HashCloak

Code Review and Security Assessment For *MACI* Tally Contract

Initial Delivery: Oct 8th, 2025

Prepared For:
John Guilding | *Ethereum Foundation*

Prepared by:
Manish Kumar | *HashCloak Inc.*

Table Of Contents

Executive Summary

The *Ethereum Foundation's* PSE team engaged HashCloak Inc to perform code review and security assessment for MACI's Tally contract, a core component responsible for aggregating and verifying the results of a MACI-based voting process and funding related infrastructure.

As part of the engagement, HashCloak conducted a comprehensive security review of the Tally contract focused on identifying common security vulnerabilities, best practices, logical errors, and architectural designs that could impact the integrity of vote tallying, fund allocation, and access control mechanisms. We also ran Slither, a smart contract static analyzer tool for detecting vulnerabilities in the solidity smart contract.

Overall, we found the code to be well documented and the implementation adheres to Solidity best practices.

We found the issues range from Medium to Informational:

Severity	**Number of Findings**
Critical	0
High	0
Medium	1
Low	2
Informational	2

Scope

For the audit, the given repository was considered:

- https://github.com/privacy-ethereum/maci-platform at commit 853158e4d0e4bef4a4dcad41a2dac7170d99aa19

With following files in scope:

- packages/contracts/contracts/maci/Tally.sol

Overview

Minimal Anti-Collusion Infrastructure (MACI) is an open-source public good that serves as infrastructure for private on-chain voting. It provides privacy and collusion resistance for on-chain voting, both in a quadratic and non-quadratic fashion by using encryption and zero-knowledge proofs (zk-SNARKs) to hide how each person voted while still publicly revealing the final result.

The (in scope) Tally contract serves as the core component responsible for aggregating and verifying the results of a MACI-based voting process. It extends functionality from the TallyBase (from maci-contracts/contracts/Tally.sol) contract and integrates with verifier for finalizing and exposing verifiably correct aggregated voting outcomes while maintaining the privacy.

Methodology

The audit was conducted through a combination of manual verification and using static analyzer tools and cross verifying the solana's common security vulnerabilities, best practices and architectural designs of the tally contract. While the primary focus was the in-scope codebase, we also inspected dependency behavior and interactions with external components such as the Verifier, Registry, and Tally (from maci-contracts/contracts/Tally.sol).
We also used Slither, a smart contract static analyzer tool for detecting any potential vulnerabilities in the smart contract.

Overview of Evaluated Components

Tally.sol

- Access control: use of onlyOwner
- Correct Initialization of contract & preventing re-initialization
- Arithmetic safety
- Error handling and revert conditions
- Data consistency
- Reentrancy and external calls
- Withdrawal authorization
- Any undefined behavior
- Usage of dependencies/import
- Missing access control modifiers for a function
- Initialization, updation and uses of state variables
- Incorrect role-based access control
- General logic errors
- Block Timestamp Manipulation
- Lack of Input Validation

Findings

MACI-1: `Pausable` imported but not used

Type: Medium
Files affected:

- packages/contracts/contracts/maci/Tally.sol

Description: In Tally.sol ,`Pausable` is imported and inherited in the Tally contract but none of the functions in the Tally contract use the associated modifiers. The pause and unpause can be added with onlyOwner modifier for pausing/unpausing contract calls. The Pausable modifier (`whenNotPaused`) can be applied to functions that modify critical state or handle funds, such as `deposit`, `addTallyResults`, and `claim` to allow emergency halting of protocol activity, if needed.

Recommendation: If the intention is to allow pause/unpause then add onlyOwner exposed functions to pause/unpause otherwise the import should be removed to avoid any confusion.

