certora

Security Assessment Report

EigenLayer Multichain PT 1

July 2025

Prepared for the EigenLayer team

Table of Contents

Project Summary

Project Scope

Project Name	Repository (link)	Initial Commits	Reviewed Commits	Platform
EigenLayer MultiChain PT1	Eigenlayer-contracts and Eigenlayer-middleware	a1df1a6 and 285e9d9	a3adb74, 53bb19c, c6be96a and c77cc5f, ec35d75, 4982292	EVM

Project Overview

This document describes the findings of the manual review of **EigenLayer MultiChain PT 1**. The work was undertaken from **July 14, 2025** to **July 22, 2025.**

The following files are included in our scope:

- eigenlayer-contracts/src/contracts/permissions/KeyRegistrar.sol
- eigenlayer-contracts/src/contracts/multichain/BN254CertificateVerifier.sol
- eigenlayer-contracts/src/contracts/multichain/ECDSACertificateVerifier.sol
- eigenlayer-middleware/src/middlewareV2/tableCalculator/BN254TableCalculator.sol
- eigenlayer-middleware/src/middlewareV2/tableCalculator/ECDSATableCalculator.sol
- eigenlayer-middleware/src/middlewareV2/registrar/AVSRegistrar.sol
- eigenlayer-middleware/src/middlewareV2/registrar/presets/AVSRegistrarAsIdentifier.sol
- eigenlayer-middleware/src/middlewareV2/registrar/presets/AVSRegistrarWithAllowlist.sol
- eigenlayer-middleware/src/middlewareV2/registrar/presets/AVSRegistrarWithSocket.sol

Protocol Overview

The EigenLayer multichain features aim to allow AVS to broadcast the stake of operators to L2 chains, thereby extending the reach of the EG system.

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	1	1	1
Medium	1	1	1
Low	2	2	1
Informational	7	7	7
Total	11	11	10

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
H-01	Incorrect array construction in BN254TableCalculator leads to corrupted operatorInfoTreeRoot	High	Fixed
M-01	Broken Timestamp Validation in ECDSACertificateVerifier Prevents Verification of Older Certificates	Medium	Fixed
L-01	Missing access control can lead to state pollution	Low	Acknowledged
L-02	Suboptimal Deployment of AVSRegistrarAsIdentifier Due to Passing AVS in the Constructor	Low	Fixed

High Severity Issues

H-01 Incorrect array construction in BN254TableCalculator leads to corrupted operatorInfoTreeRoot		
Severity: **High**	Impact: **High**	Likelihood: **High**
Files: BN254TableCalculatorBase.sol#150	Status: Fixed	

Description:

The function _calculateOperatorTable constructs an array, operatorInfoLeaves, which contains the weight of all operators that have registered a valid key.

When an operator has no valid key, the loop skips the operator. Once all operators have been checked, the array is truncated to the actual count of registered operators.

The issue is that registered operators are not added one by one in the operatorInfoLeaves array, but are directly set with an index i identical to the i of the initial for loop. This means that zero-value gaps will occur if registered operators are alternating with non-registered operators.

As an example:

- 4 operators A, B, C, D
- A & D have registered keys, B and C do not
- operatorInfoLeaves is created with size 4

Loop 0: A is set to operatorInfoLeaves[0]

Loop 1: B has no key, he is skipped

Loop 2: C has no key, he is skipped

Loop 3: D is set to operatorInfoLeaves[3]

The array now contains: `[A][0][0][D]`

Then the array is truncated with the actual number of registered operators, which is 2.

The array now contains: `[A][0]`

This is then Merkleized into `operatorInfoTreeRoot` with completely incorrect values.

Recommendations:

Add the values one by one to the array instead of copying the index of the for loop. The approach in ECDSATableCalculatorBase can be followed.

Customer's response: Fixed in 6088c42.

Fix Review: Fix confirmed.

