PUBLIC

Code Assessment

of the Grove ALM Controller Smart Contracts

August 15, 2025

Produced for

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help GroveLabs with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Grove ALM Controller according to Scope to support you in forming an opinion on their security risks.

GroveLabs offers Grove ALM Controller, a fork of Spark ALM Controller, that implements a set of on-chain components of the Grove Liquidity Layer designed to manage and control the flow of liquidity between Ethereum mainnet and L2s by leveraging Sky DSS Allocator.

This review focused on the first version of Grove ALM Controller for which a separate review was conducted, covering functionality added since Spark ALM Controller v1.5.0, in particular the Centrifuge V3 integration.

The most critical subjects covered in our audit are access control, the correct integration with Centrifuge V3. The general subjects covered are gas efficiency, documentation and composability.

Security regarding all the aforementioned subjects is high.

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	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	7
• Risk Accepted	5
• Acknowledged	2

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 Grove ALM Controller 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	11 Aug 2025	1658e2034e0149ae1f5db0692370822008417903	Initial Version

For the solidity smart contracts, the compiler version `0.8.25` was chosen and `evm_version` is set to `cancun`.

Grove ALM Controller is a fork of Spark ALM Controller. The review scope covered the differences between Grove ALM Controller's initial commit and Spark ALM Controller v1.5.0. This report also includes the risk accepted, acknowledged issues and notes from the upstream v1.5.0 audit report as these remain relevant to the code being reviewed. The following files are in scope:

```
src/
    ALMProxy.sol
    MainnetController.sol
    ForeignController.sol
    RateLimitHelpers.sol
    RateLimits.sol
    interfaces/
        IALMProxy.sol
        IRateLimits.sol
        CCTPInterfaces.sol
        CentrifugeInterfaces.sol
        ILayerZero.sol
    libraries/
        CCTPLib.sol
        CentrifugeLib.sol
        CurveLib.sol
        PSMLib.sol
deploy/
    ControllerDeploy.sol
    ControllerInstance.sol
    ForeignControllerInit.sol
    MainnetControllerInit.sol
```

2.1.1 Excluded from scope

All other files are out of scope. In particular, all 3rd-party protocols Grove ALM Controller integrate with are out of scope and assumed to work honestly and correctly as documented, including: Aave, Ethena, Centrifuge, Morpho, Maple, and Curve Stableswap-NG Pools.

For the curve integration, it is assumed only the Stableswap-NG Plain pools will be used. Note that the deployment script is in scope. However, governance should validate the deployment.

In addition, the inherent centralization risks of USDC are out of the scope of this review:

- USDC is deployed behind a proxy, and its implementation can be upgraded by an admin.
- CCTP relies on a set of centralized offchain signers to provide the bridging attestation.

Note that the deployment script is in scope. However, governance should validate the deployment.

2.2 System Overview

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

GroveLabs offers Grove ALM Controller, a fork of Spark ALM Controller, that implements a set of on-chain components of the Grove Liquidity Layer designed to manage and control the flow of liquidity between Ethereum mainnet and L2s by leveraging Sky DSS Allocator.

On each chain, the following contracts are deployed. All of the contracts inherit the standard AccessControl and grant the *DEFAULT_ADMIN_ROLE* role to an *admin* which can configure other roles.

ALMProxy: Entity that holds funds and interacts with external contracts (e.g. DssAllocator, PSM). Thus, it holds the required privileges to interact with other contracts. It exposes `doCall()`, `doCallWithValue()`, and `doDelegateCall()` to the *controller* role for customized executions. `receive()` is also implemented to receive native tokens.

RateLimits: Defines and enforces limits for liquidity flows. Limits can be configured by the *admin*. The rate limit will linearly grow from `lastAmount` with `slope` over the time elapsed (tracked with `lastUpdated`), and is capped `maxAmount`. Limits will be consumed or recharged by the **controller** with `triggerRateLimitDecrease()` or `triggerRateLimitIncrease()`.

