



Security Assessment Report



EigenLayer Hourglass pt1

August 2025

Prepared for EigenLayer team

Table of Contents

Project Summary	3
Project Scope	3
Project Overview	4
Security Considerations	4
Audit Goals	4
Coverage and Conclusions	5
Findings Summary	6
Severity Matrix	6
Detailed Findings	7
Low Severity Issues	8
L-01: TaskMailbox::createTask() may create tasks that cannot be completed	8
L-02: ReleaseManager allows for publishing releases with upgradeByTime equal to the current block	10
L-03: Restrictive check in _validateBN254Certificate()	12
Informational Issues	14
I-01. isValidRelease() and getLatestUpgradeByTime() may panic when no releases exist	14
I-02. Incorrect NatSpec in registerExecutorOperatorSet()	15
I-03. Incorrect diagram in Hourglass framework documentation	16
I-04. Unused imports can be removed	16
Disclaimer	17
About Certora	17

Project Summary

Project Scope

Project Name	Repository (link)	Audited Commits	Platform
EigenLayer Hourglass pt1	https://github.com/Layr-Labs/eigenlayer-contracts https://github.com/Layr-Labs/eigenlayer-middleware/	a77bd0f e5d3622 , 510bdb1	EVM

The following files are included in the audit scope:

Core protocol:

- src/contracts/avs/task/TaskMailbox.sol
- src/contracts/avs/task/TaskMailboxStorage.sol
- src/contracts/core/ReleaseManager.sol
- src/contracts/core/ReleaseManagerStorage.sol
- src/contracts/interfaces/ITaskMailbox.sol
- src/contracts/interfaces/IAVSTaskHook.sol
- src/contracts/interfaces/IReleaseManager.sol
-

Middleware:

- src/avs/task/TaskAVSRegistrarBase.sol
- src/avs/task/TaskAVSRegistrarBaseStorage.sol
- src/interfaces/ITaskAVSRegistrarBase.sol

Project Overview

This document describes the findings of the manual review of **EigenLayer Hourglass pt1**. The work was undertaken from **July 31, 2025** to **August 06, 2025**.

The team manually audited the respective files using code analysis and inspection tools to search for sensitive patterns and assess code structure. During the audit, the Certora team discovered issues in the code, as listed on the following page.

Security Considerations

The purpose of this audit is to investigate three on-chain components in the HourGlass framework used by offchain services to simplify building, deploying, and managing AVS projects: the TaskMailbox, the ReleaseManager, and the TaskAVSRegistrarBase.

During the audit, our team investigated both the direct (e.g., relating to fee split and refunds) and indirect (e.g., DoS attacks, unexpected task configuration changes) attack vectors against these three smart contracts from the perspective of different actors (AVSs, operators, etc.) participating in the EigenLayer ecosystem. We also looked into potential edge cases related to the Task lifecycle state machine and its transition functions.

The audit did not uncover any major issues.

Audit Goals

1. Enumerate the attack surface of the on-chain components of the HourGlass framework.
2. Find specific attack vectors in which attackers could realistically lead to incorrect behavior of said components.
3. Suggest limited and accurate fixes for such attack vectors.
4. Suggest modifications to improve the code's overall security stance.

Coverage and Conclusions

1. The overall quality of the audited code is high.
2. The audited on-chain components have essentially no exposed direct attack surface.
3. We have discovered a few minor issues in the code and the documentation, which can be improved – they are listed below.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	3	3	2
Informational	4	4	3
Total	7	7	5

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
L-01	TaskMailbox::createTask() may create tasks that cannot be completed	Low	Fix confirmed.
L-02	ReleaseManager allows for publishing releases with upgradeByTime equal to the current block	Low	Acknowledged.
L-03	Restrictive check in _validateBN254Certificate()	Low	Fix confirmed.
I-01	isValidRelease() and getLatestUpgradeByTime() may panic when no releases exist	Info	Fix confirmed.
I-02	Incorrect NatSpec in registerExecutorOperatorSet()	Info	Fix confirmed.
I-03	Incorrect diagram in Hourglass framework documentation	Info	Acknowledged.
I-04	Unused imports can be removed	Info	Fix confirmed.

Low Severity Issues

L-01: `TaskMailbox::createTask()` may create tasks that cannot be completed

Severity: **Low**

Impact: **Medium**

Likelihood: **Low**

Files: [TaskMailbox.sol](#)

Status: Fix confirmed.

Description:

`TaskMailbox::createTask()` function enforces that only the latest reference timestamp can be used:

JavaScript

```
// Get the operator table reference timestamp
uint32 operatorTableReferenceTimestamp =
IBaseCertificateVerifier(_getCertificateVerifier(taskConfig.curveType))
    .latestReferenceTimestamp(taskParams.executorOperatorSet);
```

However, the `latestReferenceTimestamp` may already be stale, because there is no validation of whether the `maxStalenessPeriod` from the `CertificateVerifier` has elapsed.

As a result, tasks that can never be verified can be created.

Recommendations:

Consider validating that the data is not stale before the task creation:

JavaScript

```
/// @inheritdoc ITaskMailbox
function createTask(
    TaskParams memory taskParams
```



```
) external nonReentrant returns (bytes32) {  
    ...  
  
    // Get the operator table reference timestamp  
    uint32 operatorTableReferenceTimestamp =  
    IBaseCertificateVerifier(_getCertificateVerifier(taskConfig.curveType))  
        .latestReferenceTimestamp(taskParams.executorOperatorSet);  
  
    +    uint32 maxStaleness =  
    IBaseCertificateVerifier(_getCertificateVerifier(taskConfig.curveType)).maxO  
    peratorTableStaleness(taskParams.executorOperatorSet.key());  
    +    require(maxStaleness == 0 || block.timestamp <=  
    operatorTableReferenceTimestamp + maxStaleness "stale timestamp");
```

Customer's response: Fixed in [6a57826](#).

