PUBLIC

Code Assessment

of the SVM ALM Controller Smart Contracts

January 05, 2026

Produced for

by

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 17
4 Terminology 18
5 Open Findings 19
6 Resolved Findings 21
7 Informational 29
8 Notes 36

1 Executive Summary

Dear all,

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

Keel Finance implements SVM ALM Controller, a Solana program for the Keel capital engine designed to manage and control the flow of liquidity allocated by Sky.

The most critical subjects covered in our audit are functional correctness, secure integration with 3rd-party protocols, and access control.

The general subjects covered are precision of arithmetic operations, front-running, and code complexity.

In summary, we find that the codebase provides a good level of security. Note active monitoring is required during the project lifetime to ensure a secure integration with 3rd-party protocols, for more considerations please refer to the Notes section.

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	10
• Code Corrected	8
• Code Partially Corrected	1
• Risk Accepted	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 SVM ALM Controller repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

The following rust files are in scope:

```
program/src/
    account_utils.rs
    constants.rs
    entrypoint.rs
    enums/
        controller_status.rs
        integration_config.rs
        integration_state.rs
        integration_status.rs
        integration_type.rs
        mod.rs
        permission_status.rs
        reserve_status.rs
    error.rs
    events.rs
    instructions.rs
    integrations/
        atomic_swap/
            atomic_swap_borrow.rs
            atomic_swap_repay.rs
            config.rs
            initialize.rs
            mod.rs
            state.rs
        cctp_bridge/
            cctp_state.rs
            config.rs
            cpi.rs
            initialize.rs
            mod.rs
            push.rs
            state.rs
        drift/
            balance.rs
            config.rs
            constants.rs
            cpi.rs
            initialize.rs
            math.rs
```

```
            mod.rs
            pdas.rs
            protocol_state.rs
            pull.rs
            push.rs
            shared_sync.rs
            sync.rs
            utils.rs
        kamino/
            balance.rs
            config.rs
            constants.rs
            cpi.rs
            initialize.rs
            kfarms_protocol_state.rs
            klend_protocol_state.rs
            mod.rs
            pdas.rs
            pull.rs
            push.rs
            push_pull_accounts.rs
            shared_sync.rs
            sync.rs
        lz_bridge/
            config.rs
            cpi.rs
            initialize.rs
            lz_state.rs
            mod.rs
            push.rs
            reset_lz_push_in_flight.rs
            state.rs
        mod.rs
        shared/
            lending_markets.rs
            mod.rs
        spl_token_external/
            config.rs
            initialize.rs
            mod.rs
            push.rs
            state.rs
    lib.rs
    macros.rs
    processor/
        emit_event.rs
        initialize_controller.rs
        initialize_integration.rs
        initialize_reserve.rs
        manage_controller.rs
        manage_integration.rs
        manage_permission.rs
        manage_reserve.rs
        mod.rs
        oracle/
```

```
            initialize_oracle.rs
            mod.rs
            refresh_oracle.rs
            update_oracle.rs
        pull.rs
        push.rs
        shared/
            account_checks.rs
            data_utils.rs
            emit_cpi.rs
            mod.rs
            pda_utils.rs
            rate_limit_utils.rs
            token_extensions.rs
        sync_integration.rs
        sync_reserve.rs
    state/
        controller.rs
        discriminator.rs
        integration.rs
        keel_account.rs
        mod.rs
        oracle/
            account.rs
            mod.rs
        permission.rs
        reserve.rs
```

As of Version 2, the following file has been added to the scope:

```
program/src/math.rs
```

As of Version 2, the following files have been removed from the scope:

```
program/src/
    account_utils.rs
    integrations/drift/math.rs
    processor/claim_rent.rs
```

As of Version 3, the following file has been added to the scope:

```
programs/src/integrations/cctp_bridge/constants.rs
```

V	Date	Commit Hash	Note
1	26 Oct 2025	e087e3562b7a017e74e19cd348ef26dee8a35900	Initial Version
2	19 Nov 2025	980c7ff41ff534c5b0ea9e2603604d24a13c77c6	After Intermediate Report
3	23 Nov 2025	1f362e0ab976571814a7adf3cc1492baee94771e	Release v1.0.1

4	23 Dec 2025	9d60efc524452df08b556a038be344cb4012c61e	Add Confidential Transfer Fee Config

For the solana programs, the Pinocchio library `v0.9.2` was used and the rust version `v1.90.0` was chosen.

At the time of writing (November 2025), Agave release `v3.0.10` is used on Solana Mainnet Beta. This review cannot account for future changes and possible bugs in Solana and it's libraries

The review considers the integration with Kamino protocol at release `v1.12.7`, and Drift protocol at `v2.148.0-beta.1`.

2.1.1 Excluded from scope

Generally all other files are out of scope.

The following dependencies are assumed to be correct and work as documented:

- The Pinocchio library.
- The associated token programs, SPL Token and Token 2022 programs.
- The 3rd-party programs integrated, in particular, CCTPV1, LayerZero OFTs and other components (Endpoint, Library, Executor), Kamino protocol, Drift protocol.

The Solana blockchain is assumed to work as documented and its timestamp is assumed to be positive only.

2.2 System Overview

This system overview describes the latest version (Version 4) 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.

Keel Finance offers SVM ALM Controller, a Solana program for the Keel capital engine designed to manage and control the flow of liquidity allocated by Sky. It provides several integrations with DeFi protocols and bridges.

The Solana program is built with the Pinocchio framework. On Solana, the program and the actual data storage are separated into different accounts, where PDA (Program-Derived Address) is commonly used for data management and authentication. For simplicity of notation, PDAs will be in *italic* text in the following sections.

2.2.1 Controller

The SVM ALM Controller program allows initializing multiple SVM ALM *Controller* instances that are identified by distinct IDs. Each *Controller* is associated with a *Controller Authority* derived with its address that will be the signer of CPIs performed by the controller.

Note that along with its *Controller Authority* address, a status is stored which can be either:

- Frozen: Disallows all operations except unfreezing and very few other instructions such as oracle refreshes.
- Active: Allows all operations.
- PushPullFrozen: Disallows any kind of fund allocation (i.e. moving funds disabled).
- AtomicSwapLock: Disallows most operations during an atomic swap.

The following instructions are provided:

1. `process_initialize_controller()`: Permissioned function that only `KEEL_DEPLOYER_MSIG` can call and that creates a new *Controller* with a unique ID with an active permission manager *Permission* account granted to a specified authority.
2. `process_manage_controller()`: Changes the status of a *Controller*. Note the following:
    - Only the unfreezer role can set the status to Active again.
    - Only the freezer role can set the status to Frozen or PushPullFrozen.
    - If the status was Frozen, the status can only be set to Active.
    - Note that the status AtomicSwapLock cannot be set with this instruction.
1. `process_claim_rent()`: Allows the reallocator role to transfer the SOL balance (e.g. rent refund from Drift v2) held by the *Controller Authority* to an **arbitrary** destination. Note that it is not expected that governance or anyone else bridges SOL to ALM controllers (e.g. native drop).

Further, note that `process_emit_event()` is provided but is only intended for self-invocations. More specifically, it is restricted to the *Controller Authority* and intended to emit CPI events. Events are typically emitted after any operation.

Note that all other accounts are attached to a given controller.

2.2.2 Account Activity

Most other accounts implement a status flag where the account can be:

- Suspended: The account shall not be used (however, accounts are not closed).
- Active: The account is active and can be used.