Controllers: Dictates which operations an ALMProxy shall perform. Note that multiple controllers could point to the same ALMProxy. **MainnetController** and **ForeignController** are implemented that define operations in the context of respective ALMProxy.They share a similar structure but feature some different third-party integrations and operations. The *admin* can configure the parameters for 3rd-party integrations and setup other roles: *relayer* that triggers ALMProxy executions and *freezer* that can revoke a *relayer* in emergency.

For more details regarding these contracts and the unchanged 3rd-party integrations, please consult the Spark ALM Review. Note the Superstate Integration has been removed.

2.2.1 Centrifuge V3

Grove ALM Controller further integrate with Centrifuge V3. In general, the deposit and redemption (including the respective cancellation) functionalities stay the same as Centrifuge V2, while the following function is added to both MainnetController and ForeignController:

transferSharesCentrifuge: The controller now allows transferring share class tokens to a cross-chain recipient address. The recipient for each destination chain (`centrifugeId`) must be explicitly configured by the *admin* using `setCentrifugeRecipient()`. The transfer amount is rate-limited based on the token and the destination chain. Once the message is processed on the destination chain (`Spoke.executeTransferShares()`), the shares are minted to the recipient.

2.3 Trust Model

ALMProxy: The *admin* is fully trusted, otherwise, it can setup controllers and trigger any calls with the privilege of ALMProxy. The *controller* is also trusted.

In addition, the ALMProxy requires several roles to operate, which are assumed to be setup properly by governance, for instance:

1. It requires `bud` role to swap without fee on DSS LitePSM.
2. It requires `wards` role on AllocatorVault to `draw()` and `wipe()` USDS.
3. It needs sufficient allowance from AllocatorBuffer to move minted USDS.

MainnetController and ForeignController:

1. The *admin* is fully trusted, otherwise they can DoS the controller, or steal the bridged money on the destination domain by changing the mint recipient.
2. The *relayer* is semi-trusted, and they can only change the liquidity allocation in the worst case. The *freezer* is also semi-trusted which can temporarily DoS the controller in the worst case.

Before initializing the contracts, the governance should always carefully examine whether the deployed contracts match the expectations.

RateLimits: The *admin* is fully trusted to configure the limit data and *controller* correctly.

The 3rd-party integration requires an extended trust model:

- It is assumed Grove ALM Controller will not interact with weird ERC-20 (rebasing / low decimals / ...) and ERC-4626 vault (low token decimals / without share inflation protection / ...). Otherwise, for instance, in case an ERC-4626 has low decimals, a *relayer* may amplify the loss due to rounding errors in shares conversion with many calls for ALMProxy on L2s.
- Grove ALM Controller is subject to the inherent risks of these protocols (i.e. risks of upgradeability, RWAs, governance ...) and generally the third party protocols receiving funds are assumed to be non-malicious.

For the shared integration with: Arbitrary ERC-4626, Aave and Aave-like protocols, Ethena, Maple, Morpho, Curve StableswapNG pools, and LayerZero, please consult the Trust Model of Spark ALM Review.

Centrifuge V3: *Centrifuge is assumed to be the only ERC-7540 integrated.* The privileged roles (wards) are fully trusted, in particular:

- The wards of the ERC-7540 vaults are fully trusted, otherwise they can DoS the system by changing the manager or asyncRedeemManager contract.
- The wards of the asyncRedeemManager is fully trusted, otherwise they may 1) stop fulfilling deposits, redemptions, or cancellation requests; 2) fulfilling the requests with bad conversion rate and incur loss to the users.

Further, shares transfer related infrastructures (Gateway, Spoke, Hub...) and the privileged roles are expected to work correctly and honestly. Otherwise, the shares in flight may be stuck due to censorship.

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 our findings. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

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	7

5.1 Centrifuge Conversion Rate May Change Between Request Submission and Execution

Correctness Low Version 1 Risk Accepted

CS-GRVALM-001

When requesting a redemption from a Centrifuge ERC-7540 vault, the rate limit is decreased by an estimation of the withdrawable assets (`convertToAssets(shares)`) based on the latest conversion rate.

