HashCloak

Code Review and Security Assessment
For
MACI V3

Initial Report: February 20th, 2025
Updated Report: March 3rd, 2025
Updated Review: March 17th, 2025

Prepared For:
John Guilding | *Ethereum Foundation*
Nicolas Serrano | *Ethereum Foundation*

Prepared by:
Alex Yu-Chen Liao | *HashCloak Inc.*
Manish Kumar | *HashCloak Inc.*

Table Of Contents

Executive Summary

The Ethereum Foundation's Privacy Ethereum team engaged *HashCloak Inc.* to perform a code review and security audit for MACI v3, an Ethereum application that provides privacy and collusion resistance for on-chain voting. The security audit was done from January 12th to February 20th with 2 auditors.

Our engagement began with a comprehensive review of MACI's voting workflow and the entities involved at each stage. We then assessed the implementation against known Solidity and Circom vulnerabilities and conducted threat modeling to identify potential risk vectors. Following this, we performed a detailed analysis of the codebase with particular emphasis on ZK circuit constraints and smart contract logic flaws that could enable double voting, vote censorship, or other issues compromising the guarantees of private and collusion-resistant voting. We also employed static analysis tooling for both Circom and Solidity such as Circomspec, Slither and Aderyn to find common vulnerabilities across both Circom circuits and Solidity contracts.

Our review found that the source code is well-documented, with detailed comments that aid readability and comprehension. The codebase also includes extensive tests that cover most of the MACI components, which improves overall reliability and security.

On March 10, we reviewed the following branch and associated commits to ensure that no new issues have been introduced into the codebase. All issues found in the audit have been resolved as of commits:

- Branch:
    - https://github.com/privacy-ethereum/maci/tree/main
    - https://github.com/privacy-ethereum/excubiae/tree/main
- Commits:
    - https://github.com/privacy-ethereum/maci/commit/3a64a2386bff4bcddc7dba5137476921db22959c
    - https://github.com/privacy-ethereum/excubiae/commit/a7640301fcdf805a45a4647bc33045900b31f37d

Overall, we found the issues range from Critical to Informational:

Severity	**Number of Findings**	**Status**
Critical	1	Resolved (MACI-1)
High	3	Resolved (MACI-2, MACI-3, MACI-4)
Medium	5	Resolved (MACI-5, MACI-6, MACI-7, MACI-8, MACI-9)
Low	3	Resolved (MACI-10, MACI-11, MACI-12)
Informational	6	Resolved (MACI-13, MACI-14, MACI-15, MACI-16, MACI-17, MACI-18)

Scope

For the audit, we considered the following repositories:

- https://github.com/privacy-scaling-explorations/maci/ at commit d79aaad297f0721bae2ec7c416dc989442b8d335
- https://github.com/privacy-ethereum/excubiae/ at commit a7c8b6bc0ba998b13920a7e8044bf5fac22efe4b

With following files/folders in scope:

- maci/packages/circuits/
- maci/packages/contracts/
- maci/packages/testing/ts/__tests__/
- excubiae/packages/contracts/contracts/

For the review, we looked into the following repositories:

- https://github.com/privacy-scaling-explorations/maci/ at commit d79aaad297f0721bae2ec7c416dc989442b8d335
- https://github.com/privacy-ethereum/excubiae/ at commit a7640301fcdf805a45a4647bc33045900b31f37d

Overview

MACI is an Ethereum application designed to provide privacy and collusion resistance for on-chain voting. The protocol achieves these guarantees through the use of encryption and zero-knowledge proofs (zk-SNARKs), which conceal individual voting choices while enabling public verification of the final results. Participants begin by registering and joining a poll through the MACI smart contracts, then cast their votes by submitting encrypted messages to the poll contract. Once the voting period concludes, the coordinator processes and tallies the votes, publishing zk-SNARKs proofs to attest to the correctness of both the processing and tallying outcomes.

