EMURGO

IKIGAI TECHNOLOGIES

# Discovery Project

# Audit Report

Version 1.0

**Release Date: 23 August 2024**

**EMURGO Diligence**

research.emurgo.io/emurgo-diligence

# Contents

# Disclaimer & Scope

## Disclaimer

This report is provided without warranties or guarantees regarding the security or quality of the code under review.

In conducting its audit, EMURGO Diligence prioritized identification of:

- Critical security vulnerabilities based on common attack vectors as they apply to the protocol, which if unresolved might expose the protocol to critical risks such as halting or loss of user funds
- Prominent areas for improvement of code quality

EMURGO Diligence does not claim to have exhaustively identified all potential vulnerabilities and areas of improvement.

## Scope

- The scope of this audit is limited to the on-chain code specified on **pages 9-10** (Files Audited).
- While Ikigai Technologies provided EMURGO Diligence code files pertaining to off-chain transaction construction for informational purposes, these files fall outside the scope of this audit and no conclusions are presented related to this code's security or quality.
- The findings and recommendations contained in this report reflect information gathered by EMURGO Diligence during the course of the Audit Period **(see page 5)**, and exclude any code modifications made beyond that period.
- Initial findings were presented in a preliminary version of this report, and those acknowledged and/or resolved by Ikigai Technologies are noted along with steps taken to resolve them.

EMURGO Diligence reviewed subsequent modifications to the code in question and offered its opinion on the satisfactoriness of Ikigai's resolutions. Code in commits made after the initial state of the codebase **(see page 9)** carry the same disclaimer outlined above.

# Executive Summary

The Discovery protocol aims to provide a fair and transparent mechanism for launching tokens, ensuring that retail participants have a genuine opportunity to acquire tokens based on their commitments. During our audit, EMURGO Diligence identified critical vulnerabilities that posed significant risks to the protocol's integrity. These vulnerabilities have since been addressed by the development team, resulting in enhancements to the protocol’s security and stability.

While additional findings do not present immediate threats to the protocol in its current implementation, all minor and informational findings have been acknowledged by the development team as valid. Although these findings remain unresolved, their acknowledgment demonstrates the team's awareness and careful consideration. Addressing these findings in future iterations could further improve the protocol’s maintainability, comprehensibility, and robustness.

We reiterate our initial recommendations for improvements to the source code and accompanying documentation, particularly in areas such as module structure, code refactoring, validation criteria aggregation, and code readability. Implementing these enhancements would support wider usage, ease of maintenance, and prevent the emergence of future vulnerabilities.

Throughout the audit process, EMURGO Diligence has adhered to industry best practices and exercised due diligence in identifying and evaluating potential vulnerabilities within the protocol. However, it is important to note that this audit report is provided without warranties or guarantees of the quality or security of the code. Our investigation spanned various potential vulnerabilities, including scenarios where the protocol could be exposed to risks that might compromise its integrity or security. This report addresses only the findings we identified, and it is possible that other vulnerabilities may exist that were not detected during our review. Therefore, any decision to deploy the protocol in a production environment should be made with this understanding in mind.

# Methodology

From March 5, 2024 to Aug 23, 2024, Ikigai Technologies contracted EMURGO Diligence to conduct a formal audit of its Discovery protocol, which is developed in collaborative partnership with Anastasia Labs. A comprehensive, multiphase audit process was performed in parallel by two independent teams of three auditors. The audit was performed in the following phases:

1. **Prelude:** Discussions were held between all stakeholders to gather customer requirements, discuss methodology, assess audit scope, and provide estimates regarding timeline of deliverables.
2. **Setup:** EMURGO Diligence prepared a remote development environment with all tools required for code review and internal testing.
3. **Discovery:** EMURGO Diligence conducted a preliminary review of the codebase, including:
    - Diagramming the protocol's off-chain endpoints to present a comprehensive view of protocol functionality.
    - Preparing a table mapping on-chain functions to compiled artifacts and their off-chain references.
    - Initial identification and documentation of common vulnerabilities.
    - Meeting with Anastasia Labs to clarify ambiguities and discuss initial findings.
1. **Confirmation:** Internal testing was conducted to confirm critical vulnerabilities, and supplemental property based testing was performed. Upon conclusion of this phase EMURGO Diligence provided Ikigai Technologies an informal written update to present high-priority findings.
2. **Compilation:** Audit teams met to present and compare findings, which were independently verified by each team. Findings were subsequently reconciled and compiled into a preliminary report.
3. **Resolution:** The preliminary report was delivered to Ikigai Technologies and identified findings were resolved by Ikigai and Anastasia Labs. The final report was prepared documenting resolutions taken.