However, the conversion rate may change between the redemption request submission and execution, hence the actual withdrawable assets after execution may not match the rate limit decreased at submission time.

5.2 Centrifuge Deposit / Redemption Can Be DoSed by Cancellation

Security Low Version 1 Risk Accepted

CS-GRVALM-002

The *relayers* can request to cancel pending deposits / redemptions on Centrifuge and claim them later once being fulfilled. Note that in case there is a pending deposit cancellation, no new deposit can be made (same for redemption). Consequently, a compromised *relayer* can DoS new deposit requests by triggering a deposit cancellation (same for redemption) until the existing pending deposit is cancelled or fulfilled.

Risk accepted:

GroveLabs is aware of the risk.

5.3 LayerZero Approvals

Correctness | Low | Version 1 | Acknowledged

CS-GRVALM-003

The `OFTReceipt` contains the `amountSentLD` and the `amountReceivedLD` amounts where `amountSentLD` corresponds to the amount actually debited from the user. Hence, `send()` could potentially pull less tokens than approved (e.g. LayerZero dust removal) and pending approvals could exist. Optimally, the approvals should be revoked to prevent dangling approvals.

Note that pending approvals might introduce a corner case if ZRO are held and bridged. Namely, a dangling approval could allow for a `lzTokenFee > 0` to be collected for ZRO OFTs.

Acknowledged:

GroveLabs has acknowledged that dangling approvals may still exist.

5.4 Maple Redemption Can Be DoSed

Security | Low | Version 1 | Risk Accepted

CS-GRVALM-004

Maple redemption can be DoSed by a compromised *relayer* in two ways:

1. Each user can have at most 1 redemption request in MapleWithdrawalManager. Hence a compromised *relayer* can keep triggering dust redemptions and block the legitimate redemptions from honest *relayers*. In this case, the honest *relayers* have to cancel the dust redemptions first before triggering a legitimate one.
2. Requesting a maple redemption will consume rate limit, whereas cancelling a redemption will not recharge the limit. Consequently, if the whole rate limit is consumed by a compromised *relayer*, other *relayers* will not be able to trigger future redemptions.

Risk accepted:

As demonstrated by the test (`Attacks.t.sol`, `test_attack_compromisedRelayer_delayRequestMapleRedemption`): if a malicious *relayer* delays redemption, the *freezer* can remove the compromised *relayer* and revert to the governance *relayer*. This prevents the compromised *relayer* from continuing the attack, allowing the governance *relayer* to cancel and submit the legitimate request.

5.5 Over-reduced Limit in Maple Redemption

Correctness | Low | Version 1 | Risk Accepted

CS-GRVALM-005

In `requestMapleRedemption()`, the redemption limit will be reduced given the conversion rate between the shares and the assets with `convertToAssets()`.

In MaplePool, function `convertToAssets` assumes the pool holds `totalAsset()` without unrealized loss. However, when the redemption is processed with `processRedemptions()`, the withdrawable amount takes the unrealized loss into consideration.

Consequently, there would be a discrepancy between the rate limit decrease and the actual received tokens in the event of unrealized loss.

5.6 Relayer Can DoS SUSDE Unstaking

Security | Low | Version 1 | Risk Accepted

CS-GRVALM-006

In Ethena, two steps are required to convert sUSDe to USDe:

- A cooldown must be initiated first, which (1) burns the shares and credits the USDe to the USDeSilo contract (2) reset the `cooldownEnd` to be `cooldownDuration` from current `block.timestamp`. Note the step (2) will extend any existing cooldown asset to another `cooldownDuration`.
- When the `cooldownEnd` is reached, the sUSDe can be unstaked and the USDe will be credited to a specified receiver.

Consequently, a malicious relayer can keep triggering new cooldowns with as little as 1 wei asset to block previous exits from sUSDe to USDe, hence DoS the sUSDe to USDe conversion.