Since the previous audit, several notable features and architectural changes have been introduced in MACI v3. For the poll contract, each poll now supports custom voice credit allocation and custom gatekeeping mechanisms, enabling greater configurability to suit different use cases. To improve storage efficiency, messages are now stored in a hash chain rather than a merkle tree, and only the public key is stored during signups

instead of the full state leaf. All contracts have been updated to be non-ownable, instead relying on zero-knowledge proofs to ensure that only authorized parties can call specific functions. Additionally, a new full credit voting mode has been added, requiring users to allocate all of their credits to a single option.

Methodology

The audit was conducted through a combination of manual review and static analysis, with findings cross-verified across both approaches. The primary focus was on identifying issues that could compromise the security guarantees of the MACI voting system. We employed multiple static analyzers, including Circomspect for the Circom circuits and Slither for the Solidity contracts, to supplement our manual efforts. During the manual review phase, we first developed a thorough understanding of MACI's overall workflow and analyzed the interactions between each entity. We then systematically assessed the codebase against identified areas of concern and potential attack surfaces.

Overview of Evaluated Components

circuits

- Integer overflow/underflow
- Underconstrained circuits
- Nondeterministic Circuits
- Assigned but not Constrained circuits
- Constant Branching (if-else, loops)
- amount invariant
- Group and subgroup check of signature and public keys
- Signature scalar validation check
- Correctness of Message processing
- Private to public key generation
- Signature verification correctness
- Root calculation of binary merkle tree
- Merkle tree inclusion proof circuit
- Hashing correctness
- Correctness of converting MACI messages to a command

- Deterministic construction and use of nullifiers
- Correct use of `Num2Bits(n)` and `LessThan` circuits from Circomlib
- Use and constrained check of public signals
- Unused Output Signals

Contracts

- Implementation of cryptography components
- Verification of users
- Merkle root validation
- Potential double voting issues
- Correct usage of the proxy pattern
- Verification of the Circom proofs
- Look for any dead code
- Any attack that can impact user votes
- Any undefined behavior
- Any transaction replay or reentrancy possibility
- Verification key storage
- Correctness of tallying and processing the votes

Gatekeeping contracts

- Potential double voting issues
- Correctness of enforcing each policy
- Nullifiers management of each policy
- Correct usage of the proxy pattern
- Correct interface for each policy

Tests

- Test structure analysis
- Assertion completeness
- State management before and after tests cases
- Protocol logic verification
- Data validation
- Error checks
- Cryptographic checks such as proof generation and verification

Findings

MACI-1: Users can register and vote multiple times with a single Semaphore identity

Type: Critical

Files affected:

- excubiae/packages/contracts/contracts/extensions/semaphore/SemaphorePolicy.sol

Description: In the Semaphore policy, the scope is set to `scope = (address||groupId)`, and the nullifier is `Hash([scope, secret])` in the Semaphore circuit where secret is the Semaphore user secret key. A map is set up in the Semaphore policy contract to keep track of the used nullifier. However, a user can generate multiple addresses and create multiple valid proofs with the same Semaphore secret key.
For example, let the Semaphore secret key of Alice be `secret_alice`. She then generates three distinct Eth addresses `addr_1`, `addr_2` and `addr_3`. She can create three valid but distinct proofs with the same secret since the nullifier will be `Hash([addr_1||groupId, secret_alice])`, `Hash([addr_2||groupId, secret_alice])` and `Hash([addr_3||groupId, secret_alice])`.

Impact: Users can register and vote multiple times with a single Semaphore key for polls and MACI contracts using the Semaphore policy.

Recommendation: Change the scope to `groupID` as suggested in the Semaphore doc, so one semaphore key can only join once. Additionally, to prevent an adversary from front-running pending semaphore proofs to use in their own transaction, include the prover's address in the message field of a proof, so each proof is bound to a specific address. The updated check in `SemaphoreChecker.sol` will be

```
// Extract the subject's address
```

```
address _prover = address(uint160(proof.message));

if (_scope != groupId) {
    revert InvalidGroup();
}

if (_prover != subject) {
    revert InvalidProver();
}
```

Status: The MACI team has updated the scope and message field in a proof as recommended in this commit: [769d134a44b71a590afc1d2b99952166f4b425db]

MACI-2: Incorrect subgroup check for Public Key in the `EdDSAPoseidonVerifier` template