In the subsequent sections, functionality will be listed where the status can be changed.

2.2.3 Permission

Multiple *Permissions* can be attached to one *Controller*. Each *Permission* is derived with the *Controller* address and the permission authority (i.e. privileged address) and stores the flags for access control as well as the Account Activity status.

Note that for each *Controller*, an initial permission manager *Permission* that can manage other *Permissions* is initialized upon a *Controller* 's creation.

`process_manage_permission()` allows managing permissions in the following way:

- Permission managers: Can create new *Permission* accounts or can update the permission flags or the status of existing ones. However, they cannot suspend themselves or remove their own permission manager privileges.
- Suspenders: Can suspend permissions. However, they cannot suspend permission managers. Further, note that most arguments are discarded (only the status argument is validated to be the expected one).

Permissions should be present with its authority's signature for most operations on the *Controller* since they are checked for access control. Note that the only exception where access control is defined differently is *Oracle*.

2.2.4 Rate Limit

Outflows from the controller are rate limited to reduce the potential damage an allocator can cause. More specifically, rate limits are enforced twofold on the outflow per Reserve and per Integration.

Rate limits are defined by the following configuration:

- `rate_limit_slope`: Defines the replenishment rate per day.
- `rate_limit_max_outflow`: Defines the maximum available amount that can be consumed. Setting it to `u64::MAX` (up to 18 billion tokens with 9 decimals) effectively switches off the rate limit in practice and will not replenish the available amount with the slope. Note that in- and outflows will adjust the available amount accordingly.

The rate limit is updated as follows:

- Rate Limit Refresh: Replenishes the available outflow according to the rate but caps the value with the maximum. Additionally, stores a remainder (to prevent DoS) and updates the last update time and slot accordingly.
- Inflow: Replenishes the available outflow by the inflow but caps with the maximum.
- Outflow: Reduces the available outflow amount by the outflow and typically raises an error if an underflow occurs. Note that underflows are permitted for *Reserve* under certain conditions.

Typically, the relevant rate limits are refreshed before any integration or reserve interaction. In contrast, in- and outflow are applied typically at the very end of any integration or reserve interaction.

2.2.5 Reserve

Each *Reserve* is derived with a unique SPL Token / SPL Token 2022 instance (distinguished by the *Mint* account). Linked to an *ATA* (Associated Token Account) and owned by the *Controller Authority*, it is designed to keep track of token balances held by the controller and to enforce token-based Rate Limit for outflows. Besides the rate limit state/configuration and the last tracked balance, the Account Activity status is stored accordingly.

Reserve can be managed by the reserve and integration manager role with the following instructions:

1. `process_initialize_reserve()`: Create a *Reserve* for a given token mint with no or whitelisted extensions. The *ATA* will be created, the initial rate limit will be configured, and the initial balance will be synced with the *ATA*.
2. `process_manage_reserve()`: Updates the rate limit configurations (refreshed before) and the status.

Reserve can be synchronized permissionlessly with the following actions:

1. `process_sync_reserve()`: Synchronizes the balance with the *ATA*. Note that the reserve is typically synced before any other relevant interaction, too.

2.2.6 Integration

Each *Integration* is derived with the *Controller* address and a hash of the integration config. The integration config defines a specific integration operation (e.g. depositing USDS into Drift for a given subaccount). Further, the Account Activity mechanism is implemented accordingly, and *Rate Limit* is enforced on outbound flows.

The following instructions for managing integrations, restricted to the reserve and integration manager role, are provided:

1. `process_initialize_integration()`: Create an *Integration* for the given config and initialize the state to the default. Further, some other integration-specific logic might be present.
2. `process_manage_integration()`: Updates the integration's status, description, and rate limit (refreshed before).

Generally, the integration interaction logic is defined by the following functions:

1. `process_push()`: Pushes tokens out of a *Reserve* to a target (i.e. integration). The following access control is enforced typically:

    - Direct token transfers: Token transfer role required.
    - Bridging operations: Reallocator or liquidator role.
    - Investments: Reallocator role required.
    - Swap: Swapper role required.

1. `process_pull()`: Allows reallocators and liquidators (depending on whether the integration is liquidateable) to pull tokens from an integration to a *Reserve*.
2. `process_sync_integration()`: Permissionless function synchronizing the integration state (e.g. claimable balance). Note that this can be done even for suspended integrations. Further, note that this operation is typically performed at the very beginning of any integration action (similar to reserve synchronization).

However, not all of the above interaction logic is necessarily implemented by integrations. Also note that, wherever available, the integration state is synced before integrating.

2.2.6.1 SPL Token (Legacy & 2022)

The integration implements tokens transfers for both the Legacy and 2022 SPL Token programs. More specifically, an integration allows transferring a fixed (supported) token from the controllers *ATA* to a fixed recipient's *ATA*. Hence, the configuration is defined by the token *Mint* and program as well as the recipient *ATA* and the recipient's public key. The inner *Integration* initialization initializes the configuration (only for supported tokens) accordingly and creates the recipient *ATA* if necessary.

The supported operations are:

1. Push: The internal push function `process_push_spl_token_external()` directly transfers tokens by invoking the `TransferChecked` instruction.

2.2.6.2 CCTP V1

CCTP V1 is supported to bridge tokens cross-chain. More specifically, the integration allows bridging a fixed (supported) token to a fixed destination (domain and address).

Thus, the configuration is defined by the token mint, the destination domain and the destination address but also the minter and message transmitter programs' public keys. The inner initialization initializes the configuration accordingly and ensures that required CCTP V1 accounts are available (*LocalToken*, *RemoteTokenMessenger*).

The supported operation is:

1. Push: The internal `process_push_cctp_bridge()` bridges the funds with an invocation of the `CCTP::DepositForBurn` instruction.

2.2.6.3 LayerZero V2

LayerZero V2 is supported to send tokens cross-chain. More specifically, the integration allows bridging a fixed (supported) token to a fixed destination with a given OFT. Note that tokens with fees are not supported for this integration.

Note that the integration is mainly defined by storing the OFT program and *OFTStore* public keys and the destination EID and address. Due to the potentially arbitrary nature of the OFT program, the integration is also specific to the peer config, token mint and the token escrow public keys.

Due to the limitation of Solana CPI depth, the functionality requires 3 instructions, validated using instruction introspection, to be executed at the end of a transaction as follows:

1. Controller Push (at index **N-3**): The tokens are transferred to the *Permission* authority's *ATA* temporarily to initiate the `OFT:Send` on the controller's behalf. Since this call might be performed as a CPI (instead of a top-level instruction), a flag `push_in_flight` is set to true to prevent stealing funds by calling `Push` multiple times as CPI in one instruction.
2. OFT Send: (at index **N-2**): Top-level instruction only. Invokes `OFT::send` on the expected OFT program with the expected token, destination, recipient and amount matching the ones specified in `Push`. The token source must be the *Permission* authority's *ATA*.
3. Controller Reset LZ Push: (at index **N-1**): Top-level instruction only and permissionless. Resets the `push_in_flight` flag to false. Note that it can be invoked separately but is a no-op in such cases.

2.2.6.4 Atomic Swap

To achieve maximum composability with any exchanges, atomic swap is abstracted as an intent that specifies an input token, an output token, an expiry timestamp, and an oracle (see Oracle) used for slippage checking together with a `max_slippage_bps`. Note that parameters for oracle staleness and price inversion are part of the declared intent which the initialization creates.

