PUBLIC

Code Assessment

of the Sky Governance OApp Smart Contracts

October 17, 2025

Produced for

by

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 Governance OApp according to Scope to support you in forming an opinion on their security risks.

Sky implements a Governance OApp using the LayerZero V2 stack to facilitate the relaying of messages in a standardized manner for EVM chains and Solana.

The most critical subjects covered in our audit are functional correctness, access control, and compatibility with Sky governance. After the intermediate report, Arbitrary Call in lzReceive has been resolved and security regarding all the aforementioned subjects is now considered high.

The general subjects covered are gas efficiency and documentation. Documentation can be enriched.

Further, some general considerations are provided for secure integration with LayerZero V2, in particular:

- LayerZero V2 Considerations
- Message Passing Considerations
- OApp Call Validation

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

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	1
• Code Corrected	1
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1
• Code Corrected	1

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 files inside the Sky Governance OApp repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	29 June 2025	3aa51c88f9fa2f6660c99a807b120d8705b88932	Initial Version
2	15 Sep 2025	efc7a928fd9053b9e7d61ded2b7ac1a5358609f2	After Intermediate Report
3	09 Oct 2025	175f63c940a33e9037f82b16d0e7fa034e2d3e34	Permissioned Initialization
4	16 Oct 2025	5ad5cb6bbe624e2b1cb99acfe3e4140fa1c233b9	Final Version

For the solidity smart contracts, the compiler version `0.8.22` was chosen and `evm_version` is set to `shanghai`.

For the solana programs, the rust version `v1.75.0`, the anchor version `v0.29` and the solana CLI version `v1.17.31` were chosen.

The following files were in scope:

```
contracts/
    GovernanceControllerOApp.sol
    GovernanceMessageEVMCodec.sol
    GovernanceMessageGenericCodec.sol
    IGovernanceController.sol

programs/governance/
    error.rs
    lib.rs
    msg_codec.rs
    instructions/
        init_governance.rs
        lz_receive_types_info.rs
        lz_receive_types_v2.rs
        lz_receive.rs
        mod.rs
        set_oapp_config.rs
        set_remote.rs
    state/
        governance.rs
        mod.rs
        remote.rs
```

Since (Version 2), the EVM contracts were refactored and the following solidity smart contracts were in scope:

```
contracts/
    GovernanceOAppReceiver.sol
    GovernanceOAppSender.sol
    interfaces/
        IGovernanceOAppReceiver.sol
        IGovernanceOAppSender.sol
```

2.1.1 Excluded from scope

Generally all other files are out of scope.

LayerZero V2 itself is out of scope and assumed to function correctly as per its documentation.

The configurations of OApp and its settings (e.g. choice of libraries) on LayerZero V2 are out of scope. The DVN, executor, send and receive libraries are assumed to be properly configured.

Tests and imports, including the OApp-EVM contracts are out of scope and treated as external dependencies for this review. While the core OApp-EVM contracts are out of scope, we reviewed their implementation to understand their behavior and to confirm that the in scope contracts integrate with and use them correctly. The OFT/OApp smart contracts on EVM were previously reviewed by ChainSecurity (LayerZero OFT/OApp Audit). These contracts are particularly relevant, as the contracts in scope inherit from them.

Further, the Solana programs make use of the Anchor framework which is assumed to function properly and work as documented.

2.2 System Overview

This system overview describes the latest received version (Version 2) 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 offers a Governance OApp using the LayerZero V2 stack to facilitate the relaying of messages in a standardized manner. Currently, implementations for EVM chains as well as Solana are provided. Note that the terminology may vary depending on the chain and that the system overview might use one terminology.

2.2.1 Governance OApp Overview

Overview. The governance OApp (Omnichain Application) implements a standardized message passing framework for sending and receiving cross-chain messages that internally leverages the LayerZero V2 stack. It implements a unified approach that allows governors (e.g. Sky governance or Sky SubDAOs) to use a managed framework for message passing to trigger cross-chain governance actions.

