CERTIK

Aergo (HPP)

Security Assessment

CertiK Assessed on Nov 4th, 2025

CertiK Assessed on Nov 4th, 2025

Aergo (HPP)

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Exchange	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 11/03/2025 Final report published on 11/04/2025

Vulnerability Summary

5	0	0	5	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

1	Centralization	1 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
1	Minor	1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
3	Informational	3 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | AERGO (HPP)

AUDIT SCOPE | AERGO (HPP)

hpp-io/contracts
HPP_Migration_AERGO.sol
HPP_Migration_AQT.sol

APPROACH & METHODS | AERGO (HPP)

This audit was conducted for Aergo (HPP) to evaluate the security and correctness of the smart contracts associated with the Aergo (HPP) project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Static Analysis, Manual Review, and Formal Verification.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

REVIEW NOTES | AERGO (HPP)

Overview

The **Aergo(HPP)** is a migration project with two contract. The files currently under review include:

- HPP_Migration_AQT.sol
- HPP_Migration_AERGO.sol

The `AQTtoHPPMigration` contract in `HPP_Migration_AQT.sol` facilitates migration of `AQT` tokens to `HPP` tokens at a fixed exchange rate of `1 AQT = 7.43026 HPP`. The `AergoHPPSwap` contract in `HPP_Migration_AERGO.sol` enables `1:1` token swaps between `AERGO` and `HPP` tokens.

External Dependencies

In **Aergo(HPP)**, the module inherits or uses a few of the depending injection contracts or addresses to fulfill the need of its business logic. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

The following library/contract are considered as the third-party dependencies:

- @openzeppelin/contracts/@v5.3.0

Privileged Functions

In the **Aergo(HPP)** project, the privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the `Centralization` finding.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

FINDINGS | AERGO (HPP)

5	0	1	0	0	1	3
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for Aergo (HPP) to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 5 issues were identified. Leveraging a combination of Static Analysis, Manual Review & Formal Verification the following findings were uncovered:

ID	Title	Category	Severity	Status
AEH-01	**Centralization Risks In Source**	**Centralization**	**Centralization**	**Acknowledged**
AEH-02	Deflationary Token Acceptance Leads To Incorrect Migration Payouts And Contract Loss	Volatile Code	Minor	Acknowledged
AEH-03	Missing Token Address Validation Leads To Liquidity Removal Failures And Elevated Risk	Code Optimization	Informational	Acknowledged
AEH-04	Consider Using `Ownable2Step`	Volatile Code	Informational	Acknowledged
AEH-05	Lack Of Check Decimal	Logical Issue	Informational	Acknowledged

AEH-01 | Centralization Risks In Source

Category	Severity	Location	Status
Centralization	Centralization	HPP_Migration_AERGO.sol: 98, 112, 127, 134; HPP_Migration_AQT.sol: 128, 143, 157, 164	Acknowledged

Description

In the contract `AergoHPPSwap` and `AQTtoHPPMigration`, the role `owner` has authority over several critical functions in the protocol. Any compromise to the `owner` account may allow an attacker to misuse these capabilities.

The function `removeLiquidity(address tokenAddress, uint256 amount)` grants the owner the ability to withdraw any amount of either token (AERGO or HPP) from the contract. Because swap functionality depends on the contract maintaining sufficient liquidity, the owner effectively has the power to drain liquidity and render swaps unusable, which could disrupt user expectations and damage trust.

The `burnAergo(uint256 amount)` and `burnAQT(uint256 amount)` functions allow the owner to permanently destroy tokens held inside the contract by sending them to a burn address. While this may be intended for managing supply, if misused, it can result in irreversible loss of assets that users expect to remain available for swapping.

The functions `pause()` and `unpause()` give the owner the ability to stop or resume all swapping operations. This introduces a centralized control point where the owner can halt user activity at any time, which may lead to a denial-of-service scenario if paused unexpectedly or maliciously.

Overall, the contract relies on trust that the owner will exercise these powers responsibly.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;

AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Aergo (HPP), 11/04/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> The Owner will be configured as a Safe-based Multisig Wallet (2/3), with each signer's wallet also configured as a hardware wallet. Three hardware wallets, a 2-of-3 multisig wallet. Therefore, the centralization issue is expected to be secure in the long term.y]

AEH-02 Deflationary Token Acceptance Leads To Incorrect Migration Payouts And Contract Loss

Category	Severity	Location	Status
Volatile Code	Minor	HPP_Migration_AQT.sol: 96~121	Acknowledged