# Methodology

7. **Review of Fixes:** Ikigai Technologies and Anastasia Labs teams reviewed the "Initial Audit Report" and provided a new repository with fixes for the critical and major findings, while acknowledging the minor and informational findings without resolving them.
8. **Finalization:** EMURGO Diligence reviewed the new repository with the implemented fixes and prepared the "Final Audit Report" as detailed in the following pages. This phase successfully concludes the security audit of the Discovery protocol's on-chain smart contracts.

# Discovery Project Overview

## Background

Significant controversy has surrounded conventional distribution mechanisms for token launches on the Cardano blockchain. The most common method of token distribution involves releasing the token on a decentralized exchange (DEX) through the creation of a liquidity pool: an approach exploitable by technical users with access to front running bots, which monitor DEX contract addresses and execute purchase orders before the token becomes available to retail users. Bots acquire a significant portion of the asset before the general public, then sell it to retail users once the price rises, resulting in substantial profits at retail users' expense.

Another popular approach is to conduct a direct token sale at a fixed price per asset. This process is typically centralized, where the project team receives payments and then sends the owed amount of tokens to buyers. Even if performed in a decentralized manner, this system still fails to address the unfair advantage that front-running bots have over average users.

The community has criticized individuals who use automated tooling to front run retail users and take advantage of them. Unfortunately, the lure of profits often overshadows moral considerations or public perception. Fortunately, Cardano's core protocol can be leveraged to create an ecosystem where the "don't be evil" ideal of commerce is replaced with a "can't be evil" model. With this vision in mind, Ikigai introduces Discovery, a fair and trustless price discovery protocol.

## Purpose

Discovery provides a fair and transparent mechanism for launching tokens, ensuring that retail participants have a genuine opportunity to acquire tokens based on their commitments. This approach prevents early front  running by bots and many other attempts to obtain an unfair advantage over retail users, thus creating a more equitable environment for token distribution.

# Discovery Project Overview

## Implementation

The Discovery protocol is implemented using an on-chain linked list datastructure. To participate in a token's price discovery event, a user must commit Ada by inserting a "node" UTxO into the event's linked list, updating the next pointer of the node onto which the new node UTxO is being inserted. Users can freely withdraw their commitment up to 24 hours before the event's participation deadline, paying only Cardano transaction fees. After this, a cost equal to 25% of the committed amount is imposed to withdraw it, thus discouraging dishonest actors from manipulating perceived demand for a token by committing Ada with the intention of withdrawing it at the last minute. To perform a withdrawal, a user removes their node UTxO from the on-chain linked list, updating the next pointer of the preceding node UTxO.

When the participation deadline is reached, the Discovery Phase begins and no further commitments or withdrawals are permitted. A "Discovery Fold UTxO" is initialized at the discovery fold spending validator and used to fold over all the node UTxOs in the list, collecting the Ada commitment from each node and keeping track of the current position and total commitment collected via its datum. The discovery fold UTxO can be initialized by anyone via the discovery minting policy. The minting policy enforces that the discovery fold UTxO is initialized at the discovery fold spending validator with the correct datum and value. The Discovery Fold spending validator ensures that the fold must process the node UTxOs of the linked list in order, thus guaranteeing that no nodes can be excluded from the fold.

A completed discovery fold can be spent (and the discovery fold token burned) to mint the distribution fold token, and initialize the distribution fold UTxO with the liquidity provider tokens to be distributed to each node in the linked list proportionally to the node's ADA commitment.

