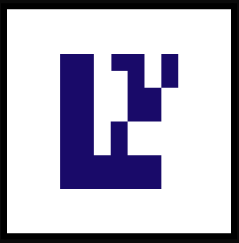
Security Assessment & Formal Verification Draft Report



EigenLayer M4

July 2024

Prepared for Eigenlayer

certora

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Eigenlayer	https://github.com/Layr-Labs/eigenlayer-contracts/pull/515	f65a310	EVM

Project Overview

This document describes the specification and verification of Eigenlayer using the Certora Prover and manual code review findings. The work was undertaken from July 9th 2024 to august 12th 2024

The following contract list is included in our scope:

```
pods/EigenPodManager.sol
pods/EigenPod.sol
libraries/BeaconChainProofs.sol
libraries/Markle.sol
libraries/Edian.sol
```

The Certora Prover demonstrated that the implementation of the **Solidity** contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts**.** During the verification process and the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Please note that a few more formal rules are not included in this report, as they were proven with an unreleased version of the Certora Prover. Once those rules are proven on a released version of the Certora Prover, we will add them to the next version of this document.

Protocol Overview

The scope of this audit is the Eigen pod, pod manager and supporting libraries. The pod manages the various validators under a user, and updates the pod manager with the number of shares to be allocated for each user based on the amount of assets locked up in the pod, or in the stakes that are registered to that pod. The pod manager manages the pods, and supplies the interface between each pod and the delegation manager. As of writing of this document, the slashing and rewarding logic have yet been implemented and so the main focus of the audit is of sound logic, and adherence to the beacon specs.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	1	-	-
Medium	3	-	-
Low	5	-	-
Informational	6	-	-
Total	15		

Severity Matrix

Impact		Likelihood		
	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High

certora

Detailed Findings

ID	Title	Severity	Status
H-01	Electra upgrade removes risk from unactivated validators.	High	Not yet fixed
M-01	Zero amount not reverting in addShares and removeShare.	Medium	Not yet fixed
M-02	Unsafe Merkle library in the case of trees that are not a power of two.	Medium	Not yet fixed
M-03	Penalties can't be synced to the pod, leading to overestimation of shares.	Medium	Not yet fixed
L-01	The protocol might rely on stale information during withdrawal, leading to withdrawal of slashed funds	Low	Not yet fixed
L-02	Attackers can make startCheckpoint(revertIfNoBalance =true) pass by sending 1 Gwei to the pod.	Low	Not yet fixed
L-03	Malicious front run verifyStaleBalance can stop verifyWithdrawalCredentials.	Low	Not yet fixed
L-04	podManager API should check that the pod has been initiated.	Low	Not yet fixed
L-05	Over estimation of newly added validators.	Low	Not yet fixed

High Severity Issues

H-01 Electra upgrade removes risk from unactivated validators		
Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: EigenPod.sol	Status: Not Fixed	Violated Property: Not applicable.

Description: During the upgrade to Electra unactivated validators (validators that never reached 32 ETH) have their balances fully transferred to the pending_balance_deposits array. The EigenPod is not aware that the assets are locked there, and mistakenly thinks that these assets are **gone.** This allows users to dodge slashing, and have risk free rewards.

It also allows third parties the option to force some validators to be considered withdrawn.

Exploit Scenario:

1. Eve sets up before the fork to Electra (or up to 8191 slots after) a bunch of validators with 31 ETH
2. After the fork, all Eve's validators report 0 balance, as well as the eigenPod reporting 0 balance.
3. Until a checkpoint is made, (by Eve) or until she chooses to deposit the extra ETH into her validators she is immune from Slashing.
4. Eve uses her immunity to heavily "risk" her assets getting major rewards. If slashing occurs the pod would report Eve as having no assets, and the slashing would fail.

Recommendations: Check the balances in pending_balance_deposits and calculate them as part of the assets locked within the pod.

Customer's response:

Fix Review:

Medium Severity Issues

M-01 Zero amount not reverting in addShares and removeShare.