The intent can be fulfilled by two steps with the following instructions:

1. `process_atomic_swap_borrow()`: Top-level instruction only and only executable before expiry. Transfers the input token to a recipient account and tracks the input and output token balances of both the vaults and the recipient accounts in state. Further, it is ensured that a repay instruction operates on the same integration and that it is the last instruction in the transaction. Note that the temporary swap state recorded between borrow and repay is also used to ensure a repay only happens after a borrow. Further, no top-level instructions are allowed to call the Controller program between a borrow and a repay. The controller status will be set to AtomicSwapLock in this step to prevent most controller operations.
2. `process_atomic_swap_repay()`: Top-level instruction only. The controller status is checked to be AtomicSwapLock and is reset to Active. The amount obtained is computed as the output token balance delta of the recipient token account. Any unused input token will be refunded to the vault. Note that the amounts are thus explicitly pulled from the recipient token accounts. The actual swap price will be computed based on the vaults' deltas. The swap will only succeed if the actual swap price is within the allowed slippage compared to a fresh oracle price. Note that the oracle might require refreshing before the instruction.

2.2.6.5 Drift v2

To allow integrating with Drift v2, an integration for lending on Drift v2 spot markets is implemented. More specifically, the integration allows lending with a given subaccount to a given spot market. Hence, the configuration stores the spot market index with the subaccount ID which is defined on initialization. The initialization additionally initializes the *UserStats* and *User* accounts (and updates the pool ID if necessary). Also note that the token is fixed for a spot market.

The supported operations are:

1. Push: The internal function `process_push_drift()` deposits the given amount to the given spot market for the given subaccount.
2. Pull: The internal function `process_pull_drift()` withdraws accordingly.
3. Sync Integration: The internal `process_sync_drift()` tracks interest generated for the given integration.

2.2.6.6 Kamino

To allow integrating with Kamino, an integration for lending on Kamino is implemented. More specifically, the integration allows depositing into a fixed market for a fixed reserve (i.e. fixed token) into an obligation with a given ID. Note that, the obligation, can be used multiple times (however, Kamino has a limit of 8 reserves per obligation). Note that the initialization will additionally initialize the required *UserMetadata* and *Obligation* accounts on Kamino.

The supported operations are:

1. Push: The internal function `process_push_kamino()` deposits the given amount to the given reserve for the given obligation. In case the obligation account has been closed in a prior operation, it also re-initializes the obligation account and refreshes it. In case a collateral farm has been deployed, user stakes are automatically refreshed.
2. Pull: The internal function `process_pull_kamino()` withdraws accordingly.
3. Sync Integration: The internal `process_sync_kamino()` tracks interest generated for the given integration. In case a collateral farm exists and remaining accounts are provided, it also harvests a reward token to the controller_authority's *ATA*.

2.2.7 Oracle

Oracle provides validated oracle values retrieved from external sources. Currently, oracles are only used for slippage checks in the Atomic Swap integration. The accounts are derived from arbitrary nonces and multiple *Oracle* accounts can be attached to one *Controller* (permissionless attachment) instance.

Oracle is versioned (currently, only version "1") and wraps an external underlying source and stores:

- Assets: Base and quote asset of the underlying (unvalidated and requires correct setup).
- Feed: Underlying account from which a safe value is read. Additionally, feeds are typed according to which source is integrated.
- Value, Precision and Recency: The value read along with the recency (according to external oracle). The precision is defined according to the external source.

Last, for access control, *Permission* is not used. In contrast, an oracle-specific authority per oracle is defined as part of *Oracle* accounts.

The supported operations are:

1. `process_initialize_oracle()`: Permissionless creation of *Oracle* for an arbitrary controller. Validates *Oracle* according to the given type. Note that the initial value and the last update slot are both 0.
2. `process_update_oracle()`: The authority can update the underlying feed (including the type and precision accordingly) or can change ownership.
3. `process_refresh_oracle()`: Reads the feed and stores the trusted price along with the read recency.

2.2.7.1 Switchboard On-Demand

Switchboard On-Demand Oracles are available for reading price feeds. More specifically, data is read from the `CurrentResult` defined in *PullFeedAccountData* accounts (see Switchboard Github). These results are validated by Switchboard programs against the various parameters defined in the source account and are the most recent result values.

2.2.8 Helpers

Note that various helpers are provided to abstract low-level logic such as deserialization and (partial) validation of input accounts. This section aims to provide a high-level overview of the most relevant helpers.

Macros:

- `define_account_struct` has been implemented to verify the input accounts' properties such as mutability, ownership, signer, and data. Note that additionally an optional signer can be specified (None is represented with the program ID).
- `cpi_instruction` has been implemented to allow simple definitions for external CPIs so that an invoke function is present.

KeelAccount (and related helpers): The trait has been implemented by all the related PDA structs. The trait offers the following functionality:

- Deserialize: Deserializes and input AccountInfo's data into a struct, validating the discriminator, underlying layout and length.
- Save/Serialize: Writes an in-memory PDA struct into an AccountInfo with the expected discriminator.
- Data Layout: The data layout is a 1 byte discriminator appended with the struct data.

PDA Account Implementations:

The following high-level functionalities are implemented for most of the PDA structs, abstracting away the low-level checks:

- Account Initialization: Initializes a struct; verifies the struct's determined PDA matches the input AccountInfo; creates the PDA and saves the initial state into the PDA storage.
- Load and Check: Loads one AccountInfo into an in-memory struct while checking ownership, data integrity, and the linking to the expected controller (and authority).
- Update and Save: updates the in-memory struct with inputs and writes the updates back to PDA storage.

2.2.9 Changelog

In Version 2, `process_claim_rent()` was added to handle transferring out SOL.

In Version 3:

1. A new status AtomicSwapLock is added to the controller to prevent most controller operations between atomic swaps (see Atomic Swap Sandwiching).
2. The creation of new controllers was restricted to `KEEL_DEPLOYER_MSIG`.

In Version 4, ConfidentialTransferFeeConfig is added as a valid extension of Mint accounts. Note this allows a Mint account with TransferFeeConfig and ConfidentialTransferMint to be used per extension combination requirement.

2.3 Trust Model

2.3.1 Internal

The system features the following roles (as documented in the README) with respective privileges and trust assumptions.

Keel PauseProxy: fully trusted; works as the ultimate super admin of the controller instance. It is expected to be the authority of a *Permission* that has all privileges expect the `can_execute_swap`,

`can_reallocate` and `can_liquidate`. In case it is compromised, all the funds held by the controller can be drained.

Freezer Multisig: semi-trusted; granted with `can_freeze_controller`, `can_suspend_permissions`, and `can_liquidate` privileges. It is assumed to perform respective pausing / suspension, withdrawing funds, or sending funds cross-chain in emergency. In case it is compromised, it can temporarily DoS the controller.

Primary Relayer: semi-trusted; granted with `can_execute_swap` and `can_reallocate` privileges to facilitate funds allocations over different integrations. In the worst case, it can incur loss that is defined by the respective slippage configurations. It is expected to be suspended by the Freezer Multisig if compromised.

Backup Relayer: semi-trusted; granted with the same privilege as the Primary Relayer. It is expected to replace the primary relayer in emergency. In the worst case it can incur the same loss as a Primary Relayer.