Type: High

Files affected:

- maci/packages/circuits/circom/utils/EdDSAPoseidonVerifier.circom

Description: The `EdDSAPoseidonVerifier` function implements a variant of eddsa from circomlib, as noted in the code comment. However, In the MACI circuits, the elliptic curve subgroup check is performed by applying point doubling operations to the Public Key as shown below:

```
var (computedDouble1XOut, computedDouble1YOut) =
BabyDbl()(publicKeyX, publicKeyY);
    var (computedDouble2XOut, computedDouble2YOut) =
BabyDbl()(computedDouble1XOut, computedDouble1YOut);
    var (computedDouble3XOut, computedDouble3YOut) =
BabyDbl()(computedDouble2XOut, computedDouble2YOut);
```

But the actual check is applied on the original Public Key(`publicKeyX`) instead of the derived value $8 * PublicKey$(`computedDouble3XOut`) as shown below:

```
var computedIsAxZero = IsZero()(publicKeyX);
var computedIsAxEqual = IsEqual()([computedIsAxZero, 0]);
```

The reference implementation performs this subgroup check on the correctly derived value (see: https://github.com/iden3/circomlib/blob/master/circuits/eddsa.circom#L105).

Impact: If a public key point is in a small order subgroup, this can make the signature equation hold trivially for multiple scalars letting invalid signatures verify as valid. This can result in a malicious user submitting fake MACI messages/votes that the circuit accepts as valid even though it was not produced by the real key owner.

Recommendation: Perform the check on the derived public key (8*PublicKey) instead of original as shown below:

```
var computedIsAxZero = IsZero()(computedDouble3XOut);
var computedIsAxEqual = IsEqual()([computedIsAxZero, 0]);
```

Status: The team fixed the issue in the commit: 8b65e9567589a61337c47d22e86c41e58059a091

MACI-3: Missing group validation check for Public Key and Signature points in the `EdDSAPoseidonVerifier` template

Type: High

Files affected:

- maci/packages/circuits/circom/utils/EdDSAPoseidonVerifier.circom

Description: The template `EdDSAPoseidonVerifier` takes input the public key and signature directly as signal inputs in their x and y components as shown below:

```
// The x and y coordinates of the public key.
    signal input publicKeyX;
    signal input publicKeyY;
    ⋮
// The x and y coordinates of the signature point.
    signal input signaturePointX;
    signal input signaturePointY;
```

But the signal input indeed is a point on the curve that is not checked. In the reference circomlib eddsa implementation, the Bits2Point_Strict converts bits to a point and also checks whether the point lies on the curve or not. But since in MACI circuits, the points are directly taken as inputs their group validation is not done and must be done by calling the `BabyCheck()` template.

Impact: Missing group validation check can result into malformed/forged MACI messages/votes to be accepted by the coordinator as the signature equation may bypass the verification with forged values

Recommendation: Add a circuit constraint that the signal inputs of Public Key and signature indeed lie on the curve by calling the `BabyCheck()` template. The check would be like:

```
include "./babyjub.circom";

template EdDSAPoseidonVerifier() {
    ...
    // The x and y coordinates of the public key.
    signal input publicKeyX;
    signal input publicKeyY;
    ...
    // The x and y coordinates of the signature point.
    signal input signaturePointX;
    signal input signaturePointY;
```

```
    BabyCheck()(publicKeyX, publicKeyY);
    BabyCheck()(signaturePointX, signaturePointY);
    ...
}
```

Status: The team has added the check in this commit: [88ff5eda8ba4193c35811a475642795fcb587604]

MACI-4: Using enforced attestations as nullifiers in the EAS Policy may allow multiple signups or votes for a user

Type: High

Files affected:

- excubiae/packages/contracts/contracts/extensions/eas/EASPolicy.sol

Description: In `EASPolicy.sol`, it keeps track of the attestations that have already been enforced as nullifiers. However, if a user can generate multiple attestations for the same scheme, or if an attestation is expired and reissued during the voting period. A user will be able to register or vote in the same MACI or poll contracts multiple times. Since the checker also requires the recipient of an attestation to be the user, we suggest tracking the enforced accounts as nullifiers instead of used attestations.