Severity: **Medium**	Impact: **Medium**	Likelihood: **Low**
Files: EigenPodManager.sol	Status: Not Fixed	Violated Property: **P-16**

Description: AddShares and removeShares do not revert for zero amount calls. This might cause the delegation manager to Queue up too much when calling depositIntoStrategy and causing a DOS for the user as they won't be able to iterate over this Queue.

This bug is also applicable to **withdrawSharesAsTokens** however in that case, cursory examination did not find any immediate problems.

Note: Exact exploit scenario was not fully investigated as it was out of scope and in a code still in development.

Exploit Scenario: Out of scope.

Recommendations: Revert on Zero amounts in those API

Customer's response:

Fix Review:

M-02 Unsafe Merkle library in the case of trees that are not a power of two.

Severity: **Medium**	Impact: **Medium**	Likelihood: **very low**
Files: Merkle.sol	Status: Not Fixed	Violated Property: **P-21**

Description: The Merkle library has some unsafe practices that are out of spec. This might not be applicable for EigenPod currently, however when handling library code, due to its portability it is advised to sanitize the inputs so that the library cannot cause faults for integrations, here is a list of suggested fixes:

processInclusionProofKeccak:

- Index should be 0 before returning from the method.

processInclusionProofSha256:

- Index should be 0 before returning from the method.

merkleizeSha256:

- leaves.length should be a power of two, and greater than 1 (note that there is a dev comment that does not include 1 is a bad length).

Recommendations: Implement the hardening fixes in order to avoid potential problems in the future.

When gas is a constraint, write “unsafe” versions of these methods (unsafeProcessInclusionProofKeccak, unsafeProcessInclusionProofSha256, ect.) with documented fault modes to be checked by the caller.

Customer’s response:

Fix Review:

M-03 Penalties can't be synced to the pod, leading to overestimation of shares.

Severity: **Med**	Impact: **Med**	Likelihood: **Med**
Files: EigenPod.sol	Status: Not Fixed	Violated Property: Not applicable.

Description: Validator's balance can decrease due to inactivity/attestation penalties. Unlike slashing, the pod doesn't have a mechanism to detect those and allow anybody to start a new checkpoint for that. So that decrease can go unnoticed, leading to the owner holding more shares than they should.
The extra shares lead to 2 types of impact:

- The owner would receive more rewards than they should
- The owner might use this to avoid slashing (up to the same amount of the overestimation, see L-01)

The decrease can be pretty slow while the Beacon chain is active enough, but in case of an inactivity leak this can be as fast as a 16 ETH decrease over 3 weeks

Exploit Scenario:

- Bob owns a pod with validator A registered to it
- Bob doesn't maintain their validator
- A severe inactivity leak begins on the beacon chain, as a result Bob's validator loses 16 ETH over 3 weeks
    - The balance of the validator is now reduced from 32 ETH to 16 ETH
- Bob keeps receiving rewards on the EigenLayer as if they hold 32 ETH, double than what they're eligible for
- Bob can also use this to avoid slashing up to 16 ETH, as mentioned above

Recommendations: There's no easy way to mitigate this one as there's no easy way to distinguish between a decrease due to penalties vs a decrease due to withdrawal.
One possible way to address this would be to allow anybody to create a new checkpoint if a very long time passed since the last checkpoint.

Customer's response:

Fix Review:

Low Severity Issues

L-01 The protocol might rely on stale information during withdrawal, leading to withdrawal of slashed funds

Severity: **Low**	Impact: **Med**	Likelihood: **Low**
Files: EigenPodManager.sol EigenPod.sol	Status: Not Fixed	

Description: The podOwnerShares in podManager and the actual assets controlled by a pod are synced during a checkpoint. If a checkpoint is old, beacon chain slashing or penalties can cause this values to drift, causing

- podOwnerShares > pod assets in the case of penalties.
- podOwnerShares < pod assets in the case of rewards.

This drift must be taken into account when a pod manager issues some requests from the Pod, to make sure both their information is in sync.

