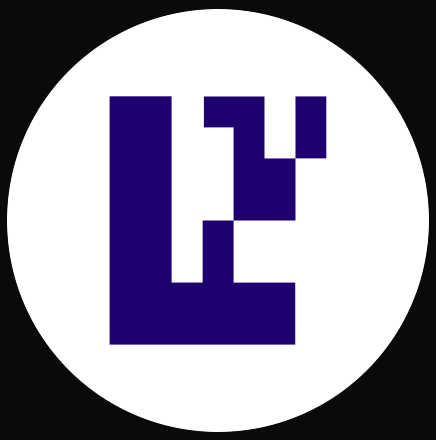
Security Assessment & Formal Verification *Final* Report



EigenLayer Protocol

Jan 2025

Prepared for EigenLayer

certora

Table of contents

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Slashing Magnitudes	EigenLayer		EVM

Project Overview

This document describes the specification and verification of EigenLayer Slashing using the Certora Prover. The work was undertaken from **January 10th 2025** to **February 11th 2025.**

The following contract list is included in our scope:

src/contracts/core/AllocationManager.sol

src/contracts/core/AllocationManagerStorage.sol

src/contracts/core/DelegationManager.sol

src/contracts/core/DelegationManagerStorage.sol

src/contracts/core/StrategyManager.sol

src/contracts/core/StrategyManagerStorage.sol

src/contracts/interfaces/*

src/contracts/libraries/SlashingLib.sol

src/contracts/libraries/Snapshots.sol

src/contracts/pods/*

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.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	1	1	1
High	1	1	1
Medium	2	2	2
Low	5	3	2
Informational	4	2	-
Total	11	9	6

Severity Matrix

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

certora

Detailed Findings

ID	Title	Severity	Status
C-01	Underflow when calculating new encumbered magnitude leads to over-risk.	Critical	Fixed.
H-01	The protocol is susceptible to reentrancy attacks.	High	Partially fixed.
M-01	_clearDeallocationQueue might not clear stale allocations.	Medium	Fixed.
M-02	Over increasing delegations when adding depositShares from negative to positive.	Medium	Fixed.
L-01	Delegation manager can add strategies that are not whitelisted.	Low	Acknowledged Wont Fix.
L-02	removeStrategiesFromOperatorSet and addStrategiesFromOperatorSet should probably have some sort of timelock.	Low	Acknowledged Wont Fix.
L-03	OperatorSet decode should be unique.	Low	Awaiting Fix.
L-04	burnShares should only work for valid strategies.	Low	Awaiting Fix.
L-05	modifyAllocation should only work for valid operators.	Low	Disregarded.

Critical Severity Issues

C-01 Underflow when calculating new encumbered magnitude leads to over-risk.

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: AllocationManager.sol	Status: Fixed.	Property violated: P-04. Magnitudes Solvency

Description: In modifyAllocation, we use unsafe addition when calculating the new encumbered magnitude.

We calculate the pendingDiff of one allocation to a new magnitude,

```
allocation.pendingDiff = _calcDelta(allocation.currentMagnitude, params⬆[i].newMagnitudes[j]);
```

Then we add that diff to the previous encumberedMagnitude

```
239        info.encumberedMagnitude = _addInt128(info.encumberedMagnitude, allocation.pendingDiff);
```

This calculation uses the _addInt128,

```
function _addInt128(uint64 a⬆, int128 b⬆) internal pure returns (uint64) {
    return uint64(uint128(int128(uint128(a⬆)) + b⬆));
}
```

which is unsafe, the pendingDiff can be as high as we wish but for this example let's say is **max uint64 - 1**.