Below is a list of minimum trust requirements for a given role flag:

- `can_manage_permissions`: fully trusted. Can configure anything.
- `can_invoke_external_transfer`: semi-trusted. Damage up to the rate limits.
- `can_execute_swap`: semi-trusted. Damage up to the rate limits.
- `can_reallocate`: semi-trusted. Damage up to the rate limits.
- `can_freeze_controller`: semi-trusted. Expected to only freeze when necessary in emergencies.
- `can_unfreeze_controller`: semi-trusted. Expected to only unfreeze. However, it is trusted to only unfreeze when allowed. Could reenable vulnerable controllers.
- `can_manage_reserves_and_integrations`: fully trusted. Can maliciously configure integrations.
- `can_suspend_permissions`: semi-trusted. Can revoke roles (exception is `can_manage_permissions`).
- `can_liquidate`: semi-trusted. Expected to liquidate when expected.

Oracle Authority: fully-trusted. Each oracle will have its own authority. That authority could manipulate the price by replacing the underlying price feed with an unexpected one. Thus, atomic swaps could be manipulated. However, `can_manage_reserves_and_integrations` role holders are expected to only use trusted oracles managed by trusted parties or noone.

Additionally, oracles will have an authority set. That authority is fully trusted as it could replace the underlying feed to an unexpected one.

Keel Deployer Multisig: semi-trusted. The `KEEL_DEPLOYER_MSIG` is trusted to initialize controllers if necessary.

2.3.2 External

The following assumptions are made regarding the integration with 3rd-party protocols.

In general, all the 3rd-party program's upgrade authorities (if exist) are semi-trusted. In the worst case the program can be upgraded, or the protocol admins can maliciously configure the protocol to steal funds up to deposited amount per rate limit.

SPL Token (2022): The tokens integrated are assumed to be carefully inspected by the governance. Otherwise, the controller may suffer a loss due to a malicious token extension, for instance the tokens are drained due to a malicious permanent delegate.

Switchboard Pull feed: It is used as the price feed for slippage protections during a swap. The price feed configurations are expected to be inspected offchain by the governance. The price feed's authority in the worst case can manipulate the price updates if compromised to incur a loss during swaps.

CCTPV1: CCTPV1 relies on a set of centralized offchain signers to provide the bridging attestation. They are assumed to deliver the cross-chain message honestly. In the worst case, the funds bridged could be lost.

LayerZero: The integrated OFTs are assumed to be trusted implementations. The LayerZero core components (Endpoint, Send / Receive Libraries, DVNs, and Executors) are assumed to deliver the cross-chain message honestly. In the worst case, the funds bridged could be lost.

Kamino: The Kamino protocol is semi-trusted; In the worst case, the protocol / market can be configured maliciously and incur a loss of deposited funds. Please see Kamino Considerations for more information.

Drift: The Drift protocol admin is semi-trusted; the keepers are untrusted; In the worst, case the protocol / spot market can be configured maliciously and incur a loss of deposited funds. Please see Drift Considerations for more information.

Further, all the markets configured for integration are assumed to be carefully inspected and analyzed by Keel to mitigate the potential manipulation risks.

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:

- 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	2

- Drift Synchronization With Outdated Data Code Partially Corrected Risk Accepted
- LayerZero No Slippage Protection Risk Accepted

5.1 Drift Synchronization With Outdated Data

Correctness Low Version 1 Code Partially Corrected Risk Accepted

CS-KSVMALM-006

The `sync_drift_balance` operation for Drift does not consider the interest generated since the last interaction with the spot market. As a result, the `sync_drift_balance` will operate on stale data and could miss some of the pending interest generated. More specifically, that is due to the SpotMarket's `cumulative_deposit_interest` being potentially outdated. Note that this is the case for all entrypoints.

Code partially corrected:

The issue has been mainly resolved. However, there is a corner case where no interest is generated.

More specifically, `UpdateSpotMarketCumulativeInterest` will not update interest if `state.funding_paused()` is `false` on Drift v2. However, deposit and withdraw will always update the interest as long as `spot_market.is_operation_paused` returns `false` for `SpotOperationUpdateCumulativeInterest` (which is also considered in the invoked function)

Risk accepted:

Keel Finance is aware. However, note that there is no other suitable way of performing an update. Thus, Keel Finance accepts the risk. However, note that in the worst case, only events are emitted incorrectly.

5.2 LayerZero No Slippage Protection

Design Low Version 1 Risk Accepted

CS-KSVMALM-011

The following holds for OFTs:

- The OFT might use less than specified due to dust handling.
- The OFT might send less than specified due to fees or dust handling.

Ultimately, funds could be lost unexpectedly as no slippage protection is implemented.

For example, a relayer could do the following:

1. Specify a small input amount (e.g. 1).
2. The amount sent would be rounded down.
3. Effectively, nothing is bridged.
4. However, the relayer can pocket the received tokens in push.

To summarize, due to fees and dust definitions slippage could be high.

Note that the above is limited:

- Requires a malicious relayer.
- In case of high fees, it additionally requires a flawed whitelisting process of OFTs or a malicious OFT admin that sets high fees.
- In case of dust draining, the impact is further limited to the OFT configuration. More specifically, the maximum dust is `10**(mint_decimals-shared_decimals)-1`. As a result, the maximum damage per iteration is `max_dust * price / 10**mint_decimals` where the price is the price per full token. Below, two examples are listed:
    - For USDS, the maximum dust would be 0, making draining impossible.
    - For some BTC token (8 decimals) bridged with an OFT (typically 6 decimals), the maximum dust is 199. That, at a BTC price of 100k, creates a maximum damage per iteration of 0.199 USD.

Risk accepted:

Keel Finance is aware and accepts the risk as the integration will be primarily used for USDS which is not affected by potential dust exploitation. The configuration of other OFTs will be carefully analyzed.

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	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	8

- Atomic Swap Sandwiching Code Corrected
- Drift Inaccurate Integration Balance Delta Code Corrected
- Drift Incorrect Reserve Account Code Corrected
- Drift Missing State Update Code Corrected
- Frontrunning Controller Initialization to DoS Code Corrected
- Kamino Inaccurate Reward Availability Discovery Code Corrected
- Kamino Latest Release Not Supported Code Corrected
- Kamino Sync Missing Reserve Inflow Code Corrected

Informational Findings	7

- Unsuitable Controller Initial Status Code Corrected
- Atomic Swap Minimum Swap Price Rounding Code Corrected
- Atomic Swapping With Self Code Corrected
- Inconsistent SPL ATA Ownership Declarations Code Corrected
- Kamino Integration Gas Optimizations Code Corrected
- Lack of Events Code Corrected
- Rate Limit Gap Does Not Decrease Remainder Code Corrected

6.1 Atomic Swap Sandwiching

Design Low Version 1 Code Corrected

CS-KSVMALM-001

Atomic Swap can be used to sandwich other instructions to create unexpected results.

1. A governance message is sent to liquidate the USDC reserve and suspend it.
2. The swapper detects the message and performs the following:
    1. Flash borrow all the USDC reserve but leave one 1 wei inside. Borrow to account X.
    2. Execute the cross-chain message. This bridges 1 wei back to mainnet with CCTP and suspends the reserve.
    3. Donate 1 wei to the controller from a second account Y.

4. Flash repay. No token delta is detected. Ultimately, the suspended reserve receives funds accordingly.

Note that the underlying problem is the lack of activeness checks in `process_atomic_swap_repay` for the owned context variables defined in `AtomicSwapRepay`:

- `integration`
- `reserve_a`
- `reserve_b`
- `controller`

However, note that an actual execution of such sandwiching is unlikely due to various required conditions as follows:

1. A relayer is compromised.
2. Such a cross-chain message arrives and the relayer must be the first to execute the message before others (e.g. LayerZero executors).