A general overview of the integration and interaction flow of the OApp is outlined below:

1. *Source Chain Governor*. Instructs the source chain OApp to send a message on its behalf.
2. *Source Chain OApp*: Validates the message. Then, sends the message through the LayerZero V2 endpoint with the configured send library to the destination OApp. Note that the execution fee (denominated in native token or LZ token) is paid in this step.
3. *LayerZero V2*: Handles the message. Eventually, the message is verified on the LayerZero V2 endpoint, marking its validity. Note that this step includes on- and off-chain components.

4. *Destination Chain OApp*: The execution of the cross-chain message is triggered and the cross-chain governance action is executed. Note that this adheres to the receive library.
5. *Destination Chain Governor*: The target receives the call and performs the action if and only if the message originates from an expected governor and source chain pair.

However, note that depending on the configuration and the chain the exact details may vary. Similarly, note that not all OApps will necessarily send or receive messages. For details relevant to either sending or receiving, we will refer to "sending OApp" and "receiving OApp" accordingly in this system overview.

Messaging. Note that the messages passed between the OApps follow the generic format below:

Origin Caller	32 bytes	Address of the OApp caller on the source chain.
Destination Target	32 bytes	Address of the Target to be called.
Destination Calldata	Variable	The calldata to be executed on the destination target.

For sending OApps, the following holds for messaging:

- The destination EIDs must be whitelisted which implies that the OApp has been deployed on the destination chain (i.e. linking of OApps is required).
- The origin caller must be the initiator of the OApp cross-chain message (e.g. `msg.sender` on EVM) to prevent impersonation.
- The origin caller must be whitelisted by the OApp governor if whitelisting is activated.
- The origin caller should ensure the correctness of the additional data as this might be chain-specific.

For receiving OApps, the following holds for messaging:

- The message is not replayable and is expected to be correct (given the LayerZero V2 correctness assumption).
- The destination EID must be the EID of the chain the OApp is deployed on (however, should be generally handled by the above).
- The receiving OApp provides a mechanism allowing the receiving target to validate the origin of the message (based on source chain EID and origin caller).

Governance Functionality. Several privileged roles can configure the Governance OApp as well as its configuration on the LayerZero V2 endpoint. Below, the generally available functionalities are listed.

The following general owner-privileged actions exist for all OApps:

- *Set Owner*: Sets the OApp's owner.
- *Set Delegate*: Sets the OApp's delegate address on the LayerZero V2 endpoint.
- *Set Peer*: Sets the peer address for a specific EID.

The following owner-privileged actions exist for sending OApps:

- *Set Enforced Options*: Sets the enforced options for specific endpoint and message type combinations.
- *Adding and Removing from Allowlist*: Manage allowlist entries.
- *Enabling and Disabling Allowlist*: Activate and deactivate the allow list.

Below are the relevant actions on the LayerZero V2 endpoint for the delegate (and technically the OApp itself):

- for all OApps:
    - *Set Config*: Sets an OApp's configuration on a registered message library.
- useful for **sending** OApps:

- *Set Send Library*: Sets the send library.

- useful for **receiving** OApps:
  - *Burn*: Marks a nonce as non-executable and non-verifiable. The nonce can never be re-verified or executed.
  - *Nilify*: Marks a packet as verified, but disallows execution until it is re-verified.
  - *Clear*: Clears a message that is efficiently burnt.
  - *Skip*: Skips the next nonce to prevent message verification.
  - *Set Receive Library*: Sets the receive library.
  - *Set Receive Library Timeout*: Sets the timeout for a receive library change during which the previous library could be used.

In general, the OApps should be configured correctly for Sky Cross-Chain Governance, for more details please consult Configuration Considerations.

Last, note that the OApp itself is not expected to hold any privileges on governed contracts.

2.2.2 EVM

Overview. Two OApps are implemented for sending (`GovernanceOAppSender`) and receiving (`GovernanceOAppReceiver`) respectively. Note that receiving a message executes a call according to the data provided by the message.

Messaging. The messages of the EVM Sender and Receiver OApps are outlined as follows:

Origin Caller	32 bytes	Address of the OApp caller on the source chain.
Destination Target	32 bytes	Address of the Target to be called.
Destination Calldata	Variable	The calldata to be executed on the destination target.