Impact: Depending on the type of attestation, if the issuer of the attestation allows users to generate multiple attestations for the same scheme, or if the attestation can be reissued after expiration or revoked, a user will be able to register or vote for more than one time.

Recommendation: Change the nullifier from a map of enforced attestations to a map of enforced accounts in the EAS policy contract. The updated contract will be:

```
contract EASPolicy is BasePolicy {
   // a mapping of addresses that have already enforced
```

```
    mapping(address => bool) public enforcedAddresses;
    ...

    function _enforce(address subject, bytes calldata evidence)
internal override {
        // ensure that the user has not been enforced yet
        if (enforcedAddresses[_subject]) {
            revert AlreadyEnforced();
        }

        enforcedAddresses[_subject] = true;

        super._enforce(subject, evidence);
    }
    ...
}
```

Status: The MACI team has changed the nullifier from enforced attestations to enforced accounts in this commit: [274d950f50a3810fa830fead31f7c41267805c25]

MACI-5: Unconstrained signal input in template `StateLeafAndBallotTransformer` for all three voting type(full, qv, non-qv)

Type: Medium

Files affected:

- maci/packages/circuits/circom/utils/full/StateLeafAndBallotTransformer.circom
- maci/packages/circuits/circom/utils/non-qv/StateLeafAndBallotTransformer.circom
- maci/packages/circuits/circom/utils/qv/StateLeafAndBallotTransformer.circom

Description: In the template `StateLeafAndBallotTransformerFull`, `StateLeafAndBallotTransformerNonQv` and `StateLeafAndBallotTransformer`, the input signals `commandPollId` and `commandSalt` is declared but is not used anywhere. As the input signal is unconstrained, they are effectively free variables that don't get bound to the Message processing proof at all. That means the prover can change it without affecting verification, which defeats the purpose of binding a proof to a specific poll and salt. To prevent a proof to be reused, it is required to bind the `pollId` and salt with the proof. Hence the signals must be constrained in a way that it cannot be changed by anyone. This can be done by applying a dummy square to force the compiler to add a constraint.

Impact: The proof is not bound to a specific poll and may be used across different polls.

Recommendation: The `commandSalt` and `commandPollId` can be constrained using the dummy square method as shown in the Tornado Nova and Semaphore circuit below to bound it to the proof. The `commandPollId` can be made a public signal instead of private.

- https://github.com/semaphore-protocol/semaphore/blob/main/packages/circuits/src/semaphore.circom#L74
- https://github.com/tornadocash/tornado-nova/blob/master/circuits/transaction.circom#L124

Status: The MACI team has added the dummy square to enforce the compiler to add a constraint in this commit: f932e8335526597bca3824e6b9b4664deca64940

MACI-6: Converting `signatureScalar` using `Num2Bit` is under-constrained

Type: Medium

Files affected:

- maci/packages/circuits/circom/utils/EdDSAPoseidonVerifier.circom

Description: In template `EdDSAPoseidonVerifier`, the signature scalar is converted into its binary representation using `Num2Bits(254)`. Converting `signatureScalar` using `Num2Bit` can potentially result in aliasing issues if the value of the scalar can be >= p. In other words, this can result in more than one representation of the scalar satisfying the circuit. Therefore It is recommended to use `Num2Bits_strict` for converting into binary representation so that only one signatureScalar satisfies the circuit.

Impact: More than one signature scalar can satisfy the circuit which can result in passing the verification of forged signatures thus, an invalid MACI message may pass the circuit verification.

Recommendation: Use `Num2Bits_strict` for converting signature scalar into its binary representation like below:

```
var computedNum2Bits[254] = Num2Bits_strict()(signatureScalar);
```

Status: The team fixed the issue in commit: [380a696d0c30817ec4e1e2cb0fdbdcb23fa2b893]

MACI-7: Missing group check for the user public key in `joinPoll()`

Type: Medium

Files affected:

- maci/packages/contracts/contracts/Poll.sol

Description: In the function `joinPoll()`, the `_publicKey` parameter is used as a public input of the proof. It is recommended to also validate whether the public key

point is in the correct subgroup in the contract instead of only relying on the check in the circuit.