Note: GroveLabs was aware of this issue. In addition, in case a malicious *relayer* DoSed the sUSDe `unstake()`, the *freezer* will revoke the *RELAYER* role from the malicious *relayer*.

5.7 Revoking Unused Approval

Design | Low | Version 1 | Acknowledged

CS-GRVALM-007

The *freezer* can remove *relayers* from the MainnetController which prevents *relayers* from triggering any more interactions or funds transfers from the ALMProxy.

However, since the Ethena integration requires actions from several external parties (see Allowance For Ethena Minter May Not Be Consumed), the actual transfers of underlying assets to mint or redeem USDe may happen even after the *relayer* is disabled.

Acknowledged:

GroveLabs acknowledged the issue and decide not to change the code since Ethena is fully trusted.

5.8 Inconsistent Bridging Rate Limits

Version 1 | Acknowledged

CS-GRVALM-008

The LayerZero integration implements rate limits per OFT and destination pair. In contrast, the CCTP integration implements a limit per destination and a global CCTP limit (always in USDC).

The LayerZero integration however lacks a notion of global limits per token and, hence, an inconsistency between the bridging rate limits for CCTP and LayerZero exists.

Acknowledged:

GroveLabs has acknowledged the inconsistency between the bridging rate limits for CCTP and LayerZero.

6 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.

6.1 Allowance For Ethena Minter May Not Be Consumed

Informational | Version 1 | Acknowledged

CS-GRVALM-009

The integration with Ethena minter for USDe minting and burning requires external parties' (delegated signers, Ethena minter and redeemer) actions. The *relayer* can only trigger the `approve()` from the ALMProxy and expect the consecutive actions will be completed by the external parties.

In the following cases the allowance may not be fully consumed:

- The delegated signers sign orders with smaller volume which do not consume all the allowance.
- The Ethena minter or redeemer refuse to submit the order, which blocks the minting or redeeming and does not consume the allowance.
- The expected minting and burning may not be executed successfully due to the restrictions on Ethena minter such as the volume exceeds per block limit.

Consequently, the actual amount used in the interactions may be less than the amount tracked by the rate limit.

Acknowledged:

GroveLabs acknowledged the issue and decide not to change the code.

6.2 Maple Manual Withdraw May Be Enabled

Informational | Version 1 | Risk Accepted

CS-GRVALM-010

A maple redemption requires two steps:

1. The user submits a redemption request.
2. The privileged redeemer processes the request.

In a typical path, no more user interactions are required after step 1, and the underlying tokens will be automatically sent to the user in step 2.

However, in case manual withdrawal is enabled for the user, another call to `MaplePool.redeem()` must be initiated to fulfill the withdrawal and trigger the underlying token transfer.

Note that manual withdrawals can only be enabled by the privileged roles (pool delegator and protocol admins) of MapleWithdrawalManager with `setManualWithdrawal()`. In this case, the ALM Proxy has to explicitly call redeem (ALM Controller's `redeemERC4626()`) to finalize the redemption. And this requires a `LIMIT_4626_WITHDRAW` configured on the ALM Controller for this MaplePool. In addition, the

manually withdrawable shares will be internally accounted in the MapleWithdrawalManager, hence the share balance of ALMProxy (`balanceOf()`) will not contain this.

6.3 Withdraw From Aave Can Be Blocked By LTV=0 Asset

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

CS-GRVALM-011

When an asset is deposited under a user for the first time, the asset will be automatically configured as collateral if its LTV is non-zero and it is not in isolation mode.

In case a user has an asset enabled as collateral which has `LTV==0`, the user will not be able to withdraw any other assets that has `LTV>0`.

As a consequence, the following theoretical attack is possible:

- An attacker observed an asset that has `LTV>0` is going to be configured to `LTV==0` on Aave.
- It can supply on behalf of the ALMProxy (or send directly) a dust amount of this aToken.
- After the parameter change on Aave, the asset has `LTV==0`. The attacker successfully DoS the ALMProxy, which will not be able to withdraw the desired aToken (i.e. aUSDS, aUSDC...) from Aave.

