Code Assessment
of the Sky LayerZero Governance DVN Smart Contracts

PUBLIC

Date June 23, 2026

Produced for Sky

By CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Sky with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Sky LayerZero Governance DVN according to **Scope** to support you in forming an opinion on their security risks.

Sky implements a cross-chain governance verification system that confirms a single message across three independent wings, Chainlink CCIP, a Sky-controlled multisig, and a quorum of independent LayerZero-aligned DVN providers, configured so that any two suffice and no single wing does. A set of deployment contracts deploys and wires these wings before handing control to Sky governance.

The most critical subjects covered in our audit are the functional correctness of the attestation broadcast and the access control over the verifier and the deployer's administrative roles. The general subjects covered are event handling, error handling, code complexity, and documentation.

In summary, we find that the codebase provides a good level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code of the `lz-dvn-broadcaster` and `lz-gov-dvns-deploy` repositories, based on the provided documentation. The table below indicates the code versions relevant to this report and when they were received.

lz-dvn-broadcaster

Version	Date	Commit Hash	Note
1	17 June 2026	f5c093b92996de09805f4de21dcc5676de00bf4d	Initial review

lz-gov-dvns-deploy

Version	Date	Commit Hash	Note
1	17 June 2026	5729b9a1e65b580a50d0d53ed147b26048c677ea	Initial review
2	23 June 2026	c72a2b3e5cf4b40546fa4f92611996e86354616e	Minor changes

For the Solidity smart contracts, the compiler version `0.8.24` was chosen.

The following files are in scope:

```
lz-dvn-broadcaster/
└── src/
    ├── DVNBroadcaster.sol
    └── DVNReplica.sol
lz-gov-dvns-deploy/
└── src/
    ├── LZDVNInit.sol
    ├── RecvSideDeployer.sol
    └── SendSideDeployer.sol
```

2.1.1 Excluded from scope

Tests and deployment scripts are excluded. The imported LayerZero and Chainlink dependencies are out of scope and are treated as trusted, in particular the `CCIPDVNAdapter` and `CCIPDVNAdapterFeeLib` (reviewed separately), the LayerZero endpoint and message libraries, and the Chainlink CCIP contracts.

3 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Sky LayerZero Governance DVN lets a governance application confirm a cross-chain message across several independent attestation paths, called wings, of which a configurable quorum must agree before the destination endpoint delivers the message. Sky uses it to bridge governance from Ethereum to other chains, with three wings: Chainlink CCIP, a Sky controlled multisig, and a quorum of independent LayerZero aligned DVN providers. Each wing is realized as one or more DVNs in LayerZero's standard verification, and the weighting, configured in the application's LayerZero receive configuration, makes any two out of the three wings sufficient and no single wing sufficient on its own.

`DVNBroadcaster` broadcasts a single attestation to a fixed set of `DVNReplica` contracts; each replica records one attestation in the LayerZero receive library and therefore counts as one DVN in the application's quorum. The deployment and initialization consists of three contracts, `SendSideDeployer`, `RecvSideDeployer`, and `LZDVNInit`, which deploy and wire the CCIP and multisig wings on the source and destination chains.

3.1 DVNBroadcaster

A `DVNBroadcaster` broadcasts one attestation from its verifier to a fixed set of `DVNReplica` contracts, so that a wing contributes several DVN slots from a single upstream event. The constructor records the LayerZero `endpoint` and the `verifier` and spawns a fixed number of replicas, each owned by the broadcaster.

`verify()` is callable only by the `verifier`: for the CCIP wing the destination `CCIPDVNAdapter`, which invokes it as its configured receive library when delivering an inbound CCIP message, and for the multisig wing the Sky multisig, which calls it directly. It checks the packet header version, reads the source endpoint id and the receiver address from the header, resolves the receiver's current receive library through `Endpoint.getReceiveLibrary`, and calls `verify()` on every replica with that library. The `confirmations` value is chosen by the verifier according to its own finality model; the CCIP wing uses `type(uint64).max`.

The contract provides the following functions:

- `verify()`: callable only by the verifier; broadcasts the attestation to all replicas against the receiver's current receive library.
- `getReplicas()` and `getReplicasCount()`: views over the spawned replicas.

3.2 DVNReplica

`DVNReplica` is a minimal contract spawned by a `DVNBroadcaster`, which is its `verifier`. Its single function `verify(receiveLib, packetHeader, payloadHash, confirmations)` is callable only by that verifier and forwards to `receiveLib.verify()`. Because each replica calls under its own address, it records one independent attestation in the LayerZero receive library.