# Files Audited

**Github Project URL**: github.com/ikigai-github/discovery

| Reference Date | Description | Commit Hash |
|---|---|---|
| 7-Mar-2024 | Original Repo for Initial Audit Report | 417143ae5739b26743bd431755132b5d87cf11bb |
| 23-July-2024 | Repo with Fixes for Final Audit Report | 80387dbe299e8f5fc4df0e6c04b2302c147b1e80 |

## Final Audit Report Details

The Final Audit Report has been generated on the Github Repo as above, provided on **23 July 2024** with Commit Hash: 80387dbe299e8f5fc4df0e6c04b2302c147b1e80 with the following individual file hashes as below:

| File Name | SHA-256 Hash |
|---|---|
| src/PriceDiscoveryEvent/MultiFoldLiquidity.hs | d824c9d7e8d1a3ae559f1823bf343993e07245baf67624ba73409f9591bb5a0f |
| src/PriceDiscoveryEvent/Utils.hs | b47380e0c54088dd96c28bb7daa3e8d4becfb65e622d58ed005426d06c646647 |
| src/PriceDiscoveryEvent/Mint/Common.hs | e50df3f4196184b0eb9654c3a49399f26f0ddae5e6e941bb4bc23722ffdba0a7 |
| src/PriceDiscoveryEvent/Mint/Helpers.hs | 71c2fcdfaaf622c977e3140924ac148e7a2369aef2e620f88e20952e7e16db90 |
| src/PriceDiscoveryEvent/Mint/Standard.hs | 8927160e8462629d5fff14f318c5beee8cee204c34a36ff6c7f1b75dd4faf679 |
| src/PriceDiscoveryEvent/ProjectTokenHolder.hs | 849621f449ba9ab6486a220d3fc525f07cc940820035a85f270881d68448dcce |
| src/PriceDiscoveryEvent/MultiFold.hs | 6f5e9693c3638fae87eb708b2cf209cf6941378f161b5ca280ff590c64705c28 |
| src/PriceDiscoveryEvent/Validator.hs | cf0869a07f81ab58bf8e3051c0f3a625bfe3ea7d06fa109aaed3595f4121e48c |
| src/Types/LiquiditySet.hs | e81c129ea2bdac84d2586532d9c271c8c3d3dd02f7828cd96650024ea71014f7 |
| src/Types/Classes.hs | f0746964e81440b55963daccb8fe65b347c0f06c536344d1bee70da66d233b15 |
| src/Types/DiscoverySet.hs | b9bf9e38fd748a2414e3806652648fc6c23f4de19eaaf51bd236195fe985859c |
| src/Types/Constants.hs | c24866561953d4497aa3bef14573062df96bbbe21f94028682e4a5f011a8635b |

# Final Audit Report Details

| File Name | SHA-256 Hash |
|---|---|
| src/AlwaysFails.hs | d0eed0ec4638d7e57660938d2944d65b2810aba384a92f365c7312f769b9084f |
| src/Testing/Eval.hs | 2c04cd0a33b56c3e396056806e93b81fead19877ea56a259dcb3143ba79e0ae1 |
| src/Testing/Test.hs | 4b5e2ffd1a39acdbedb6b2c31d67d988ef5e2a588da2418db1688bb58dd9122a |
| src/LiquidityEvent/Mint/Common.hs | 5629968a7c34c7382516e9feb75fdcca3439cc4b960976bbf3b21177ab2c58fa |
| src/LiquidityEvent/Mint/Standard.hs | 7b16e6dc99d51833410a826477a06d46f48f9e83fcc188a84b3f2c061dd43cdb |
| src/LiquidityEvent/LiquidityTokenHolder.hs | 6b14730f2af163539bc5e6be1054a0df8fc19e8eea37952498128284e5b821c3 |
| src/LiquidityEvent/Validator.hs | f5f5e28f17d565bc5a53ee8ddf661a4126eb3d97e8de235317752abe3658934b |
| src/LiquidityEvent/ProxyTokenHolderV1.hs | 831b32185c1905e332042a571e4595167ccd334923f8e4301197e34268c3b8ee |

# Findings - Severity Classification System

**EMURGO Diligence employed a four category system to identify and document findings**

| | |
|---|---|
| **CRITICAL** | Confirmed vulnerabilities presenting existential risks to the protocol, including halting or loss of user funds. |
| **MAJOR** | Findings that may violate specifications in a non-critical way, causing unexpected behavior or subjecting the protocol to risk if neglected. |
| **MINOR** | Findings not posing significant risk to protocol functionality, but impacting performance or maintainability. |
| **INFORMATIONAL** | Recommendations for improving the semantic and cosmetic quality of code. |