Status: TBD

MACI-2: `getAllocatedAmounts` does not take into account the updated alpha and may give incorrect result

Type: Low
Files affected:

- packages/contracts/contracts/maci/Tally.sol

Description: The `getAllocatedAmounts` is a public view function which gives the allocated token amount (without verification), callable by any user. The function calls the `getAllocatedAmount` which uses `alpha` to calculate the allocated amount. It must be noted that the value of `alpha` is 0 before the first claim. if anyone calls `getAllocatedAmounts` before any claim is made, that will give an incorrect result as `alpha` is 0. Note that in the `claim` function, it is checked whether alpha is 0 or not in the very beginning and based on that the updated alpha is calculated.

Impact: If the `getAllocatedAmounts` is called before any claims, that will give incorrect allocated token amount

Recommendation: Add same check present in the claim function in the beginning of the getAllocatedAmounts function:

```
if (alpha == 0) {
      alpha = calculateAlpha(totalAmount());
    }
```

Status: TBD

MACI-3: `tallyResults` for a given index is fetched without checking `isSet` for a given index

Type: Low
Files affected:

- packages/contracts/contracts/maci/Tally.sol

Description: The function `claim` and view helper `getAllocatedAmount` directly access `tallyResults[params.index].value` without verifying whether the result for that index has actually been set ie, `tallyResults[params.index].isSet` true or not which indicates that this value was set and initialized. If isSet is false, the value field defaults to 0, which may lead to incorrect allocated token amounts computation and potentially allowing invalid or empty recipients to claim allocations.

Impact: May lead to incorrect allocated token amounts computation and potentially allowing invalid or empty recipients to claim allocations or logical inconsistencies.

Recommendation: Add an explicit validation step in all functions that read tallyResults to ensure the tally entry is valid:

```
require(tallyResults[index].isSet, "Tally result not set");
```

Status: TBD

MACI-4: `deposit` function allows 0 amount deposits

Type: Informational
Files affected:

- packages/contracts/contracts/maci/Tally.sol

Description: The `deposit()` function allows users to deposit tokens into the contract without verifying that the deposit amount is non-zero. While this does not move any tokens, it still emits a Deposited event and executes a safeTransferFrom() which is unnecessary and redundant.

Recommendation: Add an explicit amount check at the beginning of the function to prevent zero-value deposits:

```
require(amount > 0, "Deposit amount must be greater than zero");
```

Status: TBD

MACI-5: Inconsistency in code comment and MAX_VOICE_CREDITS definition

Type: Informational
Files affected:

- packages/contracts/contracts/maci/Tally.sol

Description: The comment states that MACI allows 2 ** 32 voice credits max but the `MAX_VOICE_CREDITS` constant is hardcoded as 10 ** 9. This creates a discrepancy between documentation and implementation and can create confusion and/or potentially leading to incorrect scaling.

Recommendation: Ensure that MAX_VOICE_CREDITS matches the actual maximum used in MACI core logic. If the true limit is indeed 2 ** 32, the definition should be corrected otherwise, update the comment to reflect the intended value to avoid confusion.

Status: TBD

References

- https://github.com/privacy-ethereum/maci
- https://github.com/crytic/slither
- https://github.com/OpenZeppelin/openzeppelin-contracts
- https://maci.pse.dev/docs/introduction