Impact: An invalid public key point will be viewed as a valid input when joining a poll in the Solidity contract. Depending on the implementation of the subsequent circuit, an invalid public key point can result in malformed MACI messages to be accepted by the coordinator.

Recommendation: Add `CurveBabyJubJub.isOnCurve(x,y)` check for the `_publicKey` in the function.

Status: The MACI team has added the `inOnCurve()` check in the `joinPoll()` function in this commit: [ea0a98ac503a00b6be48b407b8359090883b3381]

MACI-8: Missing `expirationTime` check for the EAS attestation

Type: Medium

Files affected:

- excubiae/packages/contracts/contracts/extensions/eas/EASChecker.sol

Description: In the function `_check()` where the attestation of a user is verified, there's no check against the expiration time of the attestation.

Impact: Users with an expired attestation will still be accepted during the verification and be able to register and cast a vote.

Recommendation: Add a check for the `expirationTime` of an attestation as below:

```
if (
    attestation.expirationTime > 0 &&
    attestation.expirationTime <= block.timestamp
){
```

```
    revert AttestationExpired(attestation.expirationTime);
}
```

Status: The MACI team has added the expiration time check as suggested in this commit: [22439c5767c8eb904b07e55ca614d14177d36f5d]

MACI-9: Incorrect input for the `verifyProof()` function in the Semaphore checker contract

Type: Medium

Files affected:

- excubiae/packages/contracts/contracts/extensions/semaphore/SemaphoreChecker.sol
- excubiae/packages/contracts/contracts/test/extensions/mocks/SemaphoreMock.sol

Description: In the function `_check` of the SemaphoreChecker contract, it passes the `_scope` as the first input for the `verifyProof` function. However, in the `ISemaphore.sol` contract interface and in the official Semaphore contract, the first input should be the `groupId`. The same issue can be found in the `SemaphoreMock.sol` contract for testing where `verifyProof()` takes the scope as the first input and retrieves the `groupId` from it.

Impact: The Semaphore checker contracts may not be compatible with contracts implementing the official Semaphore interface.

Recommendation: Replace `_scope` with `groupId` as the first input for the `verifyProof` function, and update the `verifyProof` function in the `SemaphoreMock.sol` contract to accept `groupId` as the first input.

Status: The MACI team has updated the first input from `scope` to `groupID` for the `verifyProof()` function in this commit: [b86d2a22dd0d293b55d1a1f34070af6f9c11d0dc]

MACI-10: Linked Poseidon hash libraries in `MACI.sol` is only verified against zero

Type: Low

Files affected:

- maci/packages/contracts/contracts/MACI.sol

Description: In the constructor of `MACI.sol`, the linked Posiedon libraries are only verified by checking the result of `hash2([uint256(1), uint256(1)])` is non-zero instead of checking whether it returns the correct hashing result.

Impact: Incorrect Poseidon libraries will still pass the verification as long as it doesn't return zero for the `hash2()` function.

Recommendation: Compare the result of `hash2([uint256(1), uint256(1)])` with the correct Poseidon hash result instead of zero.

Status: The MACI team has updated the linked library check as recommended in this commit: [23ae8ed87ece93f637df1b29dbba0fcfa0aac7cb]

MACI-11: Verify `voteOptionTreeDepth` to prevent out-of-bounds array access for `emptyBallotRoots`

Type: Low

Files affected:

- maci/packages/contracts/contracts/MACI.sol

Description: In the function `deployPoll()`, the `emptyBallotRoot` for the `DeployPollArgs` is retrieved by getting `emptyBallotRoots[args.treeDepths.voteOptionTreeDepth - 1]`. It is advised to check the value of `voteOptionTreeDepth` to prevent out-of-bounds issues.

Impact: An uncaught array out-of-bounds issue may occur when deploying new polls.