# Findings - Resolution Status Classification

| STATUS | DESCRIPTION |
|---|---|
| **RESOLVED** | Dev. Team has resolved the reported issue. |
| **ACKNOWLEDGED** | Findings that have been acknowledged by the project team. Projects can decide to not fix findings for whatever reason. |
| **PENDING** | Dev. Team has neither acknowledged nor resolved the issue. |

## Summary of Findings

| Severity | Number of findings |
|---|---|
| CRITICAL | 2 |
| MAJOR | 1 |
| MINOR | 6 |
| INFORMATIONAL | 7* |

* Including 1 new issue created as a result of the final fixes by Dev. Team on this repo

## Overview of Findings and Resolution Status

| S. No | ID | Severity | Title | Status |
|---|---|---|---|---|
| 1. | IKG-401 | Critical | Absent datum verification in reward fold | RESOLVED |
| 2. | IKG-402 | Critical | Minimum UTxO Violation | RESOLVED |
| 3. | IKG-301 | Major | Erroneous restriction on commitment modification | RESOLVED |
| 4. | IKG-201 | Minor | Inefficient pand'List function | ACKNOWLEDGED |
| 5. | IKG-202 | Minor | Module organization & code duplication | ACKNOWLEDGED |
| 6. | IKG-203 | Minor | Diffuse PTHolder token validation | ACKNOWLEDGED |
| 7. | IKG-204 | Minor | Weak project token validation | ACKNOWLEDGED |
| 8. | IKG-205 | Minor | Potential for arbitrarily large datums | ACKNOWLEDGED |
| 9. | IKG-206 | Minor | Potential unrestricted assets in commitment UTxOs | ACKNOWLEDGED |
| 10. | IKG-101 | Informational | Insufficient documentation & specification | ACKNOWLEDGED |
| 11. | IKG-102 | Informational | Ambiguous naming | ACKNOWLEDGED |
| 12. | IKG-103 | Informational | Rigid reference input handling | ACKNOWLEDGED |
| 13. | IKG-104 | Informational | Dead code | ACKNOWLEDGED |
| 14. | IKG-105 | Informational | Line length | ACKNOWLEDGED |
| 15. | IKG-106 | Informational | Use of ViewPatterns | ACKNOWLEDGED |
| 16. | IKG-301A | Informational | Undocumented deadline restriction on commitment modification | ACKNOWLEDGED |

# Details of Findings

## 1. IKG-401: Absent datum verification in reward fold

**Findings:** CRITICAL **Status:** RESOLVED

### Description

The prewardFoldNode functions (src/PriceDiscoveryEvent/MultiFold.hs:651, src/PriceDiscoveryEvent/MultiFoldLiquidity.hs:701) contain various checks to ensure that the due amount of tokens are taken from the rewardFoldValidator and distributed to a participant who committed ADA to the nodeValidator.

The due amount of tokens is sent back to the nodeValidator and must also be accompanied by a correct datum to ensure that the participant can withdraw the tokens later. However, only the value is checked and not the datum. Datum verification is also absent in the nodeValidator script, which delegates all verification to the rewardFold script. This issue poses a critical risk of loss of user funds.

Using the supplied off-chain code, we verified that one can use payToAddress with the nodeValidator address instead of payToContract to omit the datum and all provided tests still pass, despite the fact that no user will be able to withdraw funds from the nodeValidator contracts (since outputs without datums at contract addresses are unspendable).

### Recommendation:

We recommend introducing additional checks to ensure that a properly constructed datum is included on the output.

### Resolution:

- The function `prewardFoldNode` function has been commented out and replaced with `prewardFoldNodes`. This function uses the `pfoldCorrespondingUTxOs` function which again uses the `prewardSuccessor` function.
- `prewardSuccessor` addresses this issue by restricting datum modification of node inputs.

## 2. IKG-402: Minimum UTxO violation

**Findings:** CRITICAL **Status: RESOLVED**

### Description

The protocol allows users to withdraw ADA from their commitment in the discovery set of up to 10 ADA (PriceDiscoveryEvent/Validator.hs, l. 82), which implies that a user can create a participating node in the discovery set that holds a total of less than 3 ADA, since a minimum quantity is not enforced.