For the sending OApp, `quoteTx()` can be called to estimate the fee by forwarding the call to the EndpointV2. Then another call to `sendTx()` can send the message through the EndpointV2. Note authorization is checked upon sending that restricts the `msg.sender` is allowed to call a target contract at a destination chain.

For the receiving OApp, the verified message can be consumed by anyone (typically the paid executor) with `EndpointV2.lzReceive()` which further triggers `OApp.lzReceive()`. That initiates the cross-chain governance action to the governed contract with the calldata and the attached `msg.value`. If the call reverts, the execution reverts.

Note that the `messageOrigin` defines the source EID as well as the initiator of the OApp message. For validating calls performed by the OApp, please consider OApp Call Validation.

Please refer to Configuration Considerations for more details about the **EVM Configuration for Cross-Chain Governance**.

2.2.3 Solana

Overview. The Solana OApp program implements the logic for an OApp that only allows receiving cross-chain messages. Note that the program can be used for multiple OApps.

Namely, the instruction `oapp::init_governance` is provided to create an OApp with a given ID while setting the admin (owner) and relevant Address-Lookup-Tables (ALTs). Note that this generates a governance PDA that will be registered on the LayerZero V2 endpoint with `endpoint::register_oapp` to set its delegate and should be used on the source-chain as the recipient address of the LayerZero V2 cross-chain message.

On Solana, some instructions should be to be called prior to the message reception, especially the delegate-only ones:

- `init_receive_library` / `init_config`: (delegate-only) init the receive library and relevant configurations.
- `set_receive_library` / `set_config`: (delegate or OApp) set the receive library and relevant configurations.
- `init_nonce`: (delegate-only) initialize the Nonce and PendingInboundNonce PDAs.
- `init_verify`: (permissionless) initialize the PayloadHash PDA for each message to be received.

Note for Solana, we refer to the program with the context of the governance PDA as the OApp. Hence, the general descriptions apply to the individual OApps. Note that there is no global configurations.

Additionally, note that so-called CPI authority PDAs will sign the CPI for an OApp and are derived based on the origin caller and the source (and the governance PDA as part of the OApp definition).

Messaging. The Solana OApp implements only receiving messages and the payload layout is defined as follows:

Origin Caller	32 bytes	Caller on the origin domain
Program ID	32 bytes	Target program the OApp will invoke with its CPI Authority PDA.
Accounts Length	2 bytes	Number of accounts present in the item below.
Accounts	Accounts Length * 34 bytes	List of accounts `(account, is_signer, is_writable)` to be passed to the target program with the respective length.
Data	Variable	The data to be passed to the program.

To receive, the verified message can be consumed by anyone (typically the paid executor) with `oapp::lz_receive` which consumes the message on the endpoint with `endpoint::clear`. Then, the program proceeds to perform a CPI to the target program with the data and accounts. The signer of the invocation's signer correspond to the CPI authority PDA. Given the definition of the CPI authority PDA, it can be used for validating the origin of cross-chain governance actions. For details regarding validation, please consider OApp Call Validation.

Last, note that two helper functions are provided for aggregating accounts so that the executor can successfully execute `oapp::lz_receive`. `oapp::lz_receive_types_v2` defines these compactly using the ALTs. `oapp::lz_receive_types_info` provides the accounts needed for `oapp::lz_receive_types_v2`.

Governance Functionality. Note that additional governance functionality is required for Solana. Namely, the following owner-privileged actions are provided:

- *Set ALTs*: Updates the ALTs.

Please refer to Configuration Considerations for more details about the **Solana Configuration for Cross-Chain Governance**.

2.2.4 Changelog

In Version 1, the EVM GovernanceControllerOApp allows both sending and receiving messages. It also provides two different entrypoints to send messages: `sendEVMAction()` and `sendRawBytesAction()`. This has been changed since Version 2, where a different message layout is implemented. Further, in Version 1, an allowlist is defined to whitelist address that can send messages to any targets on any peers; since Version 2 fine-grained access control is implemented that restricts calls to specific target contract on target destination.