Description

The `migrateAQTtoHPP` function is intended to accept `AQT` tokens from a user and pay out `HPP` tokens at a fixed rate `MIGRATION_RATE`. It requires `aqtAmount` to be whole tokens (`aqtAmount % 10**18 == 0`), computes `hppAmount = (aqtAmount * MIGRATION_RATE) / 10**18`, checks `HPP.balanceOf(address(this)) >= hppAmount`, calls `AQT.safeTransferFrom(msg.sender, address(this), aqtAmount)`, and then transfers `HPP.safeTransfer(msg.sender, hppAmount)`.

```
96
    function migrateAQTtoHPP(uint256 aqtAmount) external nonReentrant whenNotPaused
{
97          if (aqtAmount == 0) revert ZeroAmount();
98
99          // Check if amount has decimals (must be whole tokens only)
100          if (aqtAmount % (10 ** 18) != 0) {
101              revert DecimalsNotAllowed();
102          }
103
104          // Calculate HPP amount to receive
105          // aqtAmount is in wei, so we divide by 10^18 to get token count,
106          // then multiply by MIGRATION_RATE
107          uint256 hppAmount = (aqtAmount * MIGRATION_RATE) / (10 ** 18);
108
109          // Check HPP liquidity
110          if (HPP.balanceOf(address(this)) < hppAmount) {
111              revert InsufficientHPPLiquidity();
112          }
113
114          // Transfer AQT from user to contract
115          AQT.safeTransferFrom(msg.sender, address(this), aqtAmount);
116
117          // Transfer HPP to user
118          HPP.safeTransfer(msg.sender, hppAmount);
119
120          emit TokensMigrated(msg.sender, aqtAmount, hppAmount);
121      }
```

The implementation assumes the contract will receive exactly `aqtAmount` after `safeTransferFrom`. It does **not** verify the actual `AQT` balance change (before/after) and therefore does not detect `fee-on-transfer` / deflationary token behaviors

where the token contract burns or takes a fee during transfer. As a result the internal calculation of `hppAmount` is based on the user-supplied `aqtAmount`, not on the `actualReceived = afterBalance - beforeBalance`.

The impact is that when `AQT` is a deflationary or fee-on-transfer token, the contract may receive **less** than `aqtAmount` but still pay full `hppAmount` to the user. This causes a direct economic loss to the contract (shortfall equals expected minus actual received AQT converted at the migration rate). Combined with the fact that liquidity check for `HPP` happens **before** receiving `AQT`, the contract can wrongly transfer `HPP` while being underfunded in `AQT`. In worst cases repeated exploitation can drain protocol-held assets or break migration invariants.

Scenario

1. Alice calls `migrateAQTtoHPP` with `aqtAmount = 1 * 10**18` (1 whole AQT). The contract computes `hppAmount` and verifies `HPP.balanceOf(address(this))` is sufficient.
2. `AQT` is a fee-on-transfer token that charges a 1% transfer fee. When `AQT.safeTransferFrom(Alice, address(this), 1 AQT)` executes, the contract receives only `0.99 AQT` (i.e. `actualReceived = 0.99 * 10**18`).
3. The contract, unaware of the fee, transfers `hppAmount` based on `1 AQT` to Alice. The contract has therefore given more HPP value than the net AQT it actually received, incurring a net loss.
4. An attacker or an uninformed user repeatedly migrates fee-on-transfer `AQT` tokens, causing the contract to lose assets proportional to the transfer fees over many operations.

Recommendation

1. Ensure `AQT` Is Non-Deflationary: enforce (by documentation, frontend checks, and on-chain configuration) that only non-deflationary tokens — tokens that do not burn or take transfer fees — are accepted for migration. Maintain a governance-controlled whitelist of supported `AQT` token addresses and reject unknown or unverified tokens.

2. Verify Actual Received Balance If It Is Deflationary: if you must accept tokens that may be deflationary, compute the actual received amount using a before/after balance check and base `hppAmount` on the `received` value (or revert if `received != aqtAmount` to disallow deflationary transfers).

Alleviation

[Aergo (HPP), 11/04/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> AQT is not fee-on-transfer token. So it seems there should be no problem.

AEH-03 Missing Token Address Validation Leads To Liquidity Removal Failures And Elevated Risk

Category	Severity	Location	Status
Code Optimization	Informational	HPP_Migration_AERGO.sol: 98~104; HPP_Migration_AQT.sol: 128~135	Acknowledged

Description

The `removeLiquidity` function is intended to allow the contract `owner` (or multisig) to withdraw a specified `amount` of an ERC-20 token identified by `tokenAddress` from the contract and transfer it to the caller. The function checks `amount != 0`, constructs an `IERC20 token = IERC20(tokenAddress)`, then calls `token.safeTransfer(msg.sender, amount)` and emits `LiquidityRemoved`.

The implementation lacks validation of `tokenAddress` (for example `tokenAddress != address(0)`), does not verify that `tokenAddress` is a contract, nor does it restrict which tokens can be removed. Because `safeTransfer` performs an external call into the `token` contract, passing an invalid or malicious `tokenAddress` can cause immediate reverts or arbitrary external code execution. An accidental `tokenAddress = address(0)` will cause the transfer call to revert (no code at `address(0)`), preventing the owner from removing liquidity; passing a malicious token contract can execute attacker-controlled logic during `safeTransfer`.