If this node is included in a discovery fold, it will trigger a complete halt of the distribution phase when trying to process the node, because during distribution of funds (prewardFoldNode and prewardFoldNodes, Multifold.hs:l526, l711) it is enforced with equality that the output value to the node validator must be initialValue nodeCommitment foldingFee.

In the proposed scenario, this value is almost certainly not enough to satisfy minUTxO requirements for an output that contains the same datum but includes additional tokens. This vulnerability has been verified by creating such a low-ADA node and attempting to fold the rewards over the resulting set, resulting in failure.

### Recommendation:

- We recommend allowing more than the minimum ADA to be sent to the node validator to handle minUTxO violations at a negligible cost of a few extra ADA per violation.
- Commitment modification should also ensure that a minimum amount of ADA is retained at the Discovery node.

### Resolution:

- This issue has been addressed by requiring the commitment to be raised by at least 11 Ada and prohibiting any reduction in the  commitment.
- This will maintain Minimum ADA but the problem is that it will restrict the user from withdrawing ADA from their commitment.

### In the following line of code:

```
passert “Cannot reduce ada value” (plovelaceValueOf # ownInputF.value #<
plovelaceValueOf # ownOutputF.value - 10_000_000)
```

Suppose `ownInputF.value` is 10_000_000 lovelace, this means that the `wnOutputF.value` must be at least 21_000_000 lovelace to satisfy this check.

# 3. IKG-301: Erroneous deadline restriction on commitment modification

**Findings:** MAJOR **Status: RESOLVED**

## Description

The protocol's two spending validator functions each contain a check that will cause modifying commitment actions to fail if the transaction's validity interval falls after 86,400 PPOSIXtime units prior to the deadline (src/PriceDiscoveryEvent/Validator.hs:93 and src/LiquidityEvent/Validator.hs:100). Since PPOSIXtime values are measured in milliseconds, not seconds, this value should likely be 86,400,000 (i.e. 24 hours). Furthermore, the protocol's specification does not include any mention of restricting modification of commitments within a specified window of time prior to the event deadline; thus, the purpose of this check is ambiguous and it is unclear whether it accords with the protocol's intended functionality.

## Recommendation:

We recommend clarifying the intended purpose of this check in the protocol's specification, and correcting the two values to 86,400,000 milliseconds.

## Resolution:

**RESOLVED**

`86_400` *has been changed to* `86_400_000` *to represent 24 hours*

# 4. IKG-201: Inefficient pand'List function

**Findings:** MINOR **Status: ACKNOWLEDGED**

## Description

A widely used helper function, pand'List (src/PriceDiscoveryEvent/Utils.hs:680), accepts a list of boolean elements and folds over them with a 'pand' operation. This approach mandates the calculation of the entire list, irrespective of individual true or false outcomes. Consequently, if a condition evaluates to false early in the list, the function will continue processing all remaining elements, resulting in unnecessary computation.

Additionally, the implementation of pand'List relies on the space leaking Haskell function foldl1, which (via underlying use of foldl) accumulates unevaluated "thunks" in memory with no benefit in performance.

## Recommendation:

We recommend replacing validation logic relying on pand'List with individual 'pand' operations for each condition. By adopting this approach, the functions will immediately return false upon encountering the first false condition and eliminate unnecessary computation. Alternatively, pand'List might be reimplemented to promptly output false when it encounters a false condition. If the function is retained, it should be reimplemented to use foldr1 or foldl'1 instead of foldl1 to prevent unnecessary memory consumption.

## Resolution:

**ACKNOWLEDGED**

# 5. IKG-202: Module organization & code duplication

**Findings:** MINOR **Status: ACKNOWLEDGED**

## Description

The protocol's source code is organized into two divisions: LiquidityEvent and DiscoveryEvent.

However there are several modules related to the LiquidityEvent division that are located in the DiscoveryEvent directory. Moreover, there is a large volume of duplicated code between the LiquidityEvent and DiscoveryEvent divisions, as well as utility code that, while shared by both divisions, is currently located in the DiscoveryEvent directory.

Within certain modules, multiple script generator functions are present. Conventionally each script generator function is assigned its own module. The coexistence of multiple script generator functions within a single module blurs the lines of separation, potentially leading to confusion and reduced code maintainability.