2.3 Trust Model

Owner of the Governance OApp. Fully trusted. Can set the `canCallTarget` authorization mapping and OApp configurations (e.g. peers and enforced options). Is expected to perform these configuration changes correctly. In the worst case, the OApp could be maliciously configured to allow receiving malicious cross-chain messages. Additionally, it could DoS or censor the message relay or prevent the execution of messages.

Delegate of the OApp in LayerZero. Fully trusted. Initially set to the owner of the OApp who can reassign this role. Can configure the library and manipulate the message handling on the EndpointV2 contract. In the worst case, the OApp could be maliciously configured to allow receiving malicious cross-chain messages. Additionally, it could DoS or censor the message relay or prevent the execution of messages.

Addresses authorized to send. Fully trusted. Expected to be configured correctly by the owner of the OApp. These address have privileges to trigger calls specific targets on foreign domains. In the worst case the target contracts can be compromised by the respective authorized address.

LayerZero. LayerZero is out of scope for this review and is trusted to behave correctly and deliver messages to the correct destination, the LayerZero executor is trusted to always deliver messages with the correct provided message value and gas limit. Depending on the exact configuration, LayerZero might be fully trusted to not censor messages or similar. However, the configuration is out-of-scope for this review.

The governed contracts would be in danger in case LayerZero behaves incorrectly, for example due to a compromise, an L2 finality issue, or any other failure. Because the `lzReceive()` function cannot be paused in emergency, compromised DVNs and executors may verify and execute unintended or malicious governance calls.

The configuration required for managing the LayerZero applications on multiple chains is considered out of scope for this review and should be performed by the `delegate` or its owner through the utility functions, please consult Configuration Considerations for more details.

3 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.

4 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.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	1

- Arbitrary Call in lzReceive Code Corrected

High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- init_governance Can Be DoSed Code Corrected

Informational Findings	1

- Unchecked Program ID Code Corrected

6.1 Arbitrary Call in `lzReceive`

Security Critical Version 1 Code Corrected

CS-SOAG-001

The EVM contract `GovernanceControllerOApp` implements the `_lzReceive()` function to allow for arbitrary calls to arbitrary targets which could allow for by-passing access control so that messages could potentially be forged arbitrarily.

`_lzReceive()` allows for arbitrary call targets and calldata. As a consequence, any message could call `endpoint` to call privileged functions such as `EndpointV2.setDelegate()`. As a consequence, configurations could be changed in critical ways. For example, the required DVN configuration could be changed to allow for forging malicious messages.

Additionally, `endpoint.lzToken()` could be selected as the target to pull funds from users that have approved the `GovernanceControllerOApp` contract (as part of paying fees in such token).

In general, arbitrary call allows abusing the privileges over all contracts that `GovernanceControllerOApp` has.

Note that the likelihood is limited if allowlists on all source domains are activated due to the partial trust in allowed addresses. However, by default the EVM contracts do not activate the allowlist.

To summarize, a severe escalation of privileges is possible due to arbitrary function calls.

Code corrected:

Since Version 2, the EVM Gov OApp has been redesigned to require authorization for each triplet `(_srcSender,dstEid,dstTarget)`. Consequently calling EndpointV2 is by default prohibited and authorization needs to be granted explicitly.

6.2 `init_governance` Can Be DoSed

Security | Low | Version 1 | Code Corrected

CS-SOAG-004

In the SVM Governance OApp, an instruction `init_governance` is defined where a governance PDA can be initialized with an `id` as the only variable part of the seeds. However, the governance initialization is permissionless, hence Sky Governance's call to `init_governance` can be front-run by an attacker which occupies the same `id`. Consequently, the governance initialization could be temporarily DoSed, and it may not be initialized with a desired `id`.

Code corrected:

Since Version 3, the initialization of new governance instance has been restricted to the Governance Program's upgrade authority, which is fully trusted if exists.

6.3 Unchecked Program ID

Informational | Version 1 | Code Corrected

CS-SOAG-005

In the SVM Governance OApp, `lz_receive` expects several accounts for the execution which includes an executable unchecked `program` account. This account, expected to match the `program_id` of the CPI call, however, is neither validated nor used.