Due to the way Solidity truncates when downcasting, this would result in decreasing the encumberedMagnitue, and queueing up pending allocation with a huge number, later when we get the next updated allocation we would add that number allowing us to decrease and increase the encumberedMagnitude even past the maxMagnitude

```
allocation.currentMagnitude = _addInt128(allocation.currentMagnitude, allocation.pendingDiff);

// If the completed change was a deallocation, update used magnitude
if (allocation.pendingDiff < 0) {
    info.encumberedMagnitude = _addInt128(info.encumberedMagnitude, allocation.pendingDiff);
}
```

Recommendations: It would be recommended to sanitize the values of newMagnitudesnewMagnitudes as well as moving to safe casting instead of the current implementation of the _addInt128.

stomer's response: Acknowledged.

Fix Review: Properly Fixed In this PR. We would recommend in addition to the fix to add a check that each newMagnitude is less than or equal to WAD, This would make the code even more future proof.

High Severity Issues

H-01 The protocol is susceptible to reentrancy attacks.		
Severity: **High**	Impact: **High**	Likelihood: **Very Low**
Files: DelegationManager.sol	Status: Partially fixed.	

Note: We have not found a full exploit for this bug, The example here shows one vector, however, due to the complexity of bugs of this sort, and the simplicity of fixing them, we still hold the severity as High Although the impact is Medium.

Description: When withdrawing funds from strategies, such as in _completeQueuedWithdrawal we can break the flow due to the strategies using erc77 other hookable underlying assets. In this case, the protocol uses possibly stale information when calculating the prevSlashingFactors and newOwner which could lead to a weird reentrancy guard when completing withdrawals. This can lead to stale information regarding shares and deposits.to major issues if future updates allow each strategy to specify a different withdrawal method.

Recommendations: To fix this issue multiple approaches can be taken.

1. Change _completeQueuedWithdrawal to follow the Check Effect Interaction pattern by recalculating the stale values of each loop.
2. Implementing a multi-contract cross-reentrancy This can be done by implementing a view method that exports the lock state, and changing that state from the external contracts.
3. Implementing an external contract that would be shared by all contracts to make sure that all contracts share the same lock.

Customer's response: Acknowledged.

Fix Review: Partially Fixed in this PR. This issue is at the current state of the code Theoretical, however changes to the code might introduce this bug in the future. The current fix of adding reentrency guard on all external methods, doesn't quite cover all vectors, as the DelegationManager calls other contracts that are not protected by the same reentrency guard.

We would recommend considering adding a cross-contract Reentrency guard in a future update. And consider recalculating each external value in the loop (using more gas but maintain check effect interaction pattern better.

_getSlashingFactors and _getSlashingFactorsAtBlock are the methods that call external contracts.

Medium Severity Issues

M-01 _clearDeallocationQueue might not clear stale allocations.

Severity: **Medium**	Impact: **Medium**	Likelihood: **Low**
Files: AllocationManager.sol	Status: Fixed.	Property violated: P-06. Deallocations effect blocks are in ascending order P-07. DeallocationQueue allocations uniqueness

Description: When removing allocations from the deallocations queue, it is assumed that the allocations in the queue move from most stale to least stale,

however, because the updated allocation is fetched

```
(StrategyInfo memory info, Allocation memory allocation) =
    _getUpdatedAllocation(operator⬆, operatorSetKey, strategy⬆);
```

The allocation might hold an effectBlock that is later than some of the other allocations in this queue, this happens because when we modifyAllocations we check only the pendingDiff

```
require(allocation.pendingDiff == 0, ModificationAlreadyPending());
```

If we were slashed then the pendingDiff might have been rounded to 0, allowing us to modify an allocation without removing and adding it back to the queue.

```
if (allocation.pendingDiff < 0) {
    uint64 slashedPending =
        uint64(uint256(uint128(-allocation.pendingDiff)).mulWadRoundUp(params⬆.wadsToSlash[i]));
    allocation.pendingDiff += int128(uint128(slashedPending));
```

Recommendations: It is recommended that before adding to the deallocation queue, we check the effectBlock and not the pendingDiff.

Customer's response: Acknowledged.

Fix Review: Properly fixed in this PR.

M-02 Over increasing delegations when adding depositShares from negative to positive.

