CERTORA

# AAVE Token Verification

## 1. Summary

This document describes the specification and verification of V2 of the AAVE token using the Certora Prover. The work was undertaken from November 29th 2020 through December 2nd, 2020. The latest commit that was reviewed and run through the Certora Prover was 8de8658deca6a07a67dbe41777814970ed10644e.

The scope of the project was to specify and check the delegation mechanism in AAVE token V2 (AaveTokenV2.sol).

We found one high severity issue, that an address can lose power indefinitely by delegating to address 0. The Aave team promptly fixed the bug.

### 1.1. Issues found

| Bug | Rule broken | Description | Severity | Fix |
|---|---|---|---|---|
| Delegating to address 0 reduces snapshot value indefinitely | delegate reversibility | when a user sets 0 as the delegatee, the balance is removed from the snapshot's value to a black hole | High | Disallow address 0 as the delegatee, view no-delegatee as if the delegatee is the user themselves |

### 1.2. Technology overview

The Certora Prover is based on well-studied techniques from the formal verification community. Specifications define a set of rules that call into the contract under analysis and make various assertions about their behavior. These rules, together with the contract under analysis, are compiled to a logical formula called a verification condition, which is then proved or disproved by the SMT solver. If the rule is disproved, the solver also provides a concrete test case demonstrating the violation.

The rules of the specification play a crucial role in the analysis. Without good rules, only very shallow properties can be checked (e.g. that no assertions in the contract itself are violated). To make effective use of Certora Prover, users must write rules that describe the high-level properties they wish to check of their contract. Certora Prover cannot make any guarantees about cases that fall outside the scope of the rules provided to it as input. Thus, in order to understand the results of this analysis, one must carefully understand the specification's rules.

## 1.3. Disclaimer

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

We hope that this information is useful, but provide no warranty of any kind, express or implied. The contents of this report should not be construed as a complete guarantee that the AAVE token 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.

# 2. High Level Specification

## Notations

1. ✔ indicates the rule is formally verified on the latest commit. We write ✔* when the rule was verified on a simplified version of the code (or under some assumptions).
2. ❌ indicates the rule is violated in the version under test.
3. ✍ indicates the rule is not yet formally specified.
4. 🔁 indicates the rule is postponed **<other issue, low priority?>**.

## 2.1. Snapshot related

### 2.1.1. Block number written in snapshot is monotonically increasing, and is not in the "future". ✔*

- delegate ✔
- delegateBySig ✔
- delegateAll ✔*
- delegateAllBySig ✔*
- transfer ✔*
- transferFrom ✔*

### 2.1.2. (Aave team) Whenever a user executes a transfer() or delegate(), his snapshot count (and the snapshot count of the target) is increased (needs to be checked for both voting and proposition power). ✔

### 2.1.3. Once an address becomes a delegatee it must have a snapshot. ✔

## 2.2. Delegation Related

### 2.2.1. Delegation is reversible (basic): if a user A starts from no delegate, then delegates to B, then un-delegates (by delegating to 0) - the un-delegate operation should succeed and the updated delegatee of A should be 0. ❌

- Bug found: if we delegate to 0 twice, the second call may revert, since the sender's snapshot underflows.The reason it happens is because when the delegatee is 0, delegation powers always moves from the account owner.

**2.2.2. Delegation is reversible (general): if a user A is currently delegating to B, then the un-delegate operation (by delegating to 0) should succeed and the updated delegatee of A should be 0. ✔**

**2.2.3. Delegation is idempotent: if we apply the delegate operation twice in two different blocks and no operations in-between, the delegatee and power of the delegator should not be changed. 🔁**

**2.2.4. If a user sets themselves as their own delegatee, this can be updated later. ✔**

**2.2.5. (Aave team) Whenever a user delegates his governance powers, his delegates address becomes the address of the delegatee (needs to be checked for both voting and proposition power). ✔**

**2.2.6. Only delegate* functions can update delegates ✔**

**2.2.7. A non-zero user that has a delegatee that is not 0 or the user themselves, must have a positive snapshot count. ✔**

**2.2.8. (Aave team) Delegating one type of governance power does not influence the other type. ✔**

## 2.3. Power related

**2.3.1. Only a subset of the functions can update the power of a user. ✔**

- transfer(address,uint256)
- transferFrom(address,address,uint256)
- delegate(address,uint8)
- delegateAll(address)
- delegateBySig(address,uint8,uint256,uint256,uint8,bytes32,bytes32)
- delegateAllBySig(address,uint256,uint256,uint8,bytes32,bytes32)

**2.3.2. Total power for all users in their last snapshots stays constant. ✔***

**2.3.3. If A has no delegates, then delegates to B. and B delegates to C, then when A is un-delegating the original power of A is restored. 🔁**

**2.3.4. (Aave team) When a user delegates his governance powers (both voting and proposition power) to another user, his getPowerCurrent() is 0 and the getPowerCurrent() of the delegatee is increased by his balance (property needs to be checked for delegate(), delegateAll(), delegateBySig() and delegateAllBySig()) ❌**

**2.3.5. (Aave team) When a user transfers tokens to another address, his powers (or his delegates powers) are decreased by the amount that the user is transferring out. ❌**

- There is a problem in the rule's formulation. If A delegates to B, and then transfer balance to B, B's powers are not expected to change (because they are at least A's balance + B's balance). The same occurs if A delegates to B, and B transfer balance to A.

**2.3.6. (Aave team) When a user receives tokens, the receiving address governance powers (or his delegates) are increased by the received amount. ❌**

- See 2.3.5.

**2.3.7. Reformulation of 2.3.5 and 2.3.6 - Sum of powers of "effective delegates" of A and B is preserved when A transfer to B (via transfer or transferFrom), and power difference for each "effective delegate" is updated accordingly. ✔**

**2.3.8. Reformulation of 2.3.4 - When a user delegates his governance powers (both voting and proposition power) to another user, and has no delegatee currently, his getPowerCurrent() is reduced by his balance, and the getPowerCurrent() of the delegatee is increased by his balance ✔**

## 2.4. Miscellaneous

**2.4.1. The expected functions increase a user's nonce, and by 1. ✔**

- delegateBySig(address,uint8,uint256,uint256,uint8,bytes32,bytes32)
- delegateAllBySig(address,uint256,uint256,uint8,bytes32,bytes32)
- permit(address,address,uint256,uint256,uint8,bytes32,bytes32)