Appendix

- Result from Slither tool (Clipped) found to be false positive

```
'npx hardhat clean' running (wd: maci-platform/packages/contracts)
'npx hardhat clean --global' running (wd: maci-platform/packages/contracts)
'npx hardhat compile --force' running (wd: maci-platform/packages/contracts)
ERROR:ConvertToIR:Function not found init
ERROR:ContractSolcParsing:Impossible to generate IR for MACI.initPoll
(contracts/maci/MACI.sol#54-60):
 'NoneType' object has no attribute 'type'
ERROR:ConvertToIR:Function not found setRegistry
ERROR:ContractSolcParsing:Impossible to generate IR for MACI.setPollRegistry
(contracts/maci/MACI.sol#65-70):
 'NoneType' object has no attribute 'type'
ERROR:ConvertToIR:Function not found getRegistry
ERROR:ContractSolcParsing:Impossible to generate IR for Tally.init
(contracts/maci/Tally.sol#132-145):
 'NoneType' object has no attribute 'type'
INFO:Detectors:
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-return-in-a
ssembly
INFO:Detectors:
Poll.isInit (node_modules/maci-contracts/contracts/Poll.sol#17) is never
initialized. It is used in:
Poll.numMessages (node_modules/maci-contracts/contracts/Poll.sol#50) is never
initialized. It is used in:
       - Poll.numSignUpsAndMessages()
(node_modules/maci-contracts/contracts/Poll.sol#258-261)
Tally.tallyResults (node_modules/maci-contracts/contracts/Tally.sol#64) is never
initialized. It is used in:
       - Tally.claim(IPayoutStrategy.Claim) (contracts/maci/Tally.sol#209-242)
       - Tally.getAllocatedAmount(uint256,uint256)
(contracts/maci/Tally.sol#266-279)
Tally.totalTallyResults (node_modules/maci-contracts/contracts/Tally.sol#67) is
never initialized. It is used in:
       - Tally.addTallyResults(ITally.AddTallyResultsArgs)
(contracts/maci/Tally.sol#167-181)
       - Tally.calculateAlpha(uint256) (contracts/maci/Tally.sol#284-300)
Tally.totalSpent (node_modules/maci-contracts/contracts/Tally.sol#70) is never
initialized. It is used in:
       - Tally.calculateAlpha(uint256) (contracts/maci/Tally.sol#284-300)
Tally.token (contracts/maci/Tally.sol#27) is never initialized. It is used in:
       - Tally.deposit(uint256) (contracts/maci/Tally.sol#148-152)
       - Tally.withdraw() (contracts/maci/Tally.sol#155-159)
       - Tally.totalAmount() (contracts/maci/Tally.sol#162-164)
       - Tally.claim(IPayoutStrategy.Claim) (contracts/maci/Tally.sol#209-242)
Tally.registry (contracts/maci/Tally.sol#30) is never initialized. It is used in:
       - Tally.addTallyResults(ITally.AddTallyResultsArgs)
(contracts/maci/Tally.sol#167-181)
       - Tally.claim(IPayoutStrategy.Claim) (contracts/maci/Tally.sol#209-242)
Tally.maxCap (contracts/maci/Tally.sol#33) is never initialized. It is used in:
       - Tally.getAllocatedAmount(uint256,uint256)
(contracts/maci/Tally.sol#266-279)
Tally.voiceCreditFactor (contracts/maci/Tally.sol#36) is never initialized. It is
used in:
       - Tally.getAllocatedAmount(uint256,uint256)
(contracts/maci/Tally.sol#266-279)
       - Tally.calculateAlpha(uint256) (contracts/maci/Tally.sol#284-300)
Tally.cooldown (contracts/maci/Tally.sol#39) is never initialized. It is used in:
Tally.custodian (contracts/maci/Tally.sol#42) is never initialized. It is used in:
       - Tally.withdraw() (contracts/maci/Tally.sol#155-159)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-v
ariables



INFO:Detectors:
2 different versions of Solidity are used:
- Version constraint ^0.8.20 is used by:
- Version constraint ^0.8.10 is used by:
```

```
INFO:Detectors:
Tally.addTallyResult(uint256,uint256,uint256[][],uint256,uint256,uint256,uint8)
(contracts/maci/Tally.sol#184-206) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code


Tally.cooldown (contracts/maci/Tally.sol#39) should be constant
Tally.custodian (contracts/maci/Tally.sol#42) should be constant
Tally.maxCap (contracts/maci/Tally.sol#33) should be constant
Tally.registry (contracts/maci/Tally.sol#30) should be constant
Tally.token (contracts/maci/Tally.sol#27) should be constant
Tally.totalSpent (node_modules/maci-contracts/contracts/Tally.sol#70) should be
constant
Tally.totalTallyResults (node_modules/maci-contracts/contracts/Tally.sol#67) should
be constant
Tally.voiceCreditFactor (contracts/maci/Tally.sol#36) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-
could-be-declared-constant
```