Medium Severity Issues

M-01 Broken timestamp validation in ECDSACertificateVerifier prevents verification of older certificates		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: ECDSACertificateVerifier#331	Status: Fixed	

Description:

In the ECDSACertificateVerifier, the _verifyECDSACertificate() function invokes getTotalStakeWeights(). However, the validation inside getTotalStakeWeights() performs a restrictive check on the provided referenceTimestamp:

```javascript
 function getTotalStakeWeights(
        OperatorSet calldata operatorSet,
        uint32 referenceTimestamp
    ) public view returns (uint256[] memory) {
        bytes32 operatorSetKey = operatorSet.key();
        require(_latestReferenceTimestamps[operatorSetKey] ==
referenceTimestamp, ReferenceTimestampDoesNotExist());
```

This validation does not actually check whether the `referenceTimestamp` exists — rather, it enforces that only the latest reference timestamp associated with the given `operatorSet` is considered valid.

As a result, the update fails to deliver its intended capability: the system cannot process certificates tied to older reference timestamps.

Recommendations:
Consider removing the restrictive check or replacing it with existence validation.

Customer's response: Fixed in 6dce5a9.

Fix review: Fix confirmed.

Low Severity Issues

L-01 Missing access control can lead to state pollution		
Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: SocketRegistry#23	Status: Acknowledged	

Description:

In the SocketRegistry contract, we have the external updateSocket function, which allows an operator to update their socket data.

```
Rust
    /// @inheritdoc ISocketRegistry
    function updateSocket(address operator, string memory socket) external {
        require(msg.sender == operator, CallerNotOperator());
        _setOperatorSocket(operator, socket);
    }
```

The require statement only enforces that `operator address == msg.sender`. But there is nothing checking that the operator is actually an operator registered to the AVS.

As a result, anyone can call the function and store random data under their own address, thereby polluting the `_operatorToSocket` mapping. Depending on the use by the AVS of this mapping, this can be an annoyance or a breaking issue.

Recommendations: Add access control so that only an operator registered in the AVS can update the socket.

Customer's response: Acknowledged. We are acking since there is no way to check membership of the AVS in the AllocationManager. The only introspection is on a per-operatorSet basis. Checking membership on a per-operatorSet basis would require the SocketRegistry to have strings per opSet, which is not a use case we are supporting at the moment

L-02 Suboptimal deployment of AVSRegistrarAsIdentifier due to passing AVS in the constructor

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: AVSRegistrar#31 AVSRegistrarAsIdentifier#22	Status: Fixed	

Description:

In the AVSRegistrarAsIdentifier, the following comment suggests that the immutable AVS address can be different from the actual AVS.

In EigenLayer core:

```Rust
  /// @dev The immutable avs address AVSRegistrar is NOT the address of the AVS in EigenLayer core.
```

However, setting an alternative AVS will cause issues in _validateOperatorKeys:

```Rust
 function _validateOperatorKeys(
        address operator,
        uint32[] calldata operatorSetIds
    ) internal view {
        for (uint32 i = 0; i < operatorSetIds.length; i++) {
            OperatorSet memory operatorSet = OperatorSet({avs: avs, id: operatorSetIds[i]});
            require(keyRegistrar.checkKey(operatorSet, operator), KeyNotRegistered());
        }
```

Furthermore, it is never enforced that avs == address(this), i.e., that the AVS address is set to the proxy contract. As a result, this may lead to issues when deploying AVSRegistrarAsIdentifier, since the expected behavior assumes the AVS is the proxy.

Moreover, standard ERC1967 proxy deployment requires specifying the implementation address at the time of proxy creation. During this process, the implementation address cannot be referenced by the proxy address, because deploying the implementation itself requires passing the proxy address in the constructor:

```
Rust
constructor(

        address _avs,
```

As a result, deploying the AVS requires multiple manual steps, rather than simply assigning the _avs address during the `initialize` function, which would align with standard upgradeable contract patterns and simplify the deployment flow.