## Recommendation:

We recommend moving all LiquidityEvent modules to the LiquidityEvent directory. Shared logic between the two divisions should be factored out into common utilities and placed, and placed along with existing shared utilities in a top-level module (located at the same directory level as the LiquidityEvent and DiscoveryEvent subdirectories). Script generator functions should be separated into unique modules. Implementing these recommendations will improve code maintenance and mitigate the possibility for accidental introduction of bugs and vulnerabilities intoproduction.

## Resolution:

**ACKNOWLEDGED**

# 6. IKG-203: Diffuse PTHolder token validation

**Findings:** MINOR **Status: ACKNOWLEDGED**

## Description

The pbeginRewards validator function (src/LiquidityEvent/ LiquidityTokenHolder.hs:157) applies the function pfindCurrencySymbolsByTokenName, which filters the validator's input value and returns currency symbols matching the provided token name (PTHolder). The result is further validated to ensure only one such asset is included in the input, and that one unit of it is burned in the transaction. As currently implemented, this validation will succeed if any token named PTHolder is burned (i.e. an asset with a currency symbol unrelated to the protocol), in theory allowing a reward token to be minted. Although this potential vulnerability is precluded by a check in the reward token minting policy, which verifies the currency symbol of the PTHolder token, this diffusion of the token's validation logic reduces the readability and maintainability of the code.

## Recommendation:

Rather than filtering potential currency symbols by token name and validating the currency symbol in the associated minting policy, we recommend integrating this logic within the validator, passing the PTHolder token's currency symbol to the function and using this symbol to identify the correct token. Consolidating this check within the validation logic would improve the overall clarity and maintainability of the codebase by making validation rules more easily understandable.

## Resolution:

**ACKNOWLEDGED**

## 7. IKG-204: Weak validation by token prefix

**Findings:** MINOR **Status: ACKNOWLEDGED**

### Description

In the pLiquiditySetValidator function (src/LiquidityEvent/Validator.hs:80), the minted value from the transaction context is checked to determine whether a token with a token name containing a specified prefix is minted or burned. While the node token minting policy currently ensures that any linked-list interaction mints or burns such a token, it's important to note that within the spending validator, any token whose name contains the specified prefix can be minted or burned to pass the validation, regardless of its currency symbol. While it's unlikely that an attacker would intentionally mint or burn such a counterfeit node token to lock their funds in the contract, this presents a slight weakness in the validation logic.

### Recommendation:

We recommend reviewing and strengthening this check to ensure that only valid project tokens are involved in linked list interactions, thereby enhancing the security and robustness of the protocol. This can be achieved by passing the currency symbol to the validator and checking for the presence of a token matching both the valid currency symbol and the prefixed token name.

### Resolution:

**ACKNOWLEDGED**

# 8. IKG-205: Potential for arbitrarily large datums

**Findings:** MINOR **Status: ACKNOWLEDGED**

## Description

A potential vulnerability related to finding IKG-402 (Minimum UTxO violation) could emerge if a user is able to insert UTxOs with arbitrarily large datums into the discovery set (including very large bytestrings as ‘key’ or ‘next’ values, which are present in the datums used by both validators, or a very large ‘commitment’ value in the LiquidityEvent validator). Although this vulnerability is safeguarded against in the PriceDiscoveryEvent validator (by ensuring that key values correspond to PubKeyHash values from the signatures list and thus restricting their length), the use of an unbounded integer value for the unused ‘commitment’ field of the PLiquiditySetNode type used as a datum in the LiquidityEventValidator subjects it to potential risk of excessively large datums being attached to node set UTxOs, ultimately rendering them unspendable and causing the protocol to halt.

## Recommendation:

The unused ‘commitment’ integer field should be removed from the PLiquid-itySetNode type definition. Caution should be exercised when modifying the protocol in the future to ensure that sufficient validation to limit datum size is retained.