Code corrected:

A new status flag, `AtomicSwapLock`, was added to the controller.

It introduces the following changes:

1. On borrow, locks the controller accordingly.
2. On repay, requires the controller to be locked and unlocks the controller.
3. **All actions except the following raise an error now accordingly if the status is `AtomicSwapLock`:**
    1. `process_emit_event`: allowed but can only be called through self-CPI.
    2. `process_initialize_controller`: allowed but cannot touch the current controller.
    3. `process_manage_controller`: technically allows changing the status from `AtomicSwapLock`. However, if an instruction changing the status were to be sandwiched, repaying would fail due to atomic repay. Further, it cannot change the status back to `AtomicSwapLock`.
    4. `process_refresh_oracle`: allowed. It is fine assuming the price feed cannot be manipulatd during this transaction.
    5. `reset_lz_push_in_flight`: allowed as it is a no-op.

To summarize, during an atomic swap, no actions on the controller are allowed to manipulate its state until the debt is repaid.

6.2 Drift Inaccurate Integration Balance Delta

Design | Low | Version 1 | Code Corrected

CS-KSVMALM-002

The update of the integration state for Drift Push is inaccurate.

More specifically, the token delta of the `spot_market_vault` is considered. However, the actual delta might be different due to rounding. A more accurate representation would be to consider the delta of `get_drift_lending_balance()` as this represents the claimable balance more accurately.

Code corrected:

Code has been adapted to update the integration `state.balance` with `get_drift_lending_balance()`.

6.3 Drift Incorrect Reserve Account

Correctness Low Version 1 Code Corrected

CS-KSVMALM-003

An incorrect account can be passed to `process_sync_drift()` and lead to an incorrect event emission.

More specifically, the *Reserve* is not validated to match the given *Integration*. As a consequence, `reserve.mint` might be incorrect, and an incorrect event could be emitted.

Code corrected:

The synchronization now always retrieves the mint from the spot market for more accurate results.

6.4 Drift Missing State Update

Correctness Low Version 1 Code Corrected

CS-KSVMALM-004

Various state updates are missing for Drift Push and Pull:

1. Push: Does not update the state according to the synchronization.
2. Pull: Does not update the state according to the synchronization.
3. Pull: Does not update the state according to the generated delta.

Ultimately, later operations may emit incorrect events (e.g. emitting a loss scenario after 3.).

Code corrected:

The code has been adjusted to correctly set the state. More specifically, push and pull both do the following:

```
state.balance = get_drift_lending_balance(spot_market_state, inner_ctx.user)?;
```

Ultimately, the synchronization delta and the integration delta are applied at once.

6.5 Frontrunning Controller Initialization to DoS

Design Low Version 1 Code Corrected

CS-KSVMALM-012

The initialization of controllers is permissionless and the derived address depends solely on an input ID. As a consequence, the initialization could be DoSed:

The ID of controllers is at most 65535. Technically, all IDs could be used, leading to a permanent DoS of initialization. However, the cost of such a DoS is non-negligible and can be circumvented by deploying a new program..

Code corrected:

Code has been corrected and now the instruction `initialize_controller` is restricted to a hardcoded deployer address.

6.6 Kamino Inaccurate Reward Availability Discovery

Correctness | Low | Version 1 | Code Corrected

CS-KSVMALM-008

The integration synchronization function for Kamino, `process_sync_kamino`, fails to detect rewards properly.

More specifically, currently it tries finding the reward info from the reserve *FarmState* with `find_reward_index_and_rewards_available`. That returns the `reward_info.rewards_available` along with the index.

However, `rewards_available` might be an unexpected variable. For example, the value could be 0 while rewards would still be available. Note that the amount describes the distributable amount which will be used in refreshes. For distributable rewards, `rewards_issued_unclaimed` exists. While the difference is subtle, it could be that `rewards_available` remains 0 if no new rewards are added. In contrast, `rewards_issued_unclaimed` will be non-zero (if refreshed before) as long as there is some reward.

With the current implementation some rewards may be unharvestable. Further, note that technically the harvesting CPI could simply always be invoked to prevent this scenario.

Code corrected:

Rewards are now always collected without a `rewards_available` check. Note that relayers can always skip reward collection by not specifying remaining accounts.

6.7 Kamino Latest Release Not Supported

Design | Low | Version 1 | Code Corrected

CS-KSVMALM-007

This review considers the integration with Kamino Lend v1.12.5. However, during the review, Kamino Lend is upgraded with a new release v1.12.6. It allows the creation of many reserves of same mint on the same market, hence the PDA derivation of the following accounts has been changed when a new reserve is created:

- `reserve_liquidity_supply`
- `fee_receiver`
- `reserve_collateral_mint`
- `reserve_collateral_supply`

Consequently, SVM ALM Controller cannot integrate with the newly created *Reserves* due to the PDA checks upon integration initialization.

Code corrected:

The derivation and checks of the affected and used PDAs is removed. Note that Kamino checks those as part of the respective instructions.

Thus, the PDAs created by the newer Kamino versions are supported.

6.8 Kamino Sync Missing Reserve Inflow

Correctness | Low | Version 1 | Code Corrected

CS-KSVMALM-010

When harvesting rewards for Kamino in `process_sync_kamino`, the update of the reserve is missing.

Note that the sequence of events emitted corresponds to:

1. Credit the integration for the Sync.
2. Debit the integration for a hypothetical Withdrawal.
3. Credit the reserve for the hypothetical Withdrawal.

Note that the in- and outflow for the integration is 0. However, the inflow for the reserve is positive.

As a consequence, the tracked reserve balance should be updated. However, it is not. As a consequence, the balance reported through events and the tracked balance may start to mismatch. For example, a balance synchronization would detect an increase and treat it as a donation, leading to double counting in events. Also, note that paired with the balance update the expected rate limit increase is coupled (i.e. `update_for_inflow`).

Code corrected:

The above has been resolved accordingly. Note that the rate limit is also increased for the integration which can be done as well:

```
reserve.update_for_inflow(clock, reserve_vault_balance_delta)?;
integration.update_rate_limit_for_inflow(clock, reserve_vault_balance_delta)?;
```

6.9 Unsuitable Controller Initial Status

Informational | Version 3 | Code Corrected

CS-KSVMALM-028

Controllers can be initialized with the unsuitable status AtomicSwapLock. Note that, effectively, such a controller is unusable due to no operations being available to it and the status being unchangeable due to a lack of permissions.

Code corrected:

Code has been corrected and now the instruction `initialize_controller` checks the initial status cannot be `AtomicSwapLock`.

6.10 Atomic Swap Minimum Swap Price Rounding

Informational **Version 1** **Code Corrected**

CS-KSVMALM-013

The minimum swap price computation in `check_swap_slippage` computes the following (simplified):

```
let min_swap_price = oracle_price
    .checked_mul(BPS_DENOMINATOR.saturating_sub(max_slippage_bps).into())
    .unwrap()
    .checked_div(BPS_DENOMINATOR.into())
    .unwrap();
```

Note that the minimum should optimally be greater than the real minimum swap price. However, that is not guaranteed:

1. The division above rounds down.
2. For inverted price feeds, the `oracle_price` is rounded down.

To summarize, the above should be ceiled divisions to ensure that the real minimum price is respected.

