658f3628df173e25aac200e2629db1080a7745e/contracts/contracts/Poll.sol#L167

Description
It is best practice to use '>=' instead of '==' here if the intended behavior is to ensure that the number of messages is bounded above.

Suggested Solution
Change '==' to '>='.

Implemented fix
https://github.com/privacy-scaling-explorations/maci/pull/1204/files#diff-fa7d63f7f5375be5c1497cad1815404fcac0d30fe52897bc5fef7400cfecc5fb

Fix Log

Issue	Severity	Status
[C1]	Critical	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1170

[C2]	Critical	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1170
[C3]	Critical	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1201
[M1]	Major	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1218
[N1]	Minor	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1225 https://maci.pse.dev/docs/topup
[N2]	Minor	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1200
[W1]	Warning	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1204
[W2]	Warning	Not Addressed. Requires large changes and may be addressed in future versions.
[R1]	Recommendation	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1072/files#diff-f39275978242a9b1a7f74784b2e64eaa347acd73c535c35d92f46246c7618c15
[R2]	Recommendation	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1204/files#diff-d63b651b32439543231abca52b1794fc2605e677a569e898e50d271297f62ccd
[R3]	Recommendation	Fixed. https://github.com/privacy-scaling-explorations/maci/pull/1204/files#diff-fa7d63f7f5375be5c1497cad1815404fcac0d30fe52897bc5fef7400cfecc5fb

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.