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: EigenPodManager.sol	Status: Fixed.	Property violated: P-05. Add Shares Integrity

Description: EigenPodManager _addShares uses the wrong value when returning the addedShares.

```
return (prevDepositShares < 0 ? 0 : uint256(prevDepositShares), shares⬆);
```

If we transition from negative to positive we would return 0 instead of prevDepositShares but we return the full shares (although we truncated the negative addition)

Recommendations: In the case where prevDepositShares is negative instead of returning shares, return updatedDepositShares

Customer's response: Acknowledged, also found internally during the audit, fix is already waiting for review.

Fix Review: Properly fixed in this PR.

Low Severity Issues

L-01 Delegation manager can add strategies that are not whitelisted.		
Severity: **Low**	Impact: **Low**	Likelihood: **High**
Files: DelegationManager.sol	Status: Acknowledged Wont Fix.	

Description: Delegation Manager when completing a queued withdrawals might add a strategy that was whitelisted and later removed, this is because addShares does not check if the strategy is whitelisted and so adds it to the list of strategies. Allowing the user to user Illegal strategy

Recommendations: Check that the strategy is not whitelisted before allowing the shares to be added.

Customer's response: We agree that the DelegationManager can bypass the whitelist like this, but the whitelist feature was originally intended to serve more as limitation on deposits in the StrategyManager. If users have already deposited via the StrategyManager, we don't want to prevent withdrawals (even if the strategy has been removed from the whitelist).

Fix Review: Not applicable.

L-02 removeStrategiesFromOperatorSet and addStrategiesFromOperatorSet should probably have some sort of timelock.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: AllocationManager.sol	Status: Acknowledged Wont Fix..	

Description: addStrategiesFromOperatorSet and removeStrategiesFromOperatorSet should have some sort of timelock to make sure that operators don't instantly become slashable/unslashable for a specific strategy.

Recommendations: Add a time lock to those operations to allow avss and operators time to respond to such changes

Customer's response: We're aware of this, but feel adding yet another queue/delay mechanism to the AllocationManager is more trouble than it's worth. Ultimately, allocating magnitude to a strategy signals consent to be slashed, regardless of the AVS's configuration changes. This is documented as expected behavior in our contract docs here.

Fix Review: Not applicable.

L-03 OperatorSet decode should be unique.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: OperatorSetLib.sol	Status: Awaiting fix.	

Description:

The decode method for operatorSets should always return unique decodings for the ids.

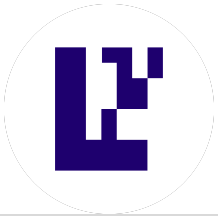
```
function decode(
    bytes32 _key⬆//  160 ->0 96 100000000000000->/  160 ->0 96 000000000000000
) internal pure returns (OperatorSet memory) {
    /// forgefmt: disable-next-item
    return OperatorSet({
        avs: address(uint160(uint256(_key⬆) >> 96)),
        id: uint32(uint256(_key⬆) & type(uint96).max)
    });
}
```

In the current implementation, we see that id is truncated to uint32 which would allow multiple decodings

Recommendations: add an if or requirement that the key's lower 96 bytes is less than max uint32.

Customer's response: Acknowledged.

Fix Review: Awaiting fix.



L-04 burnShares should only work for valid strategies.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: StrategyManager.sol	Status: Awaiting fix.	

Description: burnShares should only work for valid strategies, it is unadvised to call an unverified external contract.

Recommendations: check that the strategy is valid by checking if sharesToBurn != 0, This has the side benefit of saving some gas on edge cases where this function is called by mistake.

Customer's response: Acknowledged.

Fix Review: Properly fixed in this PR.

L-05 modifyAllocation should only work for valid operators.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: AllocationManager.sol	Status: Disregarded.	

Description: When calling modifyAllocation allow only valid operators to proceed.

Recommendations: require(delegation.isOperator(operator), InvalidOperator());

Customer's response: Operators are assumed to be valid because modifyAllocations require an operator to have an allocation delay. Setting an allocation delay requires one to be an operator.