3.3 Deployment and Initialization

The remaining three contracts deploy and wire the system before handing control to Sky governance.

The following order of execution is expected:

1. `SendSideDeployer` is deployed on L1.
2. `RecvSideDeployer` instances are deployed on the respective L2s.
3. `SendSideDeployer.configure()` is invoked for each L2 instance.
4. *The configuration is tested.*
5. `SendSideDeployer.handOff()` is invoked.
6. Any subsequent deployment, performs step 2 and 3. However, step 3 is performed through a spell using `LZDVNInit`.

3.3.1 SendSideDeployer

`SendSideDeployer` deploys and configures the CCIP wing on the source chain (Ethereum) and then transfers control to Sky governance. The constructor deploys a `CCIPDVNAdapterFeeLib` (initialized once, with ownership renounced, since this fee library version reads none of the owner set configuration), deploys a `CCIPDVNAdapter` with the `SendSideDeployer` as its admin, points the adapter at the fee library, grants `MESSAGE_LIB_ROLE` to the send library so accrued fees can later be swept, and allowlists the provided governance applications. The adapter's allowlist restricts which applications may use the wing, and is enforced only while at least one application is allowlisted. The `CCIP_ROUTER` and `CHAINLOG` addresses are hardcoded Ethereum mainnet addresses.

The contract provides the following functions:

- `configure()`: callable only by the deployer; wires the CCIP adapter for a remote through `LZDVNInit`.
- `withdrawFunds()`: callable only by the deployer; recovers test funds before handoff.
- `handOff()`: callable only by the deployer; revokes the given applications from the allowlist, grants `DEFAULT_ADMIN_ROLE` and `ADMIN_ROLE` to the Sky `MCD_PAUSE_PROXY`, and revokes both roles from itself.

3.3.2 RecvSideDeployer

`RecvSideDeployer` deploys the destination chain side in its constructor: a `CCIPDVNAdapter` configured to accept inbound CCIP messages from the source adapter for the Ethereum endpoint, a CCIP `DVNBroadcaster` whose verifier is that adapter, and a multisig `DVNBroadcaster` whose verifier is the Sky multisig, each with a configured number of replicas. It then revokes its own admin roles, leaving the adapter with no admin: its configuration is fixed, and the deployment only supports delivery from Ethereum to the destination chain.

3.3.3 LZDVNInit

`LZDVNInit` is a library that wires a CCIP adapter for one remote. It redeclares inline the structs and interfaces it uses, so it can be copied into governance spells that do not carry the LayerZero dependencies. `wireCCIPDVN()` sets the adapter's destination configuration (chain selector, peer, gas, and fee multiplier) and points the adapter's `receiveLibs` entry for the remote at the remote CCIP broadcaster, so that a CCIP relayed attestation is delivered to that broadcaster rather than directly to the receive library. It requires the fee multiplier to be either zero (the adapter default) or at least `1e4`: the fee is `ccipFee * multiplierBps / 1e4`, so `1e4` charges exactly the CCIP fee (break even) and a lower value would undercharge. It performs no other input validation, assuming deployment correctness is checked off-chain.

4 Trust Model

This section enumerates the privileged roles and trusted external components of the broadcaster and deployment contracts.

Note: each `DVNBroadcaster` contributes a fixed number of attestations, one wing, to an application's DVN quorum. Under the intended receive configuration any two of the three wings suffice and no single wing does, so none of the roles below can forge a cross-chain message on its own. The worst single wing outcomes are contributing that wing's attestations toward, but not completing, a quorum, and withholding them so that messages relying on the wing stall. Forging a delivery requires compromising two wings.

4.1 Roles

4.1.1 `verifier` (DVNBroadcaster)

- **Capabilities:** The sole caller of `DVNBroadcaster.verify()`, fixed at construction and immutable. For the CCIP wing it is the destination `CCIPDVNAdapter`, which reaches `verify()` as its configured receive library when relaying an inbound CCIP message; for the multisig wing it is the Sky multisig, which calls `verify()` directly. A call fans the same `packetHeader`, `payloadHash`, and `confirmations` out to every `DVNReplica` of that broadcaster.
- **Trust level:** Fully trusted for its own wing.
- **Worst case:** A compromised verifier can drive its replicas to attest an arbitrary `(packetHeader, payloadHash)` with arbitrary `confirmations`, producing that wing's entire block of attestations for forged data, or can withhold attestations to stall the wing. Because one wing is insufficient under the intended quorum, this alone neither forges nor blocks a delivery; it contributes toward a forged quorum or toward a liveness failure that a second wing would complete.