Recommendation:
Consider updating the comments and specifying the AVS address upon initialization.

Customer's response: Fixed in f32dc54.

Fix Review: Fix confirmed.

Informational Issues

I-01. Missing length validation of inputs in _registerBN254Key

Description:
In the `_registerBN254Key` function of KeyRegistrar.sol, the abi.decode operation is used to extract BN254 key components from two bytes calldata inputs (`keyData` and `signature`) without explicit checks on their lengths.

If the calldata is longer than needed, the solidity decoder will not revert. While being mostly theoretical, this could, in some rare edge cases, lead to subtle bugs involving shifting bytes.

Recommendation:

As a best practice, it is recommended to enforce the bytes to be the exact length required.

- keydata = 192
- signature = 64

Customer's response: Fixed in 3c88320.

Fix Review: Fix confirmed.

I-02. ECDSA Key Registration allows smart contract addresses due to the validation check being too permissive

Description:
The `KeyRegistrar._registerECDSAKey()` function accepts smart contract addresses as valid ECDSA keys due to overly permissive signature verification checks. The function uses `_checkIsValidSignatureNow()`, which calls `isValidSignatureNow`, which supports both ECDSA and ERC-1271 signature schemes, allowing smart contracts to successfully register as "ECDSA keys".

The operators cannot use this fake ECDSA key since it will revert on ECDSACertificateVerifier._parseSignatures, so there is no impact aside from blocking themselves.

Still, since there is no functional reason for allowing ERC1271 signature schemes, the validation check should be restricted to avoid any unforeseen future side effects.

Recommendation: Restrict the check to not allow ERC-1271.

Customer's response: Acknowledged. The documentation of KeyRegistrar.md has been updated to address this issue.

I-03. Incorrect Interface naming can lead to confusion and mistakes

Description:
The `AVSRegistrarWithSocket` is `SocketRegistry,` which is `SocketRegistryStorage`, which is `ISocketRegistry`.

However, this interface is stored in `ISocketRegistryV2.sol`.

In the interface folder, we can find `ISocketRegistry.sol,` which also holds an `ISocketRegistry` interface.

Both interfaces have different functions, errors, and events. This is not a recommended practice since it can easily lead to import confusion, interface collision, and developer mistakes.

Recommendation: It is recommended to rename the interface used in `SocketRegistryStorage` to `ISocketRegistryV2` in order to avoid confusion.

Customer's response: Fixed in f900920

Fix Review: Fix confirmed.

I-04. Incorrect NatSpecs and comments

Description:

Several contracts have incorrect or out-of-date documentation in the function NatSpecs and inline comments.

1. **AVSRegistrar** states in registerOperator that _validateOperatorKeys "update[s] key if needed" (line 47), but we can see it is an `internal view` function which does not and cannot update the state.

2. **AVSRegistrar** contains the following line in the NatSpec for _validateOperatorKeys (line 90):

```
None
@dev This function assumes the operator has already registered a key in the Key Registrar
```

   The function does **not** assume that an operator has a key, the goal of the function is to validate the existence of such a key.

3. In **IECDSATableCalculator**, it has NatSpec of calculateOperatorTable states (line 15):

```
None
@dev The output of this function is converted to bytes via the calculateOperatorTableBytes
function
```

   This is incorrect since calculateOperatorTableBytes is a separate external function not linked to `calculateOperatorTable`.

4. In **IKeyRegistrar**, the NatSpec for the checkKey function states (line 88):

```
None
@dev Only authorized callers for the AVS can call this function
```

   Looking at the actual checkKey function, we can clearly see there is no access control, and anyone can call this function to check the key registration status for an operator.

5. **IAVSRegistrar** states in the NatSpec of deregisterOperator (line 23) that a reverting call will be ignored. This is no longer the case since the try/catch in AllocationManager has been removed.

Recommendation: These comments should be updated and removed as necessary.