Code corrected:

Ceiled division is now used accordingly.

6.11 Atomic Swapping With Self

Informational **Version 1** **Code Corrected**

CS-KSVMALM-014

Atomic swap's borrow does not explicitly restrict the recipient token accounts to not be the controllers' ATAs. While no abuse is possible, introducing the checks could clarify code.

Code corrected:

The code has been adjusted to return an error if the accounts are equal:

```
if payer_account_a.key.eq(vault_a) || payer_account_b.key.eq(vault_b) {
    msg!("Recipient/payer: account cannot be vault");
    return Err(SvmAlmControllerErrors::InvalidAccountData.into());
}
```

6.12 Inconsistent SPL ATA Ownership Declarations

Informational | Version 1 | Code Corrected

CS-KSVMALM-018

When pushing SPL tokens, `process_push_spl_token_external` will assume that the owner of `PushSplTokenExternalAccounts.recipient_token_account` is one of the token programs. However, in `PushSplTokenExternalAccounts.checked_from_accounts` it implements the clause:

```
&& !ctx
    .recipient_token_account
    .is_owned_by(&pinocchio_system::ID)
```

However, the token account will never be owned by the system due to the ownership declaration within the context. Similarly, the `CreateIdempotent` invocation will be commented to "to create or validate" the ATA for the recipient as it will only be possible to validate it.

Ultimately, context ownership declaration and code are inconsistent.

Code corrected:

The ownership declaration now includes the system ID, too.

6.13 Kamino Integration Gas Optimizations

Informational | Version 1 | Code Corrected

CS-KSVMALM-020

- In `process_pull_kamino()`, a computation of current liquidity value is performed first in `sync_kamino_liquidity_value()` and again for `liquidity_value_before`. The second computation is redundant and can use the return value of the first.
- In `process_pull_kamino()`, the following type casting with `into()` is redundant: `Err(ProgramError::InvalidAccountData.into())`.

Code corrected:

The code has been adjusted accordingly.

6.14 Lack of Events

Informational | Version 1 | Code Corrected

CS-KSVMALM-022

`process_initialize_controller` performs two tasks:

1. Initialize a controller.
2. Initialize the initial permission account for the initial permission manager.

For the controller initialization, the `ControllerUpdate` is emitted through `controller.emit_event` (similar to all other controller updates).

In contrast, no event for the initial permission is emitted. However, for completeness it could be meaningful to emit the `PermissionUpdate` event.

Code corrected:

The missing event is now emitted.

6.15 Rate Limit Gap Does Not Decrease Remainder

Informational | Version 1 | Code Corrected

CS-KSVMALM-029

If the gap between current maximum outflow and the current available outflow is greater or equal to a newly set maximum outflow, the newly set available amount will be 0. However, unexpectedly, the remainder is not reset to 0 in such cases.

Code corrected:

The following is introduced to reset the remainder in case the newly available amount is set to 0.

```
if gap > self.rate_limit_max_outflow {
    // Reset remainder during update
    self.rate_limit_remainder = 0;
}
```

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 CCTPV1 Only Supports Legacy SPL Token

Informational Version 1 Acknowledged

CS-KSVMALM-026

For CCTPV1 integration, the token *Mint* to be bridged is checked to be owned by either the legacy SPL Token or the SPL Token 2022 program. However, the existing CCTPV1 program only supports legacy SPL Token.

7.2 Code Duplication

Informational Version 1 Acknowledged

CS-KSVMALM-015

In various instances, code is duplicated. Below is a non-exhaustive list of examples:

1. Rate limit logic for integrations and reserves is mostly equivalent and could, in theory, be shared.
2. The inner `push` and `pull` logic perform `reserve_a.sync_balance`. However, this could be moved to the outer functions, leading to less code duplication.
3. The outer `push` and `pull` logic is practically equivalent. The only exception is the integration-specific internal logic. Code could, in theory, be shared. Note that such an approach is partially followed by the Kamino integration.
4. `process_push_drift` and `process_pull_drift` are similar to a large degree. More specifically, `checked_from_accounts` is equivalent and shared logic could be moved to `check_accounts` of the configuration. Several other operations up to the CPI are equivalent. Ultimately, code could be shared.
5. While the Kamino integration considers code sharing, it could be even further shared.
6. Note that the above applies to the respective sync functions, too.
7. Further, the Atomic Swap Borrow could in theory be wrapped by an outer push to increase code sharing with other integrations. (Though this might complicate the checks of instruction introspection).

Ultimately, code duplication could be reduced. However, changes require careful thinking about potential extensions.

Acknowledged:

Keel Finance is aware and plans to refactor in the future.

7.3 Drift Inaccurate Integration Delta Event

Informational Version 1 Acknowledged

The event emitted for Drift push and pull can be inaccurate.

More specifically, the token delta, `liquidity_value_delta`, of the `spot_market_vault` is considered. However, the actual delta might be different due to rounding.

A more accurate representation would be to consider the delta of `get_drift_lending_balance()` as this represents the claimable balance more accurately. Note that, for example, the Kamino integration considers the corresponding delta. Ultimately, integrations are inconsistent.

Nonetheless, emitting `liquidity_value_delta` may be meaningful. That is due to `get_drift_lending_balance()` potentially returning zero for small Drift-internal balances (internal balances are scaled). Emitting 0 could signal that no deposit was made depending on interpretation.

Acknowledged:

Keel Finance is aware and acknowledges the finding. However, Keel Finance prefers to keep it as is.

7.4 Drift No Slippage Protection

Informational | Version 1 | Risk Accepted

Given that lenders receive interest and thus the value of the deposited tokens increases. This typically leads to rounding errors (and could be amplified by inflation attacks).

Note that as a consequence, a deposit of 1 may lead to being eligible for 0 (or other values). Ultimately, slippage checks are missing to prevent unwanted, potentially slow draining, of the ALM controller's funds.

Note that the above is limited:

- Requires a malicious relayer.
- Theoretically the Drift admin could inflate the share price by donation with the privileged instruction `handle_deposit_into_spot_market_vault`. However, this requires a malicious admin, which can steal the funds even simpler by upgrading the program.
- Assuming an honest Drift admin, slippage for deposits is hard to leverage for attacks. Generally, it can be easily achieved for tokens with more than 9 decimals. However, for tokens such as USDS, a manipulation of `cumulative_deposit_interest` for better effectiveness is required. However, that is non-trivial as it requires, for example, borrowing at full utilization for a longer time period. Potentially, there could be more effective ways. More precisely, a 100% slippage scenario, for example, can be created as soon as `10**(19-token_decimals) < cumulative_interest` holds. Note that `cumulative_interest >= 10**10` typically holds.
- Note that slippage for withdrawals is similarly difficult.

However, note that there might be other potential manipulation vectors (or new ones in the future).

Risk accepted:

Keel Finance is aware and accepts the risk and responded:

```
The slippage attack is unlikely for multiple reasons including relayer needing to be compromised along side
an Integration that has been set up for a Drift pool that has minimal tokens and thus open to manipulation.
This is something that is prevented via configuration due to Keel only being able to set up Integrations for
markets that have been analyzed by the risk team
```

7.5 Frontrunning Oracle Initialization to DoS

Informational | Version 1 | Acknowledged

CS-KSVMALM-019

The initialization of oracles is permissionless and the derived address depends solely on an input nonce. As a consequence, the initialization might be DoSed. Thought the nonce of oracle is bytes 32 and eventually a nonce might be available.