## Resolution:

**ACKNOWLEDGED**

# 9. IKG-206: Potential unrestricted assets in commitment UTxOs

**Findings:** MINOR **Status: ACKNOWLEDGED**

## Description

The protocol's two spending validator functions (pDiscoverySetValidator in src/PriceDiscoveryEvent/Validator.hs and pLiquiditySetValidator in src/LiquidityEvent/Validator.hs) do not contain validation to limit assets permitted in commitment UTxOs produced by linked list actions. While this vulnerability is currently safeguarded against by separate validation logic in the associated minting policies (src/PriceDiscoveryEvent/Mint/Common.hs:124, src/LiquidityEvent/Mint/Common.hs:116), which restricts the number of unique assets to 2, such diffusion of validation logic makes protocol behavior more difficult to understand and increases the risk of opening a vulnerability in the future. Permitting unrestricted assets in commitment UTxOs could potentially render the protocol vulnerable to a token dust attack: if too many tokens are included in a node commitment UTxO inserted into the linked list, the UTxO would be unspendable, thus breaking the fold functionality and making the protocol unusable.

## Recommendation:

While the protocol currently has sufficient safeguards in place, we recommend consolidating validation logic and making the asset restriction explicit in the spending validators to improve readability and remove any potential for future vulnerabilities.

## Resolution:

**ACKNOWLEDGED**

# 10. IKG-101: Insufficient documentation & specification

**Findings:** INFORMATIONAL **Status: ACKNOWLEDGED**

## Description

The absence of comprehensive documentation limits effective evaluation of the codebase in alignment with its intended design, a difficulty further compounded by the intricate structure of the code and the high complexity of the Plutarch language. The absence of documentation explaining how to correctly instantiate the contracts with all necessary parameters could lead to incorrect deployment. The off-chain code provided to our team, which served as de facto documentation for instantiation of the contracts, is incomplete and lacks code for instantiation for the Liquidity variant of the protocol.

The protocol specification provided is also limited in scope and detail.

## Recommendation:

We recommend developing comprehensive documentation that is both up to date and clearly reflects the logic implemented by each major script generator function and its associated data types. Documentation should also discuss architecture and design considerations and provide usage examples with transactions and instructions for deployment.

The protocol specification should be expanded to discuss each component in greater detail and include diagrams demonstrating interconnections and dependencies among components.

## Resolution:

**ACKNOWLEDGED**

## 11. IKG-102: Ambiguous naming

**Findings:** INFORMATIONAL **Status: ACKNOWLEDGED**

### Description

Numerous types and functions throughout the codebase have names that hinder code readability:

- Distinct items have highly similar names which are easily conflated
- Names include abbreviations which have unclear semantics

Additionally, the chosen token names (in src/Types/Constants.hs) lack descriptiveness and provide little context regarding their purposes.

Given the high complexity of the protocol and its multitude of overlapping functionalities, it is critical that clear lines of demarcation be drawn between various components and for each of their roles to be intuitable based on their assigned names.

### Recommendation:

We recommend renaming items in the code to increase the intuitive categorization of code based on purpose, and to use more verbose names without abbreviations that clarify the role of each component.

### Resolution:

**ACKNOWLEDGED**

## 12. IKG-103: Rigid reference input handling

**Findings:** INFORMATIONAL **Status: ACKNOWLEDGED**

### Description

Certain segments of the code impose a strict requirement for a singular reference input, posing limitations that could result in unexpected transaction behavior. For instance, this constraint can be incompatible with transaction builder packages (like MeshJS) known to include transaction inputs' TxOutRefs in reference inputs list in addition to the specified reference TxOutRefs. Such actions would result in failed validations due to the strict adherence to a single reference input protocol.

### Recommendation:

To address the rigidity imposed by the singular reference input requirement, we propose a revised validation mechanism capable of dynamically filtering and identifying the correct reference input based on its datum structure and values. This would accommodate diverse reference inputs, allowing users more flexibility and expanding the protocol's compatibility with various transaction builders, while ensuring validation success provided the correct reference is present.

### Resolution:

**ACKNOWLEDGED**

## 13. IKG-104: Dead code

**Findings:** INFORMATIONAL **Status: ACKNOWLEDGED**

### Description

The codebase contains many utility functions that are not used anywhere in the protocol's functionality. This surplus of functions introduces unnecessary complexity and potential confusion to the codebase, hindering code readability and complicating future maintenance efforts.

### Recommendation:

We recommend a comprehensive review of the codebase to identify and remove any unused utility functions.