During withdrawSharesAsTokens EigenPodManager doesn't verify that the accounting of shares is up to date.
A possible impact of this issue is withdrawal of slashed shares. In case of overestimation (e.g. due to slashing or penalties) of the shares that the owner holds, this might allow an owner with slashed shares to withdraw some of those shares.

Exploit Scenario:

- Bob owns a pod with 2 validators
  - validators A and B, with a balance of 32 ETH each. A total of 64 ETH shares
- Bob's operator on EigenLayer misbehaves, and as a result Bob is slashed by 32 ETH

    - Bob now holds 32 ETH of shares (the original 64, minus 32 after slashing)
    - Validator A and B still hold 32 ETH each.
- Bob initiates a full withdrawal for validator A
- Once the withdrawal is finalized and the funds are in the pod, Bob creates a checkpoint
- Bob initiates the withdrawal of 32 ETH from the pod via the Delegation Manager
- Right before the withdrawal the pod is complete validator B is slashed on the Beacon chain
    - Bob now should hold only 31 ETH of shares (32 from previous step, minus 1 ETH due to slashing)
- Before anybody calls `verifyStaleBalance()` to update the pod about the slashing, Bob withdraws the full 32 ETH of shares from the pod
    - Bob was able to withdraw one more ETH than they should have (32 ETH when they own only 31 ETH of shares)
    - Also note that calling `verifyStaleBalance()` alone isn't sufficient to stop Bob from withdrawing that extra 1 ETH, you'd also have to finalize the checkpoint by supplying the proof for validator B

Recommendations:

- Make sure the protocol relies on fresh data when executing a withdrawal (i.e. requiring a new checkpoint)
- Alternatively, make sure that any event of decrease can be immediately synced to the pod (e.g. penalties, see M-03)
- In case a decrease is proved (e.g. for slashing - when `verifyStaleBalance()` is called), consider halting withdrawals or holding back some funds till the decrease is fully registered to the pod (in case of slashing - the checkpoint is finalized)
- Consider adding a mechanism to burn the slashed funds by sending them to a dead address, as soon as they made available in the pod

Customer's response:

Fix Review:

L-02 Attacker can make `startCheckpoint(revertIfNoBalance=true)` pass by sending 1 Gwei to the pod.

Severity: **Low**	Impact: **Low**	Likelihood: **Med**
Files: EigenPod.sol	Status: Not Fixed	

Description: The `startCheckpoint()` function has a parameter named `revertIfNoBalance`, when this is set to true the function would revert if there's no new balance in the pod.
An attacker can send 1 Gwei to the pod to make this check pass even when there's no new balance.

```
        // If the caller doesn't want a "0 balance" checkpoint, revert
        if (revertIfNoBalance && podBalanceGwei == 0) {
            revert("EigenPod._startCheckpoint: no balance available to
checkpoint");
        }
```

Exploit Scenario:

- Bob is expecting 3 ETH that are about to be sent to their pod
- Bob calls `startCheckpoint()` with `revertIfNoBalance=true`, expecting the call to revert if the 3 ETH didn't reach the pod yet
- Alice front runs this tx, sending 1 Gwei to the pod (1 Gwei is worth much less than 1 cent)

- The checkpoint started at an unfavorable time for Bob, before receiving the expected 3 ETH
- Bob now has to finalize the started checkpoint before they can start a new one to record those 3 ETH as shares

Recommendations: Allow the user to specify a minimum new ETH for a new checkpoint, if the new balance is any less than that amount - revert.

Customer's response:

Fix Review:

L-03 Malicious front run verifyStaleBalance can stop verifyWithdrawalCredentials.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: EigenPod.sol	Status: Not Fixed	

Description: A front run call to verifyStaleBalance can update currentCheckpointTimestamp to the current block.timestamp, causing verifyWithdrawalCredentials to revert.

Exploit Scenario:

1. Alice calls verifyWithdrawalCredentials with the intent to add new validators.
2. Eve front runs verifyStaleBalance.
3. Alice's call fails.