Note that there is no rate limit and token restriction on function `withdrawAave()`. The *relayer* can withdraw the full balance of the asset with `LTV==0`, which resets the `usingAsCollateral` flag to `false` and recovers the ALMProxy from the DoS.

Acknowledged:

GroveLabs has acknowledged this issue and stated a rate limit for the `LTV=0` asset will be added to withdraw this asset in case this attack happens.

6.4 `msg.value` Validation in `transferTokenLayerZero`

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

CS-GRVALM-012

The relayer attaches `msg.value` to calls to `transferTokenLayerZero()` to pay for the LayerZero V2 fees. Note that there might be two scenarios:

- The relayer does not provide sufficient value (e.g. pricing changed between transaction sending and arrival). Then, if the controller does not hold the relevant native token delta, the call reverts. However, that works as expected.
- Similarly, the relayer might provide a `msg.value` that is too high due to similar reasons. In such cases the native token could be stuck in the controller.

Ultimately, the second scenario could lead to native tokens in the controller. Typically, they will not be used. However, technically the relayer could reuse them to future LayerZero V2 fees. However, refunding the relayer with remaining delta might be more meaningful.

7 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.

7.1 A Compromised Ethena Minter Or Redeemer May Execute A Bad Order

Note Version 1

Ethena's minter and redeemer are two crucial roles that can submit the signed orders to the Ethena minting contract. Since the delegated signers are only semi-trusted and can be malicious, the minter and redeemer are fully trusted to never submit bad orders signed by the malicious delegated signers.

In the worst case if Ethena' minter or redeemer are compromised, they may collude with a malicious delegated signer to execute an order with bad quote that drains the approved USDC from ALMProxy.

7.2 Aave Interprets Uint256 Max Withdrawal as Full Withdrawal

Note Version 1

The *relayers* should be aware that Aave will interpret a withdrawal with `type(uint256).max` amount as a full withdrawal with user's balance. The *relayers* should be careful of this special behavior if they are dependent on the input amount.

7.3 Asynchronous Operations May Be Interfered

Note Version 1

The execution of some asynchronous operations may be interfered and unable to finalize due to another operation. For instance, the operation of `prepareUSDeMint()` will be initiated to mint USDe, which simply grants allowance of tokens to be deposited. Before the 3rd-party operation to consume the allowance, another operation, i.e. `swapCurve()`, may use up the tokens. This may cause the 3rd-party operation to fail due to insufficient token balances.

The *relayers* should be careful of the asynchronous operations and avoid the interference of different operations.

7.4 Avoid Morpho Deposit Into Market With Bad Debt

Note Version 1

Upon a deposit into MetaMorpho vault, shares will calculated based on the aggregated expected balance over all the markets in the `withdrawQueue`. In case there is unrealized bad debt in any of the underlying markets, the new deposits will bear this impairment. The *relayers* should monitor the markets conditions and not deposit or reallocate into markets with unrealized bad debt.

7.5 Centrifuge Shares Transfer Refund to ALMProxy

Note Version 1

When transferring Centrifuge shares to another chain with `transferSharesCentrifuge()`, the *relayer* has to attach `msg.value` to pay for the transfer fuel. Note that in case the fuel is excessive, it will be refunded to the ALMProxy instead of the *relayer*.

7.6 Curve Withdrawal Slippage

Note Version 1

When removing liquidity from Curve with `removeLiquidityCurve()` a balanced withdrawal is performed. Note that the performed slippage protection is not strictly necessary. Namely, assuming tokens are pegged, that is due to no negative slippage in terms of "underlying value" being possible.

As a consequence, note that the *relayer* will typically be forced to simulate the transaction to be able to provide rough values suitable to pass the check.