7.6 Inaccurate Error

Informational | Version 1 | Acknowledged

CS-KSVMALM-016

Below is a list of inaccurate errors:

1. In `process_emit_event()`, when unpacking the accounts, it will return an `Err(NotEnoughAccountKeys)`. This message could be inaccurate if there is more than 1 account in the slice.
2. `DriftConfig.check_accounts()`, the `spot_market_index` is validated against the config. If they do not match, `Err(ProgramError::InvalidAccountData)` is returned. This message could be inaccurate since the argument is validated.

Code partially corrected and acknowledged:

1. Not corrected. Keel Finance acknowledges that the error might be inaccurate. However, prefers to keep the code consistent with other Solana programs that raise the same error in such cases.
2. Corrected. The error used is now `InvalidInstructionData`.

7.7 Inconsistent Code

Informational | Version 1 | Acknowledged

CS-KSVMALM-017

The code can sometimes be inconsistent. Below is a non-exhaustive list of such inconsistencies:

1. The status of a controller `Controller.status` is disallowed to do a no-op update:

   ```
   if self.status == status {
       msg!("Controller status must change");
       return Err(ProgramError::InvalidArgument.into());
   }
   ```

   Ultimately, this is inconsistent with other management operations as they allow setting the current status (i.e. no-op updates).
2. All accounts except `Reserve` implement a `update_and_save` function while `Reserve` implements an `update` function. While the account is saved in `process_manage_reserve`, it is unnecessarily different from all other accounts.
3. `update_for_inflow` and `update_rate_limit_for_inflow` could leverage `min` functionality for more concise code.

4. The program constants for `CCTP_MESSAGE_TRANSMITTER_PROGRAM_ID` and `CCTP_TOKEN_MESSENGER_MINTER_PROGRAM_ID` are defined in the global `constants`. In contrast, the Kamino and Drift constants are kept locally within the integration's constants.
5. Various discriminators could be computed with `anchor_discriminator` but are not:
    1. CCTP: `DISCRIMINATOR` in `cctp_state` and the `discriminator` in CCTP's `cpi`.
    2. LZ: `DISCRIMINATOR` in `OftSendParams`, `OFTStore` and `PeerConfig`.
6. The ownership declarations (`@owner`) in the instruction context typically always include all possible owners of an account (for relevant accounts). However, they are missing in various declarations. Ultimately, the implementation is inconsistent. Below is a non-exhaustive list:
    1. `InitializeReserveAccounts`: for `mint` and `vault` an ownership tag could be added.
    2. `InitializeCctpBridgeAccounts` & `PushCctpBridgeAccounts`: For `remote_token_messenger` and `local_token` the respective ownership tag could be added. That could reduce code in the respective `checked_from_accounts`.
    3. `InitializeAtomicSwapAccounts`: The ownership flags could be added for the input tokens.
    4. `InitializeDriftAccounts`: Similarly that is the case. Multiple owners could be defined.
7. The `pda.rs` files typically hardcode the `program_id` argument. In contrast, the one for Kamino expects arguments.
8. `process_sync_integration` is unnecessarily inconsistent in terms of syncing the reserve with `sync_balance`.

Code partially corrected and acknowledged:

1. Not adjusted. Keel Finance prefers the current implementation.
2. Not adjusted. Keel Finance prefers the current implementation.
3. Corrected.
4. Corrected.
5. Corrected.
6. Corrected. Generally, it could be more extensive but the usage of the tag has been improved. For example, that could be in `PushCctpBridgeAccounts`, `SyncDriftAccounts`, `InitializeKaminoAccounts`, `PushPullKaminoAccounts`,
7. Corrected.
8. Not corrected. While not needed by the Drift v2 integration, and thus not necessarily inconsistent, the `reserve` could be part of the inner context and could be provided when needed by an integration.

7.8 Kamino Integration Missing Slippage Check

Informational **Version 1** **Risk Accepted**

CS-KSVMALM-009

In general, no slippage checks are performed when depositing into or withdrawing from a Kamino Reserve. In case the price per share (liquidity per collateral) can be manipulated, a loss may be incurred to the controller. For instance, if the price was inflated, a deposit would result in less shares due to rounding errors.

Note that the above is limited:

- Requires a malicious relayer.
- Note that the attack is limited to smaller rounding errors. Namely, that is due to Kamino typically recomputing the amount to deposit from the computed collateral amount. Thus, it generally becomes difficult to create such scenarios.

Nonetheless, there might be higher slippage scenarios. However, ultimately, they are rather unlikely.

Risk accepted:

Keel Finance is aware and accepts the risk and responded:

```
The slippage attack is unlikely for multiple reasons including relayer needing to be compromised along side
an Integration that has been set up for a Kamino reserve that has minimal tokens and thus open to manipulation.
This is something that is prevented via configuration due to Keel only being able to set up Integrations for
markets and reserves that have been analyzed by the risk team
```

7.9 Kamino Passing Shares for Pull

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

CS-KSVMALM-021

`process_pull_kamino()` takes an underlying token amount as argument. That amount is then later converted to shares:

```
let collateral_amount = kamino_reserve_state.liquidity_to_collateral(amount);
```

Note that this is the only usage of the argument.

Thereafter, the `collateral_amount` will be used. Consequently, `collateralAmount` could be directly passed and code could be simplified.

Acknowledged:

Keel Finance is aware but prefers it so that the interface in terms of arguments is consistent with other integrations.

7.10 Redundant Operations

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

CS-KSVMALM-023

Various operations can be considered redundant. Below is a non-exhaustive list of potentially unnecessary logic duplication:

1. Various operations implement check below manually:

   ```
   inner_ctx.vault.key().ne(&reserve.vault)
   ```

   However, note that typically this will be checked by a latter `sync_balance` operation. Examples, include `process_push_spl_token_external`, `process_push_cctp_bridge`, `process_push_lz_bridge` and `process_atomic_swap_borrow`.

2. `CctpBridgeConfig` stores the public keys of `cctp_token_messenger_minter` and the `cctp_message_transmitter`. However, these are fixed constants. In `PushCctpBridgeAccounts.checked_from_accounts` the provided context is validated against the constants with `@pubkey` but is also validated against the config. Ultimately, storing the constants in the config is redundant.
3. Often, the integrations configs (states and args) are matched multiple times to be the expected config in various callpaths (e.g. `process_push_cctp_bridge`).
4. The `market_index` for drift push/pull is not necessarily needed as it can be retrieved from the config.

Acknowledged:

Keel Finance is aware and responded:

```
We plan to leave this as is for now and are likely to refactor the integration handling in the future to
make it more comprehensive and remove redundant operations.
```

7.11 Unnecessarily Large Maximum Slippage

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

CS-KSVMALM-024

`process_initialize_atomic_swap` allows `AtomicSwapConfig.max_slippage_bps` to be above 10000. All values above 10000, are treated as 100%. However, it could be meaningful to restrict the allowed values to be at most 10000 so that slippages are defined more clearly.

Acknowledged:

Keel Finance acknowledged the behavior and specified:

```
The allowed slippage is configured by a trusted authority during Integration initialization.
In order to maintain flexibility, the program does not enforce a max slippage amount.
```

7.12 Unused Functionality

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

CS-KSVMALM-025

Various errors in `errors.rs` are unused:

1. `InvalidReserve`
2. `DataNotChangedSinceLastSync`
3. `InvalidIntegrationArgs`
4. `SerializationFailed`

Various functions are unused:

1. `account_utils.rs`: `account_is_uninitialized()` is unused making the whole file unused.
2. `processor/shared/account_check.rs` has various unused functions:

1. `verify_signer`
2. `verify_system_program`
3. `verify_owner_mutability`

Code partially corrected and acknowledged:

For unused errors, Keel Finance chose to keep them.