Fix Review: Not applicable.

Informational Issues

I-01. Enforce only valid configuration delay

Description: ALLOCATION_CONFIGURATION_DELAY should always be greater than MIN_WITHDRAWAL_DELAY_BLOCKS to avoid slashing during the upgrade.

Recommendation: ALLOCATION_CONFIGURATION_DELAY should be defined as a configurable number **Plus** MIN_WITHDRAWAL_DELAY_BLOCKS to ensure it is always greater than MIN_WITHDRAWAL_DELAY_BLOCKS

Customer's response: Awaiting customer response.

Fix Review: Awaiting customer response.

I-02. Delegation manager might fail to add shares if the strategy list is full

Description: When removing a share if the total drops to 0 it is removed from the queue, this would mean that if a user wishes to withdraw it back as a share, if their strategy manager holds the maximum number of strategies they would revert.

Recommendation: don't pop strategies automatically, instead allow a user to call and pop strategies that they have that have 0 shares in them to have better control of the flow of their strategies and avoid surprise reverts.

Customer's response: Awaiting customer response.

Fix Review: Awaiting customer response.

I-03. Increasingly decreasing accuracy in all factor calculations.

Description: When calculating the new factors:

1. Magnitudes.
2. Deposit scaling factor.
3. Beacon scaling factor.

Those values are strictly decreasing (aside from DSF which is strictly increasing) at high slash rations we get closer to 0 losing a lot of our accuracy bits.

Recommendation: Currently we would recommend to document this feature of the system and consider some design changes that would allow those values to reset to the high accuracy values at some point.

Customer's response: Acknowledged would add documentation.

Fix Review: Awaiting Fix review.

I-04. Minor wrong comment in IAllocationManager.sol

Description: We describe effectBlock in an inaccurate way.

```
75      * @param effectBlock The block number after which a pending delay will take effect
76      */
```

In the Interface we describe the effect block as the block After, however In the code we allow for the effectBlock that is equal to the block.timestemp. It's the difference between:

1. block.timestep **>** allocation.effectBlock - **Documentation.**
2. block.timestep **>=** allocation.effectBlock - **Code.**

Recommendation: Update the documentation or the code.

Customer's response: Acknowledged.

Fix Review: Awaiting Fix review.

certora

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

1. We used Solidity Compiler version 8.27 to verify the protocol, using `--optimize 1` and `--via-ir`
2. We substitute Openzeppelin's `mulDiv(uint256,uint256,uint256)` with a mathematical encoding of `mulDiv` which reverts if an intermediary or final result does not fit in `uint256`.
3. We assume that the output of OperatorSetLib.key(OperatorSetLib.OperatorSet memory os) is consistent, never returns 0 and for different input values the output is different.

Formal Verification Properties

Allocation Manager

Module General Assumptions

Module Properties

P-01. maxMagnitude is monotonically decreasing

Status: Verified	

Rule Name	Status	Description	Link to rule report
maxMagnitudeMonotonicallyDecreasing	Verified	*maxMagnitude is monotonically decreasing*	*Report*

P-02. MagnitudeHistory Keys are valid

Status: Verified	Assumptions:

Rule Name	Status	Description	Link to rule report
maxMagnitudeHistoryKeysMonotonicInc	Verified	*_maxMagnitudeHistory.keys are monotonic increasing - MagnitudeHistory Keys are valid - Snapshots' timestamps are sorted in increasing order.*	*Report*
maxMagnitudeHistoryKeysLessThanCurrentBlock	Verified	*Ensures all keys in `maxMagnitudeHistoryKeys` are less than or equal to the current block number.*	*Report*

P-03. Consistency between registered sets and their registration statuses

Status: Verified	Assumptions:

Rule Name	Status	Description	Link to rule report
SetInRegisteredIFFStatusIsTrue	Verified	*All setKeys in registeredSets should have registrationStatus as true per operator and vice versa.*	*Report*