Note: Even though not applicable to this bug, it is recommended that beaconTimestamp would always be greater than or equal to lastCheckpointTimestamp.

Recommendations: one possible fix can be:

1. Change the require to beaconTimestamp >= currentCheckpointTimestamp it should still stop validators from participating from the current checkpoint.

Customer's response:

Fix Review:

L-04 podManager API should check that the pod has been initiated.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: EigenPodManager.sol	Status: Not Fixed	Violated Property: **P-20**

Description: some API calls handle pods without checking that they have been initiated properly. This could have far reaching effects when changing delegation manager or those APi's. uninitiated pods should not have their shares modified in any way.

API is that don't check that the pod has been initialized:

- addShare
- removeShares
- withdrawSharesAsTokens

Note: Exact exploit scenario was not fully investigated as it was out of scope and in a code still in development.

Exploit Scenario: Out of scope.

Recommendations: Use the hasPod API to validate the input.

Customer's response:

Fix Review:

L-05 Over estimation of newly added validators.		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: EigenPod.sol	Status: Not Fixed	Violated Property: Not applicable.

Description: Newly verified validators use their potentially old and updated balance when receiving shares, in the last 8191 slots things like:

- Penalties.
- Hysteresis.
- Not yet implemented EIP-7251 custom ceiling for partial withdrawals.

Exploit Scenario:

- Alice has an actual balance that is lower then the effective balance.
- She verifies that the validator receives extra rewards based on the effective balance as opposed to the current balance.

Recommendations: Fetch the actual balances during verification, or dont mint extra shares until a fresh checkpoint is made.

Customer's response:

Fix Review:

Informational Severity Issues

I-01. `_gap` variable shrink is incorrect

Description: The EigenPod contract has a `_gap` variable at the end of it, that gap variable is reduced from 44 in M2 to 37 in M4.
This reduction is incorrect since the `_gap` variable is shifted only by 4 slots in M4 (due to new variables introduced in M4).
This can be seen by comparing the storage layout of M2 (`_gap` variable is at slot 58 here, run `forge inspect EigenPod storage` to get the storage layout) with the layout of M4 (now it's at slot 62).

This error has no impact, since this `_gap` variable is the last storage variable in the final deployed contract (there are currently no contracts that inherit `EigenPod`, it's the last one in the chain of inheritance) so this shift doesn't display any storage variables.

```
{
  "astId": 64096,
  "contract": "src/contracts/pods/EigenPod.sol:EigenPod",
  "label": "__gap",
  "offset": 0,
  "slot": "58",
  "type": "t_array(t_uint256)44_storage"
}
```

Recommendation: Set the `_gap` variable to 40 rather than 37.

Customer's response:

Fix Review:

I-02. EIP 6914 suggests reusing the validator index, which would break the core functionality of the Pod

Description: The pod works on the assumption that the validator index is fixed to the validator and the same index would never be associated with a different validator.
However, it's worth noting that EIP 6914 which was suggested in April 2023 recommends reusing validators' indexes after they've exited the Beacon chain.
If this were ever implemented, it would break the core functionality of the pod. A user would be able to verify withdrawal credentials, exit the validator, and once the index is reused by another validator - create a new checkpoint. The pod would assume the withdrawal credentials still point to the pod and credit the balance as shares, while this would not be the case.

Note: Currently not applicable (As of August 2024 this EIP is under the 'Stagnant' status).

Recommendation: Monitor EIP 6914 for new activity. If it's being implemented - make the necessary code changes to address this issue (e.g. verify withdrawal credentials periodically, every SAFE_EPOCHS_TO_REUSE_INDEX seconds).

Customer's response:

Fix Review:

I-03. activeValidatorCount can overflow without revert

Description: activeValidatorCount can overflow over the uint24 limit without reverting causing potential catastrophic consequences. This attack currently is impractical due to the assets required to trigger it.

Recommendation: Add a require to _verifyWithdrawalCredentials or change the time of activeValidatorCount to trigger the solidity overflow revert.