Code corrected:

Since Version 2 it is checked that the supplied program id matches the governed program id from the message.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 `dstTarget` May Contain Garbage Leading Bytes

Informational **Version 2** **Acknowledged**

CS-SOAG-006

During `lzReceive()`, the GovernanceOAppReceiver computes the dstTarget by casting bytes32 to an address (bytes20) ignoring the leading 12 bytes. Consequently, if the message is malformed (i.e. by a compromised DVN, or a wrong governance call with `L1GovernanceRelay.relayRaw()`), it will still be accepted.

7.2 Account Lookup Tables Are Not Validated

Informational **Version 1** **Acknowledged**

CS-SOAG-007

Account Lookup Tables (ALTs) are configured for compact address passing in Solana's V0 transactions, which are set during `init_governance` or `set_oapp_config` instructions. However, these accounts of ALTs are not verified to be valid PDAs of the ALT program.

Acknowledged:

Sky states:

```
We don't want to validate ALTs. Even though the admin sets something wrong,
it will just break the view-only lz_receive_types function for the executor.
Execution is permissionless anyway so there may be other actors that can execute
transactions if a malicious administrator forcefully breaks lz_receive_types.
```

7.3 Gas Optimization

Informational **Version 1** **Acknowledged**

CS-SOAG-003

The storage variable `messageOrigin` in the EVM OApp is used to store the `srcEid` and `originCaller` context temporarily for the governed contract during `lzReceive()`. It is always reset to 0 by the end of `lzReceive()` execution. Hence, transient storage could potentially be used to improve gas efficiency, while it is not available for the current compiler setting.

8 Notes

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

8.1 CPI Depth Limitations

Note Version 1

In Solana, CPI (Cross-Program Invocation) has a depth limitation of 4. Hence some operations may not be supported and revert. If users are using an executor program to trigger the Governance OApp's `lz_receive` and further call the EndpointV2 / OFT, there would be 3 CPIs left:

- For configuring OFT on EndpointV2, 3 CPIs are sufficient.
- For configuring OFT directly on the OFT itself, 3 CPIs are sufficient.
- For withdraw OFT fees, in case the token has a hook which further triggers another CPI, the transfer will revert due to exceeding 4 CPIs.
- For sending tokens with OFT, at least another 4 CPIs are needed due to the nested calls into OFT, EndpointV2, SendLibrary, and token. Hence this will revert.

If users are triggering `lz_receive` directly without an executor contract, one more CPI will be available and some operations above may become possible.

8.2 Configuration Considerations

Note Version 1

Configuration for Cross-Chain Governance

Typically it is expected that one chain per user of the OApp will serve as the governing chain and that the remaining chains are receiving chains. Otherwise, two competing governances exist. The expected configuration for the respective chains is outlined below:

- Governance Chain: The owner of the OApp and the delegate in the endpoint are set to the governance contract. In case of Sky, that would be `MCD_PAUSE_PROXY` (or another designated contract) on mainnet.
- Relayed-To Chains: The owner of the OApp and the delegate in the endpoint are set to a trusted address. Typically, it is expected that this trusted address is commanded through cross-chain messages by the governance contract on the governance chain. Note that it should be ensured that all governed contracts follow the appropriate validation, see OApp Call Validation.

However, note that there might be message patterns other than sending messages from A to B (e.g. ABA). In such cases, similar access control should be in place to ensure the validity of messages.

For **EVM** chains, it is expected that the governance actions will be calls to relays which hold the privileges on the given relayed-to chain as current infrastructure would not validate the privileges accordingly as outlined in OApp Call Validation. Similarly, that is the case for alternative messaging patterns.

For **Solana**, note that for executing governance actions in the context of Sky the validity of sent messages can only be guaranteed if the respective CPI authority is guaranteed to have initiated the action (e.g. admin of governed programs is the CPI authority, execution queue that governs but receives actions from the CPI authority). However, note that due the constraints the capabilities might be limited (e.g. initializing libraries or setting the messaging library configurations). Thus, the configuration should be considered carefully.

Configuration for OApps and EndpointV2

Both the EVM and SVM Governance OApp require correct configurations to work correctly. In general, this includes:

- Correctly setting peers to whitelist on each chain.
- Correctly setting a DVN configuration, including optional settings such as block confirmations, security threshold, the Executor, max message size, and send/receive libraries. If no send and receive libraries are explicitly set, the Endpoint will fall back to the default settings set by LayerZero Labs. In case LayerZero Labs changes the default settings, the OApps will be impacted and use the new default settings which implies a trust in LayerZero Labs.