P-04. Magnitudes Solvency

Status: Violated	Assumptions:

Rule Name	Status	Description	Link to rule report
encumberedMagnitudeEqSumOfCurrentMagnitudesAndPositivePending	Violated	*encumberedMagnitude equals the sum of currentMagnitudes and positive pendingDiffs.*	*Report* *Link to issue*
maxMagnitudeGEencumberedMagnitude	Violated	*For each operator and strategy tuple, max magnitude is greater or equal to the encumbered magnitude.*	*Report* *Link to issue*
negativePendingDiffAtMostCurrentMagnitude	Verified	*Cant have a negative pendingDiff that is greater than the current magnitude.*	*Report*

P-05. Pending Diffs valid state transitions

Status: Verified	Assumptions:

Rule Name	Status	Description	Link to rule report
pendingDiffStateTransisions	Verified	*pendingDiff can transition from 0 to negative or positive, from positive zero only, from negative to negative or zero.* *Note: This rule applies to all methods except modifyAllocation because this function makes valid transitions,* *but only at the memory level. first, it loads allocation from memory, makes several valid transitions,* *and then stores it in the storage again, which can lead to an overall violation.*	*Report*
pendingDiffStateTransisionModifyAllocations	Verified	*pendingDiff can transition from 0 to negative or positive, from positive zero only, from negative to negative or zero.* *Note: This rule is for modifyAllocations where when all the effectblock are in the future, this function cannot make more than one transition at a time.*	*Report*

P-06. Deallocations effect blocks are in ascending order

Status: Violated	Assumptions:

Rule Name	Status	Description	Link to rule report
deallocationQueueEffectBlockAscesndingOrder	Violated	*Allocations in the deallocation queue have their effect blocks in ascending order according to their queued order.*	*Report* *Link to issue*

P-07. DeallocationQueue allocations uniqueness

Status: Violated	Assumptions:

Rule Name	Status	Description	Link to rule report
deallocationQueueDataUniqueness	Violated	*At most 1 pending allocation/deallocation can be queued for a given (Operator, OperatorSet, Strategy) tuple*	*Report* *Link to issue*
noZeroKeyInDealocationQ	Verified	*Non-Zero Keys in Deallocation Queue Invariant*	*Report*

P-08. Deallocation Queue Effect Block Timing Bound Invariant

Status: Violated	Assumptions:

Rule Name	Status	Description	Link to rule report
deallocationQueueEffectBlocLessThanCurrBlockNumberPlushDelayPlusOne	Violated	*Allocations in the deallocation queue have their effect blocks equal at most the latest block number + DEALLOCATION_DELAY + 1.*	*Report* *Issue: EGSL-09 Denial Of Service Due To Unbounded Allocation Delay - By Sigma Prime*

Delegation Manager

Module General Assumptions

Module Properties

P-01. CumulativeScaledShares is monotonically increasing

Status: Verified

Rule Name	Status	Description	Link to rule report
cumulativeScaledSharesMonotonicallyIncreasing	Verified	*CumulativeScaledShares is monotonically increasing*	*Report*

P-02. CumulativeScaledSharesHistory Keys are valid

Status: Verified

Rule Name	Status	Description	Link to rule report
cumulativeScaledSharesHistoryKeysMonotonicInc	Verified	*CumulativeScaledSharesHistory Keys are valid - Snapshots' timestamps are sorted in increasing order*	*Report*
CumulativeScaledSharesHistoryKeysLessThanCurrentBlock	Verified	*Ensures all keys in `cumulativeScaledSharesHistoryKeys` are less than or equal to the current block number.*	*Report*

P-03. Operator Cannot Deregister

Status: Verified

Rule Name	Status	Description	Link to rule report
operatorCannotDeregister	Verified	*Operator Cannot Deregister*	*Report*