Recommendation: Add checks like following:

```
uint8 depth = args.treeDepths.voteOptionTreeDepth;
if (depth == 0 || depth > emptyBallotRoots.length) {
  revert InvalidVoteOptionTreeDepth(depth);
}
```

Status: The MACI team has added a check for the vote-option tree depth in this commit: [2695df1b7ff39ed595c858a81b0f21959eeff4bf]

MACI-12: Same accounts can register and vote unlimited times in the `FreeForAll` policy

Type: Low

Files affected:

- excubiae/packages/contracts/contracts/extensions/freeForAll/FreeForAllPolicy.sol

Description: In `FreeForAllPolicy`, there is no tracking of the already enforced accounts. Although it wouldn't mitigate sybil attacks entirely, we suggest to still add an enforced accounts check like in other policies, so it can fulfill the requirement of MACI that a user can only register or vote once.

Impact: A user can register and vote multiple times with the free-for-all policy.

Recommendation: Add a check for the already enforced accounts.

Status: The MACI team has added a check for the enforced accounts for `FreeForAllPolicy` in this commit: dd6cc224d2d19a1389ea41a941b7d1813992dcc5

MACI-13: Unused errors declared in the Solidity contracts

Type: Informational

Files affected:

- maci/packages/contracts/contracts/MessageProcessor.sol
- maci/packages/contracts/contracts/Poll.sol
- maci/packages/contracts/contracts/Tally.sol
- maci/packages/contracts/contracts/trees/LazyIMT.sol

Description: Several errors are unused in the contracts.

- MessageProcessor.sol:

```
error StateNotMerged();
error totalSignupsTooLarge();
error CurrentMessageBatchIndexTooLarge();
error BatchEndIndexTooLarge();
```

- Poll.sol:

```
error PollAlreadyInit();
error TooManyMessages();
error BatchHashesAlreadyPadded();
```

- Tally.sol:

```
error totalSignupsTooLarge();
error BatchStartIndexTooLarge();
error TallyBatchSizeTooLarge();
```

- LazyIMT.sol

```
error DepthCannotBeZero();
error AmbiguousDepth();
```

Recommendation: Removed the unused errors in the contracts.

Status: The MACI team has removed the unused errors listed above in this commit: [2dfb84b44898e2f7356294bbee69a3931d64d58c]

MACI-14: Missing `IPolicyFactory` inheritance for the policy factories

Type: Informational

Files affected:

- excubiae/packages/contracts/contracts/extensions/anonAadhaar/AnonAadhaarCheckerFactory.sol
- excubiae/packages/contracts/contracts/extensions/eas/EASPolicyFactory.sol
- excubiae/packages/contracts/contracts/extensions/erc20/ERC20PolicyFactory.sol
- excubiae/packages/contracts/contracts/extensions/erc20votes/ERC20VotesPolicyFactory.sol
- excubiae/packages/contracts/contracts/extensions/freeForAll/FreeForAllPolicyFactory.sol
- excubiae/packages/contracts/contracts/extensions/gitcoin/GitcoinPassportPolicyFactory.sol
- excubiae/packages/contracts/contracts/extensions/hats/HatsPolicyFactory.sol
- excubiae/packages/contracts/contracts/extensions/merkle/MerkleProofPolicyFactory.sol
- excubiae/packages/contracts/contracts/extensions/semaphore/SemaphorePolicyFactory.sol
- excubiae/packages/contracts/contracts/extensions/token/TokenPolicyFactory.sol

- excubiae/packages/contracts/contracts/extensions/zupass/ZupassPolicyFactory.sol

Description: The policy factory contracts should inherit the `IPolicyFactory` interface.

Recommendation: Add the inheritance of `IPolicyFactory` for each policy factory contract.

Status: The MACI team has added inheritance to `IPolicyFactory` for the contracts listed above in this commit: [ffda1cd0774d7d2f225ed18fb8cc8c810c97c61c]

MACI-15: Multiple variables assigned but never read.

