# Security Assessment & Formal Verification Report

# GhoStewardV2

03/24

*Prepared for*
**Aave**

# Table of content

# Project Summary

## Project Scope

| Repo Name | Repository | Commits | Compiler version | Platform |
|---|---|---|---|---|
| gho-core | Github Repository | 77dd627 | 0.8.10 | Ethereum |

## Project Overview

This document describes the specification and verification of the **GhoStewardV2** using the Certora Prover and manual code review findings. The work was undertaken from **3 March 2024** to **14 March 2024**.

The following contract is included in our scope:

- `GhoStewardV2`
- `IGhoStewardV2`

The Certora Prover demonstrated that the implementation of the Solidity contract above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of the Solidity contract. During the verification process and the manual audit, the Certora team hasn't discovered any bugs.

## Protocol Overview

The GhoStewardV2 allows approved risk providers to change GHO and GSM parameters. It checks that the new parameters are in accordance with predefined limitations, and that changes intervals are no less than a predefined parameter.

## Audit Goals

Verify that the GhoStewardV2 acts as described in the usage here.

## Findings Summary

We didn’t find any bugs/issues with the GhoStewardV2.

# Formal Verification

In the table below we specify all the formally verified rules and give a detailed description for them. A link to the Certora's prover report can be found here.

## Properties

Note: Some of the rules listed below depend on the following parameters that are defined in the file GhoStewardV2.sol:

- MINIMUM_DELAY: current value is 2 days.
- GHO_BORROW_RATE_CHANGE_MAX: current value is 5%.
- GHO_BORROW_RATE_MAX: current value is 25%.
- GSM_FEE_RATE_CHANGE_MAX: current value is 0.5%.

| ID | Rule Name | Description |
|---|---|---|
| 1 | updateGhoBorrowCap_timelock | The function *updateGhoBorrowCap* cannot be called again in less than MINIMUM_DELAY.. |
| 2 | updateGhoBorrowRate_timelock | The function *updateGhoBorrowRate* cannot be called again in less than MINIMUM_DELAY. |
| 3 | updateGsmExposureCap_timelock | The function *updateGsmExposureCap* cannot be called again in less than MINIMUM_DELAY. (for every specific gsm.) |
| 4 | updateGsmBuySellFees_timelock | The function *updateGsmBuySellFees* cannot be called again in less than MINIMUM_DELAY. (for every specific gsm.) |
| 5 | only_RISK_COUNCIL_can_call__updateFacilitatorBucketCapacity | Only the RISK_COUNCIL can call the function *updateFacilitatorBucketCapacity*. |

| | | |
|---|---|---|
| 6 | only_RISK_COUNCIL_can_call__updateGhoBorrowCap | Only the RISK_COUNCIL can call the function *updateGhoBorrowCap*. |
| 7 | only_RISK_COUNCIL_can_call__updateGhoBorrowRate | Only the RISK_COUNCIL can call the function *updateGhoBorrowRate*. |
| 8 | only_RISK_COUNCIL_can_call__updateGsmExposureCap | Only the RISK_COUNCIL can call the function *updateGsmExposureCap*. |
| 9 | only_RISK_COUNCIL_can_call__updateGsmBuySellFees | Only the RISK_COUNCIL can call the function *updateGsmBuySellFees*. |
| 10 | only_owner_can_call__setControlledFacilitator | Only the owner can call the function *setControlledFacilitator*. |
| 11 | updateGhoBorrowCap__correctness | The rule checks that:<br>- When calling the function *updateGhoBorrowCap(newBorrowCap)*, the POOL's function *setBorrowCap(..)* is called with *newBorrowCap*.<br>- The update changes up to 100% upwards. |
| 12 | updateGhoBorrowRate__correctness | The rule checks that:<br>- When calling the function *updateGhoBorrowRate(newBorrowRate)*, the POOL's function *setReserveInterestRateStrategyAddress(strategy)* is called with a fixed rate strategy with rate *newBorrowRate*.<br>- the update changes up to GHO_BORROW_RATE_CHANGE_MAX upwards or downwards.<br>- Max value is GHO_BORROW_RATE_MAX. |

| | | |
|---|---|---|
| 13 | updateGsmExposureCap__correctness | The rule checks that:<br>- When calling the function *updateGsmExposureCap(gsm,newExposureCap)*, the gsm function *updateExposureCap(...)* is called with the value *newExposureCap*.<br>- the update changes up to 100% upwards. |
| 14 | updateGsmBuySellFees__correctness | The rule checks that:<br>- When calling the function *updateGsmBuySellFees(gsm,buyFee,sellFee)*, the gsm function *updateFeeStrategy(...)* is called with a *fixedFeeStrategy(buyFee, sellFee)*.<br>- The update changes up to GSM_FEE_RATE_CHANGE_MAX upwards (in both buy and sell individually). |

# Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

# About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.