Customer's response:

Fix Review:

I-04. Unused imports at `EigenPod.sol`

Description: The following imports aren't used in `EigenPod.sol` and can be removed (note that instead of `AddressUpgradeable` the regular `Address` library is used in the code, this is implicitly imported via `SafeERC20`)

```
JavaScript
import "@openzeppelin-upgrades/contracts/access/OwnableUpgradeable.sol";
import "@openzeppelin-upgrades/contracts/utils/AddressUpgradeable.sol";
import "@openzeppelin-upgrades/contracts/utils/math/MathUpgradeable.sol";
```

Recommendation: Remove those imports

Customer's response:

Fix Review:

I-05. Document the unused inheritance of `ReentrancyGuardUpgradeable`

Description: `EigenPod` inherits `ReentrancyGuardUpgradeable` but doesn't use it. This is done in order to preserve the storage layout from M2.
It's best to document this unused inheritance so it won't be removed by mistake.

Recommendation: Add a comment

Customer's response:

Fix Review:

I-06. Check for solidity compiler bugs before deployment

Description: The project is using solidity version `^0.8.12`, which means solidity version `0.8.12` or above can be used to compile the files before deployment.
It's best to check if there are any known solidity compiler bugs for the version that's about to be used, and see if any of them is relevant for the code.

Recommendation: Before deployment, check for solidity compiler bugs for the specific compiler version that's about to be used.

Customer's response:

Fix Review:

Formal Verification

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

General Assumptions and Simplifications

- We work with objects inherited from the original contracts. In the inherited objects we add more view methods, flags, etc. These modifications don't affect the functionality of original contracts and the verification results hold also for the original contracts.
- We unroll loops for a fixed number of iterations. The exact number of iterations is mentioned for each contract.

Formal Verification Properties

EigenPod

Module General Assumptions

- We verified the contract functions against an arbitrary storage state.
- We work with scenarios where there are at most two EigenPods linked to the same EigenPodManager.
- Loops are assumed to iterate at most 3 times.
- We assume that calls don't revert due to overflows, e.g. `activeValidatorCount` is less than 2^24, `podBalanceGwei` is less than 2^64, etc.

Module Properties

P-01. activeValidatorCount is correctly calculated

Status: Verified

Rule Name	Status	Description	Link to rule report
activeValidatorsCount_correctness	Verified	*Verifies that `activeValidatorCount` always equals to the number of validators that are active*	*Report*

P-02. balanceUpdateTimestamp never decreases

Status: Verified

Rule Name	Status	Description	Link to rule report
mostRecentBalanceUpdateTimestampOnlyIncreases2	Verified	*`Validator.lastCheckpointedAt` can never decrease.*	*Report*

P-03. checkpoint timestamp is set if and only if the checkpoint fields are set

Status: Verified

Rule Name	Status	Description	Link to rule report
checkpointInfolsEmpty	Verified	*`CurrentCheckpointTimestapm` is non zero if and only if `currentCheckpoint.beaconBlockRoot`, `proofsRemaining`, `podBalanceGwei` and `balanceDeltasGwei` are non zero.*	*Report*

P-04. checkpoint timestamp is correct

Status: Verified

Rule Name	Status	Description	Link to rule report
checkpointsTimestampRemainsCorrect	Verified	*If `currentCheckpointTimestamp` is nonzero then `lastCheckpointTImestamp < currentCheckpointTimestamp`*	*Report*

P-05. During the checkpoint the checkpoint fields are correctly updated

Status: Verified

Rule Name	Status	Description	Link to rule report
beaconBlockRootDoesntChangeInChP	Verified	*During a checkpoint the* `currentCheckpoint.beaconBlockRoot` *doesn't change*	*Report*
podBalanceGweiDoesntChangeInChP	Verified	*During a checkpoint the* `currentCheckpoint.podBalanceGwei` *doesn't change*	*Report*
proofsRemainingCannotIncreaseInChP	Verified	*During a checkpoint the* `currentCheckpoint.proofsRemaining` *doesn't increase*	*Report*