P-04. Queued Withdrawal Registration Consistency Invariant

Status: Verified

Rule Name	Status	Description	Link to rule report
queuedWithdrawalsCorrect	Verified	*For a given staker, all roots existing in _stakerQueuedWithdrawalRoots also exists in storage: queuedWithdrawals[withdrawalRoot] returns the Withdrawal struct / staker != address(0) pendingWithdrawals[withdrawalRoot] returns True*	*Report*

P-05. Authorized Methods for Deposit Scaling Factor Modification Invariant

Status: Verified

Rule Name	Status	Description	Link to rule report
whoCanChangeDepositScalingFactor	Verified	*depositScalingFactor gets updated for a staker on every added deposit even if the staker is not delegated. Added shares deposits occur through* *1. increaseDelegatedShares called by deposits through the StrategyManager/EigenPodManager* *2. delegateTo* *3. completeQueuedWithdrawals as shares*	*Report*

P-06. Operators delegate to themselves

Status: Verified

Rule Name	Status	Description	Link to rule report
operatorDelegatesToThemselves	*Verified*	*Verifies that operators delegates to themselves.*	*Report*

Module Properties

P-01. Pod owner deposited shares remain positive

Status: Verified	

Rule Name	Status	Description	Link to rule report
podOwnerDepositSharesRemainPositive	*Verified*	*Verifies that podOwnerDepositShares can't go negative, assuming the current balance is a positive int256 value in podOwnerDepositShares.*	*Report*

P-02. BeaconChainSlashingFactor is monotonically decreasing

Status: Verified	

Rule Name	Status	Description	Link to rule report
BeaconChainSlashingFactorMonotonicDec	*Verified*	*BeaconChainSlashingFactor is initialized as WAD and only decreases.*	*Report*

P-03. Pod Address immutability

Status: Verified

Rule Name	Status	Description	Link to rule report
podAddressNeverChanges	*Verified*	*Verifies that ownerToPod[podOwner] is set once (when podOwner deploys a pod), and can otherwise never be updated*	*Report*

P-04. Remove Deposit Shares Integrity

Status: Verified

Rule Name	Status	Description	Link to rule report
integrityOfRemoveDepositShares	*Verified*	*This rule verifies that when a specified amount of deposit shares is removed from a staker's account, the staker's total deposited shares decrease exactly by that amount.*	*Report*

P-05. Add Shares Integrity

Status: Violated

Rule Name	Status	Description	Link to rule report
integrityOfAddShares	*Violated*	*This rule verifies that when shares are added via the addShares function, the staker's deposit shares increase by exactly the specified amount. It also*	*Report* *Link to issue*

		confirms that the function returns the expected number of added shares and that the recorded previous deposit shares match the pre-operation value.	

P-06. Integrity Of Withdraw Shares As Tokens

Status: Verified

Rule Name	Status	Description	Link to rule report
integrityOfWithdrawSharesAsTokens	*Verified*	*This rule ensures correct behavior during token withdrawal. It checks that if a staker's deposited shares are negative before the withdrawal, then afterward they either become zero or are increased by exactly the withdrawn shares. Conversely, if the deposited shares are non-negative, the withdrawal operation does not alter the staker's deposited shares.*	*Report*

P-07. Add-Then-Remove Shares Neutrality

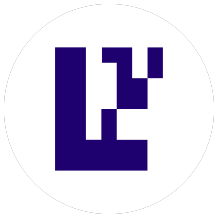
Status: Verified

Rule Name	Status	Description	Link to rule report
addThenRemoveSharesNoEffect	*Verified*	*This rule verifies that if shares are first added and then removed with the same amount, the staker's total deposited shares remain unchanged, confirming that the operations cancel each other out.*	*Report*

P-08. Remove-Then-Add Shares Neutrality

Status: Verified

Rule Name	Status	Description	Link to rule report
removeThenAddSharesNoEffect	*Verified*	*This rule verifies that if shares are removed and subsequently re-added with the same amount, the overall balance of the staker's deposited shares is preserved, ensuring no net effect from the operations.*	*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.