Fix Review: Fix confirmed.

L-02: **ReleaseManager** allows for publishing releases with **upgradeByTime** equal to the current block

Severity: **Low**

Impact: **Low**

Likelihood: **Low**

Files:
[ReleaseManager.sol](#)

Status: Acknowledged.

Description:

The `publishRelease()` function allows a release to be published with **upgradeByTime** set to the current block timestamp:

JavaScript

```
require(release.upgradeByTime >= block.timestamp, InvalidUpgradeByTime());
```

The **upgradeByTime** value is meant to tell operators by when they should upgrade. While it's only a signal and not enforced on-chain, allowing the deadline to be the same as the current block gives operators no time to upgrade. This can be confusing, especially since the latest release is treated as the valid one.

Requiring the **upgradeByTime** to be in the future (not the current block) would make the signal clearer and more useful.

Recommendations:

Update the require check to enforce that **upgradeByTime** is strictly greater than the current block timestamp:

JavaScript

```
/// @inheritdoc IReleaseManager
```

```
function publishRelease(  
    OperatorSet calldata operatorSet,  
    Release calldata release  
) external checkCanCall(operatorSet.avs) returns (uint256 releaseId) {  
    ...  
-    require(release.upgradeByTime >= block.timestamp,  
InvalidUpgradeByTime());  
+    require(release.upgradeByTime > block.timestamp,  
InvalidUpgradeByTime());  
    ...  
}
```

Customer's response: Acknowledged. This is intended functionality.

L-03: Restrictive check in `_validateBN254Certificate()`

Severity: Low	Impact: Low	Likelihood: Low
Files: TaskMailbox.sol	Status: Fix confirmed.	

Description:

The `_validateBN254Certificate()` function includes the following check on the signature's coordinates:

JavaScript

```
require(cert.signature.X !== 0 && cert.signature.Y !== 0,  
EmptyCertificateSignature());
```

This rejects any signature where either the X or Y coordinate is zero. However, on the BN254 curve, a point with one coordinate equal to zero can still be a valid point, as long as it lies on the curve and is not the point at infinity (0, 0).

This more permissive approach is already reflected in other parts of the codebase (e.g., `KeyRegistrar`), where only the all-zero case is rejected:

JavaScript

```
require(!(g1X == 0 && g1Y == 0), ZeroPubkey());
```

Recommendations:

Update the check to only reject the point at infinity, rather than any zero coordinate:

JavaScript

```
function _validateBN254Certificate(  
  IBN254CertificateVerifierTypes.BN254Certificate memory cert,  
  uint32 operatorTableReferenceTimestamp,  
  bytes32 resultHash  
) internal pure {  
  require(cert.referenceTimestamp == operatorTableReferenceTimestamp,  
InvalidReferenceTimestamp());  
  require(cert.messageHash == resultHash, InvalidMessageHash());  
-  require(cert.signature.X != 0 && cert.signature.Y != 0, EmptyCertificateSignature());  
+  require(!(cert.signature.X == 0 && cert.signature.Y == 0),  
EmptyCertificateSignature());  
}
```

Customer's response: Fixed in [948a859](#).

Fix Review: Fix confirmed.

Informational Issues

I-01. `isValidRelease()` and `getLatestUpgradeByTime()` may panic when no releases exist

Description:

The functions `isValidRelease()` and `getLatestUpgradeByTime()` in the `ReleaseManager` contract assume that at least one release exists for the given operator set. However, if no releases have been published, these functions will attempt to subtract 1 from zero, causing an underflow and reverting with a generic error. This is inconsistent with other parts of the contract, such as `getLatestRelease()`, which explicitly checks for the presence of releases and reverts with a clear error (`NoReleases()`).

Recommendations:

Add an explicit check to ensure the operator set has at least one release, and revert with `NoReleases()` when appropriate. This would improve consistency and make errors easier to diagnose.

Customer's response: Fixed in [500eff1](#).

Fix Review: Fix confirmed.

I-02. Incorrect NatSpec in `registerExecutorOperatorSet()`

Description:

`registerExecutorOperatorSet()` has the following NatSpec:

JavaScript

```
/**
 * @notice Registers an executor operator set with the TaskMailbox
 * @param operatorSet The operator set to register
 * @param isRegistered Whether the operator set is registered
 * @dev This function can be called to toggle the registration once the
task config has been set.
 */
function registerExecutorOperatorSet(OperatorSet memory operatorSet,
bool isRegistered) external;
```

However, the `isRegistered` input actually indicates whether the operator set is going to be registered or deregistered.

Recommendations:

Consider updating the NatSpec to accurately describe the function behavior.

Customer's response: Fixed in [557be06](#).

Fix Review: Fix confirmed.

I-03. Incorrect diagram in Hourglass framework documentation

Description:

In the [diagram](#) explaining the basic workflow of the Hourglass framework, step 4 appears twice, while step 3 is missing entirely. This may cause minor confusion for readers trying to follow the intended sequence.

Recommendations:

Update the diagram to correct the step numbering so that each step appears once and in order.

Customer's response: Acknowledged.

I-04. Unused imports can be removed

Description:

The following import statements have been confirmed as unused:

- TaskMailbox.sol: [line 20](#)
- TaskMailboxStorage.sol: [line 5](#)
- ReleaseManagerStorage.sol: [line 5](#)
- TaskAVSRegistrarBase.sol: [line 11](#)

Recommendations:

Remove these import statements to clean up the codebase.

Customer's response: Fixed in [cff0153](#).

Fix Review: Fix confirmed.

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.