The unused functions have been removed.

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 Active Monitoring Required

Note Version 1

SVM ALM Controller is designed to integrate with many 3rd-party protocols on Solana. Since most programs are upgradeable (e.g., Kamino Lend was upgraded twice during the review in November 2025), active monitoring is required to ensure a correct and secure integration during the project lifetime. In case a protocol's upgrade breaks backward compatibility, some integration operations may be DoSed or have unexpected behavior, and the integration state may not be synced correctly. The privileged roles are required to actively monitor such scenarios and act accordingly to minimize the loss.

8.2 Atomic Swap Considerations

Note Version 1

- Atomic borrow and repay can actually be performed with two different *Permissions* if both can execute swap.
- During Atomic repay, the reserve, and integration status are not rechecked to be active. If any of them is "suspended" in between, the swap can still succeed.
- Governance should be aware that if circular swap configuration exist, it may incur loss by swapping back and forth if the corresponding *Permission* authority is compromised.
- In theory, the oracle returned price could be negative since it is `i128`. In case a negative price is returned, slippage check will always be satisfied with any output token amount.
- A swap intent is defined as any other integrations with a rate limit that replenishes over time. If a swap should be limited to a fixed amount, the integration's limit refilling slope should be 0. Further, the swap may only be partially executed.

8.3 Difference With EVM ALM Controller

Note Version 1

SVM ALM Controller shares a similar design with EVM ALM Controllers (e.g. Spark) with the following differences:

1. The rate limiting works slightly different (i.e. the remainder and gap design are exclusive in SVM ALM Controller).
2. SVM ALM Controller implements "synchronization" functionalities to update the state with donations, cross-chain bridged funds, and yield, while EVM controllers do not.
3. Access control is slightly different. In EVM controllers, the relayers can manage all integrations while in SVM ALM Controller permissions are granted with respective flags.
4. EVM controllers features admin-defined slippage parameters for integrations in general while SVM ALM Controller does not.
5. SVM ALM Controller can be fully suspended while EVM controllers cannot.

8.4 Drift Considerations

Note **Version 1**

Governance should be aware of the below when whitelisting Drift integrations:

1. Inactive positions can be closed. Governance should ensure that the reallocator does not allow accounts to be closed as this might effect the integration.
2. Further, the pool ID of spot markets might change. Governance should ensure that this is not expected to happen.
3. A parameter `maxNumberOfSubAccounts` restricts the total number of sub accounts that can be created, in case this is reached on Drift, no new sub accounts can be created by the ALM controller.
4. An `init_fee` is charged to the *Permission* authority when initializing the sub account, note this fee can be adjusted by the Drift admin.
5. Additionally, there are various variants of pausing.

8.5 Governance Considerations

Note **Version 1**

Since Keel Proxy resides on Ethereum and relies on LayerZero cross-chain messaging to manage the ALM controller, governance should be aware of the following:

1. Ordering: If multiple messages are relayed together, they may not be executed in the expected order on Solana. In case there is an ordering dependency between them, this may not be satisfied.
2. Sandwiching: Since verified governance messages are executed permissionlessly, it should be properly analyzed that it cannot be sandwiched maliciously (e.g. sandwich a program upgrade for unexpected execution).

For more considerations please refer to the Sky LayerZero Governance OApp Review.

8.6 Kamino Considerations

Note **Version 1**

- Upon Kamino integration accounts initialization, if the reserve has no collateral farm created yet, the obligation farm account will not be created. After the collateral farm is created, anyone can create the obligation farm for the controller since `init_obligation_farms_for_reserve()` is permissionless.
- The deposit and withdrawal on Kamino Lend requires the reserve is not stale, hence `refresh_reserve()` should be called before any deposit or withdrawal in the same slot.
- When synchronizing the Kamino Lend integration state, reward tokens can also be claimed. In case multiple reward tokens are available, they can be claimed with multiple synchronizing calls. Further, it is not required to claim the rewards during a synchronization.
- If the reward token mint matches the reserve mint of this integration, events will be emitted; however, no events will be emitted if the reward token mint matches another reserve mint. Further, the reserve mint will not be synchronized for incoming rewards.
- The referrer should be carefully chosen as it cannot be changed afterward.

- Note that rewards harvested that do not equal the lent token, will be later interpreted as donations.

8.7 Kamino Farm Reward Mint Is Unchecked

Note **Version 1**

When synchronizing the Kamino Lend integration state, reward tokens, if they exist, can be claimed to its corresponding ATA. However, the reward token mint is not validated with the `VALID_MINT_EXTENSIONS`. Though a reserve cannot be created for such a token with invalid extensions hence the token cannot be used in any existing integrations.

8.8 LayerZero Bridge Considerations

Note **Version 1**

Privileged addresses should be aware that `OftSendParams` are not validated as restrictively as they could be:

- Compose Message is not validated. No recipient with supporting compose messages shall be whitelisted for the integration.
- Fees are not validated. However, ultimately that is the responsibility of the sender.
- Slippage is not validated.
- Options are not validated.

Further note the following about the integration:

- The integration continues to be valid if the `peer_config` is changed.
- `reset_lz_push_in_flight` can be permissionlessly called even if there is no lz push instruction in the same transaction since it does not require `push_in_flight` is true. And any redundant call data will be discarded. Though it is a no-op.

8.9 No Self-Transfers

Note **Version 1**

Governance should be aware that self-transfers should not be whitelisted for the SPL token integrations. Otherwise, reserve accounting could be incorrect.

8.10 Oracle Considerations

Note **Version 1**

Governance should be aware of the following:

1. The base and quote assets are not validated against the underlying feed and must be validated manually.
2. The base and quote assets could be equal. However, such a price feed is not usable at the time of writing.
3. Configurations of the price feed (`PullFeedAccountData`) are in general not validated onchain during initialization, for instance: `authority`, `min_sample_size`.

4. `process_refresh_oracle` is permissionless, namely it might be leveraged for MEV during atomic swaps.

8.11 Rate Limit Considerations

Note **Version 1**

Below is a list of non-obvious special behavior of rate limits.

1. As outlined in the system overview, setting the maximum outflow to `u64:MAX` is used as "no rate limit". However, any in- and outflow will still apply the delta to the available amount. Hence, a gap between maximum and available amount will still be created. That gap would be kept in case the maximum outflow were to be reduced which might be unexpected in case of "no rate limit".

8.12 SOL Considerations

Note **Version 1**

Governance should be aware of the following considerations regarding the interaction with SOL (`process_claim_rent()`):

1. Only the full balance held by the *Controller Authority* can be withdrawn.
2. There is no direct liquidation path for SOL. However, it could be liquidated by transferring it out to a trusted address.
3. There is no rate limits on transfers and the destination can be arbitrary. However, the amounts available are expected to be small (e.g. refunds from drift). However, no SOL balances (e.g. LayerZero native drop) should be held otherwise.
4. Note that it is not expected that governance or anyone else bridges SOL to ALM controllers (e.g. native drop).
5. Further note that for some integrations (e.g. Kamino) the native balance could be directly used for paying for account creations.