For the EVM sending OApps:

- Correctly setting the `enforcedOptions` to ensure users pay a predetermined amount of gas for delivery on the destination transaction. It should be computed such that messages sent from a source have sufficient gas to be executed on the destination chain. Setting a gas limit too small could mean that no executor has an incentive to pay for the delivery of the message at the destination, and the message should either be dropped by the admin, or some executor should execute it at a loss to resume message handling.

8.3 Experimental `lz_receive_types_v2` Feature

Note Version 1

Sky Governance OApp implements `lz_receive_types_v2` instructions which leverage a experimental feature of LayerZero V2 implemented in this open PR. It is assumed that this feature works as documented and the executor follows accordingly.

Since Oct 2nd, 2025 this experimental feature has been merged to main branch, hence expected to be followed.

8.4 Integrations May Be DoSed by Allowlist

Note Version 1

In case the allowlist is enabled on the EVM `GovernanceControllerOApp`, only listed addresses can integrate and send actions. Consequently, these integrators may be DoSed if Sky governance decides to not add them to allowlist if it is activated, or if they are removed from the allowlist later.

8.5 LayerZero V2 Considerations

Note Version 1

Governance and OApp integrators should be aware of the considerations below:

- *Execution Order*: According to the design of LayerZero V2, the delivery of messages on the destination is not guaranteed to be in the same order as they were dispatched on the source. Consequently, in case multiple governance actions are relayed to the same destination, they might be reordered and lead to unexpected results or reverts.
- *Censorship*: Denial-of-Service and censorship are possible since it is not guaranteed the DVN will verify the governance messages in time.

- *Sandwiching*: The execution of receiving messages is permissionless. Hence, anyone can trigger the execution once the message is verified. Consequently, the governance action can be sandwiched by an attacker with other operations for MEVs or attacks with flashloans in particular. This risk should be thoroughly analyzed before a governance action is dispatched.
- *Refund*: When sending messages a refund address can be provided. This refund target should be able to receive the refund. For example, on the EVM OApp (at the time of writing this is the only sending OApp) the refund address should be able to receive native token transfers (e.g. by implementing `receive()` or `fallback()` if it is a contract or an EOA using EIP-7702) and LayerZero token transfers (i.e. non-zero address). Otherwise, sending may revert if there is a refund.
- *Alternative Native Token*: LayerZero implements endpoints with alternative native tokens (e.g. EndPointV2Alt for EVM chains) where the native token has no significant value. Note that chains with such endpoints are unsupported.

8.6 Malicious Change of LZ Token

Note **Version 1**

Governances using the OApp should be aware that paying fees with LZ tokens (sending messages is EVM-only at the time of writing) requires careful considerations for crafting spells as otherwise a potentially hostile LayerZero governance could drain unexpected tokens.

Consider the following spell for sending cross-chain governance messages:

1. Quote the fee in LZ token.
2. Give an approval for `endpoint.lzToken()` to the OApp accordingly.
3. Execute the respective send function accordingly.

Note that the following attack vector might exist if LayerZero governance turns malicious:

1. Spell is scheduled.
2. LayerZero governance updates `endpoint.lzToken()`.
3. The execution fee is increased.
4. The spell executes and effectively drains the treasury (e.g. Sky token)
5. Governance is attacked by LayerZero governance.

Ultimately, governance spells should ensure that:

- The LZ token is the expected one.
- The fees are limited.

To summarize, spells paying in fees in LZ token could potentially be attacked. However, governance can craft resilient spells.

8.7 Message Passing Considerations

Note **Version 1**

The cross-chain messages sent through the OApp should be thoroughly analyzed before sending them. That for example includes validating the *exact* contents of the message.