4.1.2 Deployer

- **Capabilities:** The address that deploys `SendSideDeployer` (immutable `deployer`, recorded as the constructor caller) and gates `configure()`, `withdrawFunds()`, and `handOff()`. Until `handOff()` runs, it holds `DEFAULT_ADMIN_ROLE` and `ADMIN_ROLE` on the source `CCIPDVNAdapter`, which is deployed with the deployer contract as its sole admin. On the destination side the equivalent control exists only inside the `RecvSideDeployer` constructor, which grants itself admin, configures the adapter, and revokes both roles before returning.
- **Trust level:** Fully trusted during deployment.
- **Worst case:** While it holds admin on the source adapter, it has the adapter's full administrative surface (setting the destination peer at initial configuration, repointing the fee library, withdrawing accrued fees and the adapter's native balance, and managing the allowlist). The adapter and its fee library are reviewed separately. The trust is temporal: `handOff()` transfers `DEFAULT_ADMIN_ROLE` and `ADMIN_ROLE` to Sky governance and revokes both from the deployer, after which it retains no privilege.

4.1.3 Sky governance (`MCD_PAUSE_PROXY`)

- **Capabilities:** Receives `DEFAULT_ADMIN_ROLE` and `ADMIN_ROLE` on the source `CCIPDVNAdapter` in `handOff()`, where the address is read from the Sky chainlog under `MCD_PAUSE_PROXY`.
- **Trust level:** Fully trusted.
- **Worst case:** Full administrative control of the source adapter. The destination adapter has no admin after deployment, so governance cannot reconfigure it.

4.2 Trusted external components

- **LayerZero endpoint:** `DVNBroadcaster.verify()` calls `getReceiveLibrary(receiver, srcEid)` to resolve, at call time, the receive library each replica attests to. The endpoint is trusted to return the receiver's honest receive library.
- **ReceiveULN (`IReceiveUln`):** each `DVNReplica` calls `verify(packetHeader, payloadHash, confirmations)` on the library resolved above.
- **Chainlink CCIP:** the upstream of the CCIP wing. The destination `CCIPDVNAdapter` accepts an inbound message only from the CCIP router and the configured peer, then drives the CCIP broadcaster. CCIP's honesty and liveness are this wing's security and availability.
- **Sky multisig:** the verifier of the multisig wing, trusted to attest only messages it has independently confirmed.
- **Independent LayerZero aligned DVN providers:** the third wing, trusted to verify independently of the CCIP and multisig wings.
- **Sky chainlog:** read at the hardcoded mainnet address in `handOff()` to resolve `MCD_PAUSE_PROXY`, trusted to return the correct governance address.
- **`CCIPDVNAdapter` and `CCIPDVNAdapterFeeLib`:** deployed and configured by the deployment contracts and reviewed separately.

4.3 Assumptions for external components

- The application's receive configuration lists the replica addresses of each wing as DVNs and weights them so that any two of the three wings suffice and no single wing does.
- The LayerZero endpoint returns the honest receive library for the receiver and source endpoint id, since the broadcaster forwards every attestation to whatever address it returns.
- CCIP delivers to the destination adapter only messages that genuinely originated from the configured peer, and the Sky multisig and the independent DVN providers verify independently, so that no two wings share a common point of failure.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 Receive Library Rotation Grace Period Is Not Supported

System Consideration | Version 1

`DVNBroadcaster.verify()` resolves its target with `getReceiveLibrary` and fans the `verify()` call out to every replica against that single library. `getReceiveLibrary` returns only the currently configured receive library, or the LayerZero default.

LayerZero V2 supports rotating an OApp's receive library with a grace period. `setReceiveLibrary` records the previous library as a `Timeout` (the default library variant is `setDefaultReceiveLibrary`, and `setReceiveLibraryTimeout` adjusts it). While the timeout has not expired, `isValidReceiveLibrary` accepts both the new library and the previous one, and the endpoint's `verify` gates commit only on `isValidReceiveLibrary`. The grace period exists precisely so independent verifiers can keep attesting on the old library while the OApp migrates to the new one.

Attestations are stored per library in that library's own `hashLookup`, and the `optionalDVNThreshold` quorum must be reached within a single library. Because the broadcaster always reads `getReceiveLibrary`, its replicas only ever attest on the new library. It has no notion of the grace library and cannot also serve the old one.