Type: Informational

Files affected:
- packages/circuits/circom/coordinator/non-qv/SingleMessageProcessor.circom
- packages/circuits/circom/coordinator/qv/SingleMessageProcessor.circom
- packages/circuits/circom/coordinator/full/SingleMessageProcessor.circom
- packages/circuits/circom/coordinator/non-qv/MessageProcessor.circom
- packages/circuits/circom/coordinator/qv/MessageProcessor.circom
- packages/circuits/circom/coordinator/full/MessageProcessor.circom

Description: Multiple variables are defined in the above mentioned files such as BALLOT_NONCE_INDEX, BALLOT_VOTE_OPTION_ROOT_INDEX, STATE_LEAF_PUBLIC_X_INDEX, STATE_LEAF_PUBLIC_Y_INDEX, STATE_LEAF_VOICE_CREDIT_BALANCE_INDEX, MESSAGE_TREE_ZERO_VALUE, maxVoteOptions, NUMBER_BITS and MESSAGE_LENGTH but are never read in those files. In other files the same variables are assigned as well as used so these definitions seem redundant.

Recommendation: Remove those variables if not needed to avoid any confusion.

Status: The team removed unused variables in this commit: [cb8ce92e53ee9daae8e635cf8f716458f4aa3ac6]

MACI-16: Missing `await` on rejection assertions

Type: Informational

Files affected:

- packages/testing/ts/__tests__/unit/setVerifyingKeys.test.ts

Description: The test case:

```
expect(
  setVerifyingKeys({ ... })
).rejectedWith("This poll joining verifying key is already set in the contract");
```

returns a promise but it is returned without awaiting. This can pass the test even if a promise is never rejected. Adding `await` makes the test pass correctly, which means the logic was sound—the test was just written incorrectly ie, `await` was missing.

Recommendation: Add `await` on rejection assertions.

Status: The MACI team added await in the commit: [780162793bc56b3383cc5bb2b433a93dd238438e]

MACI-17: Wrong `stateIndex` used for messages in testing

Type: Informational

Files affected:

- packages/testing/ts/__tests__/e2e.test.ts

Description: In the test case: `it("should publish 30 messages", async () => {...});`(line 888) the `stateIndex` for users published msg is hardcoded to 1n for ALL 30 users but it should be distinct for each user.

Recommendation: Change the `stateIndex` for each user to be distinct.

Status: The state index was corrected in commit: [062565e99807243a908b7c18896ce3e7999f0c46]

MACI-18: Incorrect comments for `_index` in function `getPublicJoinedCircuitInputs()` and `verifyJoinedPollProof()`

Type: Informational

Files affected:

- packages/contracts/contracts/Poll.sol

Description: The doc comments for the `_index` in `getPublicJoinedCircuitInputs()` and `verifyJoinedPollProof()` should be the index of Poll's `pollStateRootsOnJoin` when joined instead of MACI's `stateRootOnSignUp`.

Recommendation: Update the comment for both functions.

Status: The MACI team has updated the comments as suggested in this commit: [b40e37b94b919b2bff12b96c2771f9d4b4ff2ac8]

References

- [Circomspect: Circom static analysis tool]
- [Slither, the smart contract static analyzer]
- [Aderyn]