P-06. last checkpoint timestamp can never decrease

Status: Verified

Rule Name	Status	Description	Link to rule report
lastCheckpointTSOnlyIncreases	Verified	`lastCheckpointTImestamp` *can never decrease*	*Report*

P-07. Inactive validators have empty info

Status: Verified

Rule Name	Status	Description	Link to rule report
inactiveValidatorsHaveEmptyInfo	Verified	*If a validator is `INACTIVE`, then `validator.validatorIndex`, `restakedBalanceGwei` and `lastTimestampedAt` are all zero.*	*Report*

P-08. Active validators have lastTimestampedAt correctly set

Status: Verified

Rule Name	Status	Description	Link to rule report
lastCheckpointedEqualsLastChPTS	Verified	*If a validator is `ACTIVE` then `validator.lastTimestampedAt == lastCheckpointTimestamp`. (Except during a checkpoint.)*	*Report*
lastCheckpointedNoGreaterThanLastTimestamp	Verified	`validator.lastTimestampedAt <= MAX[ lastCheckpointTimestamp, currentCheckpointTimestamp]`	*Report*

P-09. ValidatorIndex is set only once

Status: Verified

Rule Name	Status	Description	Link to rule report
validatorIndexSetOnlyOnce	Verified	`validator.validatorIndex` *is set once and then never changes.*	*Report*

P-10. ValidatorStatus correct state transitions

Status: Verified

Rule Name	Status	Description	Link to rule report

validatorStatusTransitionsCorrect	Verified	*Validator status can only change as follows:*
		`INACTIVE -> INACTIVE | ACTIVE`
		`ACTIVE -> ACTIVE | WITHDRAWN`
		`WITHDRAWN -> WITHDRAWN`
			Report

P-11. Withdrawn validators have zero balance

Status: Verified

Rule Name	Status	Description	Link to rule report
withdrawnValidatorsHaveZeroRestakedGwei	Verified	*Validator.status == WITHDRAWN then validator.restakedBalanceGwei == 0*	*Report*

EigenPodManager

Module General Assumptions

- We verified the contract functions against an arbitrary storage state.
- We work with scenarios where there are at most two EigenPods linked to the same EigenPodManager.
- Loops are assumed to iterate at most 3 times.
- We assume that calls don't revert due to overflows, e.g. `activeValidatorCount` is less than 2^24, `podBalanceGwei` is less than 2^64, etc.

Module Properties

P-12. Integrity of methods

Status: Verified

Rule Name	Status	Description	Link to rule report
addShares_integrity	Verified	*Verifies that `addShares` updates the storage correctly.*	*Report*
removeShares_integrity	Verified	*Verifies that `removeShares` updates the storage correctly.*	*Report*

P-13. Public methods are additive

Status: Verified

Rule Name	Status	Description	Link to rule report
addShares_additivity	Verified	`addShares(x); addShares(y);` *has the same effect as* `addShares(x+y)`	Report
removeShares_additivity	Verified	`removeShares(x); removeShares(y);` *has the same effect as* `removeShares(x+y)`	Report
withdrawShares_additivity	Verified	`withdrawSharesAsTokens(x); withdrawSharesAsTokens(y);` *has the same effect as* `withdrawSharesAsTokens(x+y)`	Report

P-14. Public methods are independent

Status: Verified

Rule Name	Status	Description	Link to rule report
addShares_independence	Verified	`addShares(x); addShares(y);` *has the same effect as* `addShares(y); addShares(x);`	Report
add_remove_independence	Verified	`addShares(x); removeShares(y);` *has the same effect as* `removeShares(y); addShares(x);`	Report

P-15. podAddress is set only once

Status: Verified

Rule Name	Status	Description	Link to rule report
podAddressNeverChanges	Verified	`ownerToPod[podOwner]` *is set once and never changes*	*Report*

P-16. Methods revert when called with zero shares

Status: Violated

Rule Name	Status	Description	Link to rule report
addShares_reverts	Violated	`addShares(0)` *reverts.*	*Report*
removeShares_reverts	Violated	`removeShares(0)` *reverts.*	*Report*