Customer's response: Fixed in b94926b & ca8dae3

Fix Review: Fix confirmed.

I-05. Unused import statements

Description:

Several contracts include unused import statements that can be safely removed:

1. **AVSRegistrarAsIdentifier** imports Initializable from OpenZeppelin (line 14) but does not use it.
2. **AVSRegistrarWithSocket** imports OperatorSetLib and OperatorSet (lines 11–14) but only uses them in a parent contract, not directly.
3. **ECDSATableCalculatorBase** imports the Merkle library and enables it by using Merkle for bytes32[], but never calls any of its functions. (line 18).
4. **KeyRegistrar** imports Initializable (line 4), but the contract does not inherit from it or use it.

Recommendation: Remove the unused import statements

Customer's response: Fixed in 3fc52b1, 4982292, & c6be96a

Fix Review: Fix confirmed.

I-06. Mismatch in `ECDSAOperatorInfo.weights` lengths causes incorrect stake calculations

Description:

The `ECDSACertificateVerifier` contract contains a flaw in how it handles operator weights arrays of different lengths within the same operator set. The contract allows operators to be registered with varying `weights` array sizes through `updateOperatorTable()` without any validation, but the stake calculation logic in `getTotalStakes()` assumes all operators have an identical weights structure. The core issue stems from line 337, where `getTotalStakeWeights()` uses `operator[0]` to determine how many stake categories to use based solely on the first operator's weights.

This means the whole logic depends on whichever operator is stored first. If operator[0] has fewer categories than other operators, those extra categories get ignored during calculations.
Example:

- Operator[0]: `weights = [1_000]`
- Operator[1]: `weights = [1_000, 2_000, 3_000]`
- Operator[2]: `weights = [1_000, 2_000, 3_000]`
- Operator[3]: `weights = [1_000, 2_000, 3_000]`

Expected total stakes: `[4_000, 6_000, 9_000]`

Actual total stakes: `[4_000]` (only 1 category based on operator[0]).

This bug fails silently with no errors, so the contract appears to work while giving wrong results. Functions like `verifyCertificateProportion()` and `verifyCertificateNominal()` use these incorrect totals for security checks, potentially allowing certificates to pass when they shouldn't.

In `verifyCertificateProportion()`, the function expects to validate thresholds against all intended stake categories, but due to the truncation bug, it can only validate against the categories determined by operator[0].

Using the example above, when calling `verifyCertificateProportion()` with thresholds `[30%, 40%, 50%]` for all 3 expected categories:

1. `getTotalStakes()` returns `[4_000]` (missing 2nd and 3rd categories worth `6_000` and `9_000`)
2. `signedStakes` also truncated to `[4_000]`
3. Function requires `signedStakes.length == totalStakeProportionThresholds.length`
4. Result: `1 != 3`, function reverts with `ArrayLengthMismatch()`

If the caller adjusts and only passes `1 threshold [30%]`:
- The function works, but completely ignores the 2nd and 3rd category validation
- A certificate could pass even if the 2nd and 3rd categories had insufficient stake

Recommendation: If the intention is for all operators to have the same length for their weights, validate this during the `updateOperatorTable` function.

Customer's response: Fixed in 5f6006c

Fix Review: Fix confirmed.

I-07. Missing onlyInitializing modifier in __AVSRegistrar_init

Description:

The AVSRegistrar contract defines an internal initialization function:

```Rust
/// @dev This initialization function MUST be added to a child's `initialize` function to avoid uninitialized storage.

    function __AVSRegistrar_init(

        address _avs

    ) internal virtual {

        avs = _avs;

    }
```

This function is intended to be invoked during contract initialization, as noted in its documentation. However, it lacks the onlyInitializing modifier, which enforces that the function can only be called within the context of an initialization phase.

Recommendation: Consider adding the onlyInitializing modifier.

Customer's response: Fixed in: c77cc5f.

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.