This is a documented limitation. The README states that the timeout receive library is not supported and that the OApp's receive library is expected to be set explicitly.

6 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

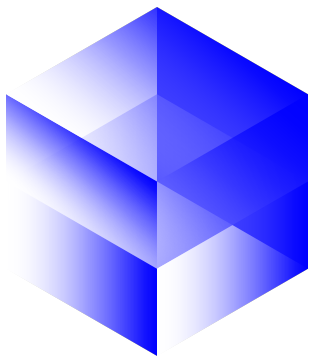
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

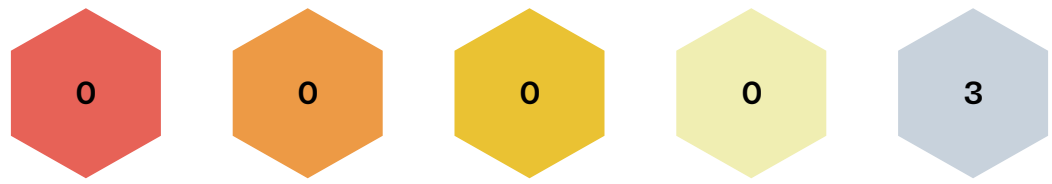
7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



No Open Issues

Total number of findings



Critical | High | Medium | Low | Informational

7.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Info	#001	Packet Header Length Is Not Explicitly Checked	Acknowledged	⊖
Info	#002	SendSideDeployer Emits No Events	Acknowledged	⊖
Info	#003	CCIP Adapter's `defaultMultiplierBps` Is Left at the 120% Constructor Default	Code Corrected	✓

7.2 Descriptions

#001 Packet Header Length Is Not Explicitly Checked

Informational	Version 1	Acknowledged

`DVNBroadcaster.verify()` reads fields from `packetHeader` at fixed calldata offsets and relies on calldata bounds checking instead of an explicit length check.

A header shorter than 81 bytes reverts inside the calldata slicing, without an explicit, named `DVNBroadcaster` error.

A header longer than 81 bytes is not rejected by the broadcaster. The trailing bytes are never read, so `verify()` resolves the receive library and fans the call out to every replica, and `ReceiveUlnBase._verify()` records an attestation keyed by `keccak256(packetHeader)` without any length validation. The oversized header is caught only at commit time, where `ReceiveUlnBase._assertHeader()` requires the header length to equal 81 and the reconstructed header hash cannot match. The stray attestations are inert but still consume the library lookup and the replica and ULN storage writes.

Acknowledged

Sky states:

> Acknowledged, we think the check is not mandatory.

#002 SendSideDeployer Emits No Events

Informational	Version 1	Acknowledged

`SendSideDeployer`'s `configure`, `withdrawFunds`, and `handOff` let the `deployer` reconfigure the adapter, sweep funds, and transfer admin to the pause proxy without emitting any event, and the contract declares no events at all, so external indexers or operational monitoring that track emitted events would not observe these actions directly. Only some surface indirectly through adapter events such as `RoleGranted` / `RoleRevoked`.

Acknowledged

Sky states:

> The meaningful actions already emit via the adapter.

#003 CCIP Adapter's `defaultMultiplierBps` Is Left at the 120% Constructor Default

Informational	Version 1	Code Corrected

`SendSideDeployer.constructor` covers the `CCIPDVNAdapter` config entrypoints relevant at deployment time, with one exception:

- invoked: the access control roles and `setWorkerFeeLib`
- deferred to `configure`: `setDstConfig` and `setReceiveLibs`
- not config functions: `withdrawFee` / `withdrawToken` / `assignJob` / `ccipReceive` / getters
- irrelevant here: `setPaused` / `setPriceFeed` / `setSupportedOptionTypes`

- the exception, never called: `setDefaultMultiplierBps`

So `defaultMultiplierBps` keeps the `12000` (120%) value the `CCIPDVNAdapter` constructor sets, even though the deployer holds the `ADMIN_ROLE` to change it. The premium is `multiplierBps == 0 ? defaultMultiplierBps : multiplierBps`, and `wireCCIPDVN` permits `cfg.multiplierBps == 0`, so a `0` per-destination multiplier charges a 20% premium rather than the break-even it reads as.

> **Code Corrected**
>
> `SendSideDeployer.constructor` now calls `setDefaultMultiplierBps(10_000)`.

8 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.