Appendix

- **Output of circumspect static analyzer (Clipped)**

```
circomspect: analyzing template 'PrivateToPublicKey'
warning: The value assigned to `isLessThan` is not used in witness or constraint
generation.
    ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/utils/PrivateToPubli
cKey.circom:29:5
    │
29 │     var isLessThan = LessThan(251)([privateKey, SUBGROUP_ORDER]);
    │     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ The value
assigned to `isLessThan` here does not influence witness or constraint generation.
    │
    = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#side-ef
fect-free-assignment.

warning: Using `Num2Bits` to convert field elements to bits may lead to aliasing
issues.
     ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/coordinator/non-qv/S
```

```
ingleMessageProcessor.circom:127:56
    |
127 |     var computedStateLeafPathIndices[stateTreeDepth] =
Num2Bits(stateTreeDepth)(stateIndexMux);
    |
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Circomlib template `Num2Bits` instantiated
here.
    |
    = Consider using `Num2Bits_strict` if the input size may be >= than the prime
size.
    = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#non-str
ict-binary-conversion.

warning: The signal `commandSalt` is not used by the template.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/utils/non-qv/StateLe
afAndBallotTransformer.circom:49:5
   |
49 |     signal input commandSalt;
   |     ^^^^^^^^^^^^^^^^^^^^^^^^ This signal is unused and could be removed.
   |
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unused-
variable-or-parameter.

warning: The signal `commandPollId` is not used by the template.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/utils/non-qv/StateLe
afAndBallotTransformer.circom:47:5
   |
47 |     signal input commandPollId;
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^ This signal is unused and could be removed.
   |
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unused-
variable-or-parameter.

warning: The signal `commandPollId` is not used by the template.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/utils/qv/StateLeafAn
dBallotTransformer.circom:47:5
   |
47 |     signal input commandPollId;
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^ This signal is unused and could be removed.
```

```
   │
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unused-
variable-or-parameter.

warning: The signal `commandSalt` is not used by the template.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/utils/qv/StateLeafAn
dBallotTransformer.circom:49:5
   │
49 │     signal input commandSalt;
   │     ^^^^^^^^^^^^^^^^^^^^^^^^ This signal is unused and could be removed.
   │
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unused-
variable-or-parameter.

warning: Using `Num2Bits` to convert field elements to bits may lead to aliasing
issues.
    ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/coordinator/full/Sin
gleMessageProcessor.circom:131:56
    │
131 │     var computedStateLeafPathIndices[stateTreeDepth] =
Num2Bits(stateTreeDepth)(stateIndexMux);
    │
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Circomlib template `Num2Bits` instantiated
here.
    │
    = Consider using `Num2Bits_strict` if the input size may be >= than the prime
size.
    = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#non-str
ict-binary-conversion.

warning: The variable `MESSAGE_LENGTH` is assigned a value, but this value is never
read.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/coordinator/qv/Singl
eMessageProcessor.circom:25:5
   │
25 │     var MESSAGE_LENGTH = 10;
   │     ^^^^^^^^^^^^^^^^^^^^^^^ The value assigned to `MESSAGE_LENGTH` here is
never read.
   │
```

```
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unused-
variable-or-parameter.

warning: The variable `NUMBER_BITS` is assigned a value, but this value is never
read.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/coordinator/qv/Singl
eMessageProcessor.circom:39:5
   │
39 │     var NUMBER_BITS = 252;
   │     ^^^^^^^^^^^^^^^^^^^^^ The value assigned to `NUMBER_BITS` here is never
read.
   │
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unused-
variable-or-parameter.


warning: Using `Num2Bits` to convert field elements to bits may lead to aliasing
issues.
    ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/coordinator/qv/Singl
eMessageProcessor.circom:131:56
    │
131 │     var computedStateLeafPathIndices[stateTreeDepth] =
Num2Bits(stateTreeDepth)(stateIndexMux);
    │
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Circomlib template `Num2Bits` instantiated
here.
    │
    = Consider using `Num2Bits_strict` if the input size may be >= than the prime
size.
    = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#non-str
ict-binary-conversion.


warning: Using `Num2Bits` to convert field elements to bits may lead to aliasing
issues.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/utils/EdDSAPoseidonV
erifier.circom:43:33
   │
43 │     var computedNum2Bits[254] = Num2Bits(254)(signatureScalar);
```

```
   │                                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Circomlib
template `Num2Bits` instantiated here.
   │
   = Consider using `Num2Bits_strict` if the input size may be >= than the prime
size.
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#non-str
ict-binary-conversion.


warning: The signal `commandSalt` is not used by the template.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/utils/full/StateLeaf
AndBallotTransformer.circom:49:5
   │
49 │     signal input commandSalt;
   │     ^^^^^^^^^^^^^^^^^^^^^^^^ This signal is unused and could be removed.
   │
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unused-
variable-or-parameter.


warning: The signal `commandPollId` is not used by the template.
   ┌─
/Users/manishspc/HashCloak/audit/maci/packages/circuits/circom/utils/full/StateLeaf
AndBallotTransformer.circom:47:5
   │
47 │     signal input commandPollId;
   │     ^^^^^^^^^^^^^^^^^^^^^^^^^^ This signal is unused and could be removed.
   │
   = For more details, see
https://github.com/trailofbits/circomspect/blob/main/doc/analysis_passes.md#unused-
variable-or-parameter.
```

Some of the results were found to be false positives.