P-17. podOwner shares are whole Gwei amount

Status: Verified

Rule Name	Status	Description	Link to rule report
podOwnerSharesAlwaysWholeGweiAmount	Verified	`podOwnerShares[podOwner]` *is never a non-whole Gwei amount*	*Report*

P-18. Limitation on negative shares

Status: Verified

Rule Name	Status	Description	Link to rule report
limitationOnNegativeShares	Verified	`podOwnerShares[podOwner]` *can become negative only as a result of a call to* `recordBeaconChainETHBalanceUpdate`	*Report*

P-19. addShares and removeShares are inverse

Status: Verified

Rule Name	Status	Description	Link to rule report
add_remove_inverse	Verified	*Performing* `addShares(x);` `removeShares(x);` *has no effect on the storage*	*Report*

P-20. addShares and removeShares revert when the podOwner doesn't have a pod

Status: Violated

Rule Name	Status	Description	Link to rule report
addShares_revertsWhenNoPod	Verified	`addShares(owner, x)` *reverts when* `owner` *doesn't have a pod.*	*Report*
removeShares_revertsWhenNoPod	Violated	`removeShares(owner, x)` *reverts when* `owner` *doesn't have a pod.*	*Report*

Merkle

Module General Assumptions

- We verified the contract functions against an arbitrary storage state.
- We verify the contract as a stand-alone one, i.e. we make no assumptions about the caller. We assume all methods may be called with arbitrary arguments.
- The method `merkleizeSha256` is only called with arrays whose length is a power of two, as the specifications require.
- Loops are assumed to iterate at most 10 times.

Module Properties

P-21. merkleizeSha256 doesn't collide

Status: Violated

Rule Name	Status	Description	Link to rule report
merkleizeSha256IsInjective	Violated	*Verifies that `merkleizeSha256` on two different inputs will not produce the same output*	*Report*
merkleizeSha256IsInjective_onSameLengths	Verified	*Verifies that `merkleSha256` on two different inputs of the same length will not produce the same output*	*Report*

P-22. processInclusionProofKeccak works correctly

Status: Verified

Rule Name	Status	Description	Link to rule report
processInclusionProofKeccak_correctness	Verified	*Changing only the `leaf` argument changes the return value of `processInclusionProofSha256`*	*Report*

Endian

Module General Assumptions

- We verified the contract functions against an arbitrary storage state.
- We verify the contract as a stand-alone one, i.e. we make no assumptions about the caller. We assume all methods may be called with arbitrary arguments.
- Loops are assumed to iterate at most 4 times.

Module Properties

P-23. fromLittleEndianUint64 works correctly

Status: Verified

Rule Name	Status	Description	Link to rule report
fromLittleEndianUint64_correctness	Violated	*Verifies that all bytes in the output are exactly as they are supposed.*	Report
transformationsAreInverse1	Verified	*Verifies that methods* `fromLittleEndianUint64` *and* `toLittleEndianUint64` *are inverse, i.e.,* `toLittleEndianUint64(fromLittleEndianUint64(x)) == x` *for all x where* `x << 64 == 0`	Report
transformationsAreInverse2	Verified	*Verifies that methods* `fromLittleEndianUint64` *and* `toLittleEndianUint64` *are inverse, i.e.,* `fromLittleEndianUint64(toLittleEndianUint64(x)) == x` *for all x*	Report

BeaconChainProofs

Module General Assumptions

- We verified the contract functions against an arbitrary storage state.
- We verify the contract as a stand-alone one, i.e. we make no assumptions about the caller. We assume all methods may be called with arbitrary arguments.
- Loops are assumed to iterate at most 24 times.

Module Properties

P-24. verifyValidatorBalance works correctly

Status: Verified

Rule Name	Status	Description	Link to rule report
verifyValidatorBalance_balanceRootUnique	Violated	*Verifies that all bytes in the output are exactly as they are supposed.*	*Report*

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.