7.7 Inconsistent Swap Rate Limit Decrease for Curve

Note Version 1

When adding liquidity to Curve, a swap can occur. Note that the rate limit adjustment is inconsistent with the adjustment in the swap function. Consider the following example:

1. Assume that swapping 50 token A returns 49 token B.
2. When using the swap function, the rate limit is reduced by 50.
3. Assume that when adding liquidity with 100 token A and 0 token B, the internal swap swaps so that 50 token A and 49 token B are added.
4. The swap performed is effectively the swap from 1.
5. However, the rate limit adjustment will be the average of the input and output deltas and will thus be 49.5.

Ultimately, there can be swap rate limit discrepancies between swap and adding liquidity. However, note that this is intended according to GroveLabs.

7.8 Maple Deposit Ignores Unrealized Losses

Note Version 1

When depositing into the MaplePool, shares are minted assuming there are no unrealized losses from the underlying loan managers, hence this deposit will bear part of the impairment immediately. In addition, withdrawals from MaplePool will bear existing unrealized loss and forfeit the potential recovery of the impairment. The *relayers* should monitor the Maple's loan and unrealized loss status before deposits and withdrawals to avoid loss to the ALMProxy.

7.9 MorphoAllocations updateWithdrawQueue Subject to Front Running

Note Version 1

In Morpho vaults, any user can supply on behalf of the vault. Since `updateWithdrawQueue()` requires the market to be empty, malicious actors can front-run this call, blocking its intended execution. This is a known issue documented in Morpho's documentation. The recommended workaround is for the allocator to bundle a reallocation that withdraws the maximum from the affected market alongside the updateWithdrawQueue call.

The Grove ALM Controller provides separate `updateWithdrawQueue()` and `reallocate()` functions. Although there's no bundled variant that combines them atomically, the expectation is that including both operations within a single transaction will mitigate the frontrunning risk.

7.10 OFT Considerations

Note Version 1

Governance should be aware that certain OFTs are not supported. Below is a list of considerations to make when adding support for an OFT:

- OFTs that try to pull more than it was specified are not supported due to lack of sufficient approval.
- OFTs could try to burn (without approval) more than it was specified are generally not supported. If more would be burned than specified, the rate limit accounting could be incorrect. Thus, such OFTs should not be added.
- OFTs with inherent rate limits could lead to unsuccessful operations. More specifically, the executions could revert due to OFT rate limits.
- OFTs should be ensured to follow the OFT standard correctly. Additionally, the underlying token should not be allowed to change or similar as this could lead to rate limit violations.
- The gas cost for configured `destinationEndpointId` should be carefully monitored to ensure that the hardcoded value of `200_000` is sufficient.

7.11 OFTs With Mandatory Fee in lzToken Are Not Supported

Note Version 1

Function `transferTokenLayerZero()` queries the fees with `quoteOFT()` prior to `send()` to prepare the fee payment by attaching required native tokens. Even though it quotes OFT with flag `_payInLzToken=false`, it does not guarantee the fee contains 0 lzToken. Hence, in case part of the fee must be paid in lzToken, `send()` will fail due to insufficient approval of lzToken.

In summary, OFTs that always require part of the fee in lzToken are not supported by Grove ALM Controller hence should not be used.

7.12 Special Cases Handling

Note Version 1

The ALM's functionality can be extended by allowing new controllers. Some currently unresolvable scenarios, could be resolved in the future if needed. For example:

1. Assume it is desired that for an L2, all funds are bridged back to mainnet. However, in case the PSM3 never holds sufficient USDC to bridge back to L1, funds will remain on L2. As a result, another controller could be whitelisted that initiates redeeming the PSM shares against the other two assets to then bridge them back to mainnet through the respective bridges.
2. The mint recipient for CCTP could be blacklisted. That effectively could DoS the USDC bridging. In that case, a new controller could be added that allows calling CCTP's `replaceDepositForBurn` to resolve the issue.

Ultimately, some unlikely (and intentionally unhandled) issues may arise with the existing controllers. To resolve such issues, new controllers can be added.