### Resolution:

**ACKNOWLEDGED**

## 14. IKG-105: Line length

**Findings:** INFORMATIONAL **Status: ACKNOWLEDGED**

### Description

Code lines throughout the codebase frequently exceed 100 characters, and in many instances are in excess of 130. Given the complexity of the protocol logic and intrinsically low readability of Plutarch code, overly long lines of code compound existing findings of readability.

**Recommendation:**

We recommend maintaining a maximum of 100 characters per line and breaking up complex lines of code across multiple lines as needed to improve the codebase's readability.

**Resolution:**

**ACKNOWLEDGED**

# 15. IKG-106: Use of ViewPatterns

**Findings:** INFORMATIONAL **Status: ACKNOWLEDGED**

## Description

The codebase makes frequent use of the ViewPatterns extension, which is not idiomatic in Haskell code and degrades code readability by requiring the reader to mentally parse program logic across both the assignment and value sides of expressions.

## Recommendation:

We recommend the use of idiomatic pattern-matching in lieu of ViewPatterns to separate logic and assignment and support a more comprehensible flow.

## Resolution:

**ACKNOWLEDGED**

# 16. IKG-301A: Undocumented deadline restriction on commitment modification

**Findings:** INFORMATIONAL **Status: ACKNOWLEDGED**

## Description

A remaining ambiguity after the resolution of IKG-301 (Erroneous deadline restriction on commitment modification)

The protocol's two spending validator functions each contain a check that will cause modifying commitment actions to fail if the transaction's validity interval falls after 86,400,000 POSIXtime units prior to the deadline (src/PriceDiscoveryEvent/Validator.hs:93 and src/ LiquidityEvent/ Validator.hs:100).

The protocol's specification does not include any mention of restricting modification of commitments within a specified window of time prior to the event deadline; thus, the purpose of this check is ambiguous and it is unclear whether it accords with the protocol's intended functionality.

## Recommendation:

Remove the check or document the intended behavior in the specification.

## Resolution:

**ACKNOWLEDGED**

# About Us

EMURGO

EMURGO Diligence is the specialized audit service division of EMURGO, one of the co-founding entities of the Cardano blockchain protocol. With a deep commitment to fostering trust and security within the blockchain ecosystem, EMURGO Diligence provides comprehensive auditing services tailored to smart contracts, decentralized applications (DApps), and blockchain protocols.

Leveraging our extensive expertise in blockchain technology, particularly within the Cardano ecosystem, we offer rigorous and thorough audits designed to identify vulnerabilities, enhance security, and ensure the robustness of blockchain projects. Our audit process is grounded in industry best practices and informed by our unique position as a foundational partner in the development of Cardano.

At EMURGO Diligence, our mission is to empower blockchain developers and businesses by providing them with the insights and guidance necessary to build secure, reliable, and efficient blockchain solutions. We are dedicated to upholding the highest standards of security and integrity, helping to drive the adoption of blockchain technology by ensuring that projects are ready to meet the demands of the market.

# Appendix A: Note on Test Coverage & Documentation

Ikigai Technologies provided EMURGO Diligence with a private code package containing off-chain code and associated tests to assist with the audit process. As previously mentioned, this code is outside the scope of the audit, and accordingly served strictly to support the audit discovery phase.

Despite this, our team has identified shortcomings in the provided test suite and have included this supplementary note to bring these to the attention of Ikigai Technologies, as well as offer recommendations for improvement.

While off-chain endpoints for transaction construction are present for both variants of the protocol (Price Discovery & Liquidity), the package only contains tests for the Price Discovery endpoints. Despite large overlap in their source code and core functionality, the two variants function as distinct protocols with unique behavior, and thus each requires equally rigorous testing. The fact that multiple findings in this report pertain to the Liquidity variant supports the urgency of rectifying this incomplete test coverage.

Our team also noted the absence of documentation to accompany the existing tests, which hindered our efforts to replicate results in a timely manner. Initial obstacles and failing tests resulted due to lack of clarity on dependency version and the correct procedure for running tests. Additionally, we encountered difficulties identifying correspondences between compiled script artifacts and their sites of usage in off-chain endpoints and tests due to inconsistent and ambiguous script identifiers, which required additional research and production of internal documentation. We recommend including such documentation in the off-chain repository and adopting clear and consistent identifiers for scripts across on-chain and off-chain code.