Recommendation

Validate inputs, constrain callable token set, and harden the function against accidental and malicious usage. Practical fixes are:

```
if (tokenAddress == address(0)) revert InvalidTokenAddress();
```

Alleviation

[Aergo (HPP), 11/04/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

AEH-04 | Consider Using `Ownable2Step`

Category	Severity	Location	Status
Volatile Code	Informational	HPP_Migration_AERGO.sol: 17; HPP_Migration_AQT.sol: 17	Acknowledged

Description

The `AergoHPPSwap` contract and `AQTtoHPPMigration` contract are using OpenZeppelin's single-step `Ownable`, where ownership transfers immediately upon `transferOwnership`, risking accidental or unwanted transfers:

```
contract AQTtoHPPMigration is ReentrancyGuard, Ownable, Pausable {
```

```
contract AergoHPPSwap is ReentrancyGuard, Ownable, Pausable {
```

The `Ownable2Step` prevents the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner's permissions actively accept via a contract call of its own.

Consider using `Ownable2Step` from OpenZeppelin Contracts to enhance the security of your contract ownership management. This contract prevents the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call.

Recommendation

Recommend using `Ownable2Step`.

Alleviation

[Aergo (HPP), 11/04/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

AEH-05 | Lack Of Check Decimal

Category	Severity	Location	Status
Logical Issue	Informational	HPP_Migration_AERGO.sol: 66~77; HPP_Migration_AQT.sol: 79~90	Acknowledged

Description

Both `AergoHPPSwap` and `AQTtoHPPMigration` contracts lack validation for token decimal consistency in their constructors. This oversight can lead to incorrect exchange rates, precision loss, or unintended fund transfers when swapping tokens with differing decimal places (e.g., 18 vs. 6 decimals).

In `AergoHPPSwap`, If `AERGO.decimals ≠ HPP.decimals`, users may receive incorrect amounts (e.g., losing precision or overpaying):

```
 83  function swapAergoForHPP(uint256 amount) external nonReentrant whenNotPaused {
 84          if (amount == 0) revert ZeroAmount();
 85          require(HPP.balanceOf(address(this)) >= amount,
"Insufficient HPP liquidity");
 86
 87          AERGO.safeTransferFrom(msg.sender, address(this), amount);
 88          HPP.safeTransfer(msg.sender, amount);
 89
 90          emit Swap(msg.sender, address(AERGO), address(HPP), amount);
 91      }
```

Example: Swapping `1 AERGO` (`18` decimals) for `1 HPP` (`6` decimals) could result in a `1e12` HPP loss per token.

The same issue exists in `AQTtoHPPMigration`.

Recommendation

Add a decimal check in the constructor:

```
constructor(address _aergoToken, address _hppToken, address _initialOwner)
Ownable(_initialOwner) {
    // Existing checks
    if (_aergoToken == address(0)) revert ZeroAddress();
    if (_hppToken == address(0)) revert ZeroAddress();
    if (_aergoToken == _hppToken) revert SameTokenAddress();

    // New: Ensure decimals match
    if (IERC20Metadata(_aergoToken).decimals() !=
IERC20Metadata(_hppToken).decimals()) {
        revert DecimalsMismatch();
    }

    AERGO = IERC20(_aergoToken);
    HPP = IERC20(_hppToken);
}
```

```
constructor(
        address _aqtToken,
        address _hppToken,
        address _initialOwner
    ) Ownable(_initialOwner) {
        if (_aqtToken == address(0)) revert ZeroAddress();
        if (_hppToken == address(0)) revert ZeroAddress();
        if (_aqtToken == _hppToken) revert SameTokenAddress();
        // New: Ensure decimals match
        if (IERC20Metadata(_aqtToken).decimals() !=
IERC20Metadata(_hppToken).decimals()) {
            revert DecimalsMismatch();
        }
        AQT = IERC20(_aqtToken);
        HPP = IERC20(_hppToken);
    }
```

Alleviation

[Aergo (HPP), 11/04/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

FORMAL VERIFICATION | AERGO (HPP)

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of Standard Ownable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,
- function `transferOwnership` that transfers the ownership to a new owner.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
ownable-renounceownership-correct	Ownership is Removed
ownable-transferownership-correct	Ownership is Transferred
ownable-owner-succeed-normal	`owner` Always Succeeds
ownable-renounce-ownership-is-permanent	Once Renounced, Ownership Cannot be Regained

Verification Results

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract AQTtoHPPMigration (contracts/HPP_Migration_AQT.sol) In Commit 8c4615deae36f1c39224e4eeb79e4765de9d1c23

Verification of Standard Ownable Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	Inconclusive	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results For Contract AergoHPPSwap (contracts/HPP_Migration_AERGO.sol) In Commit 8c4615deae36f1c39224e4eeb79e4765de9d1c23

Verification of Standard Ownable Properties

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	Inapplicable	The property does not apply to the contract

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

APPENDIX | AERGO (HPP)

Finding Categories

Categories	Description
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held

when it was invoked.

- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed Ownable Properties

Properties related to function `renounceOwnership`

ownable-renounce-ownership-is-permanent

The contract must prohibit regaining of ownership once it has been renounced.

Specification:

```
constraint \old(owner()) == address(0) ==> owner() == address(0);
```

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to address(0).

Specification:

```
ensures this.owner() == address(0);
```

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

Properties related to function `owner`

ownable-owner-succeed-normal

Function `owner` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.