Generally, the following problems may occur:

- reverting operations

- incorrectly executed operations
- message safety

Potential reasons for that could be:

- Unexecutable messages due to not adhering to the expected formats outlined in System Overview or providing incorrect data for the cross-chain calls (potentially due to state changes).
- State changes on the destination chain may change the expected execution.
- Execution pricing may change and thus lead to execution fees being insufficient.
- Operation pricing may change and thus lead to out-of-gas scenarios.
- Unexecutable or incorrect messages due dependencies of several governance actions due to no enforced execution ordering (see LayerZero V2 Considerations).

On EVM specifically, target contract upgradeability should be considered carefully in addition.

On Solana specifically, additional problems could arise:

- Unexecutable messages due to the transaction / accounts exceeding the size limits.
- The target contract is upgraded by its upgrade authority.

Note that message sending does not validate message correctness and safety. This should be performed by stakeholder off-chain.

Regarding correctness of sent messages, while some sanity checks are in place when sending, some are not in place. For example, on EVM (at the time of writing EVM OApps are the only OApps sending messages), `sendTx()` does not validate the `dstEid` match the `dstTarget` address length, and the dstCallData match the expected data layout.

8.8 Message Verification Info Reliance

Note **Version 1**

The verification of a message on EndpointV2 (`verify()`) is loosely restricted: message from non-existing peers at the moment can still be verified. Namely, for peers ever configured or a peer of address 0, the `_initializable` check will pass.

```
// EndpointV2.sol:_initializable
function _initializable(
    Origin calldata _origin,
    address _receiver,
    uint64 _lazyInboundNonce
) internal view returns (bool) {
    return
        _lazyInboundNonce > 0 || // allowInitializePath already checked
        ILayerZeroReceiver(_receiver).allowInitializePath(_origin);
}

// OAppReceiver.sol:allowInitializePath
function allowInitializePath(
    Origin calldata origin
) public view virtual returns (bool) {
    return peers[origin.srcEid] == origin.sender;
}
```

Consequently, governed contracts and integrators of governance OApp should not rely on the verification information in EndpointV2, since these messages may not pass peer validations.

8.9 OApp Call Validation

Note Version 1

Governed contract should validate OApp calls.

In EVM Governance OApp's `lzReceive()`, a low-level call will be made to the governed contract, which should properly validate the call and context before execution:

- The `msg.sender` is restricted to the OApp, ensuring the call is triggered by a cross-chain message.
- The `messageOrigin` is validated to ensure the `srcEid` is configured (or potentially stricter checks if necessary) and `originCaller` is expected (e.g. L1 governance).
- Since `lzReceive()` is permissionless, the `msg.value` attached should be validated as expected with the `callData`.

Similarly, in the SVM Governance OApp, a CPI call will be made to the governed `program_id`, which should properly validate the call and context:

- The accounts and parameters passed should be validated, especially the signers required.
- The remaining accounts are not validated in `lz_receive` and should be validated if used.

8.10 SVM Signer Impersonation in `lz_receive`

Note Version 1

In the SVM Governance OApp, `lz_receive()` eventually does a CPI call to the program encoded in the governance message, where both `payer` and `cpi_authority` are signers and can be impersonated to execute arbitrary calls. In theory, the SOL and token balances of `payer` and `cpi_authority` could be drained and their privileges could be abused if any. Governance should properly inspect the CPI call to ensure this is not abused; and executor (`payer`) should ensure the execution is as expected with for instance pre- and post-execution checks.

8.11 Upgrade Authority of Solana Programs

Note Version 1

On Solana, programs can have an upgrade authority (typically the account that originally deployed the program), which bears the privileges to upgrade the program code. If the Governance OApp program is intended to be immutable, its upgrade authority should be removed.

In addition, the governance should be careful of the `program_id` when voting and relaying a governance message. In case the `program_id` retains upgradeability and its upgrade authority is compromised or malicious, it can upgrade the program and execute arbitrary logic impersonating the CPI authority.

8.12 `msg.value` Is Not Guaranteed in `lzReceive`

Note Version 1

On the EVM chains, the LayerZero executor is expected to trigger the `lzReceive()` with the intended `msg.value`. However, this is not guaranteed and the executor (or anyone since `lzReceive()` is permissionless) may attach any `msg.value`. Consequently, if a governed call expects or relies on specific `msg.value`, it may not be satisfied and result in unexpected execution or revert.