CERTORA

# Formal Verification of aAave with delegation

## Summary

This document describes the specification and verification of tbe aAAVE token using the Certora Prover. The work was undertaken from 16th August 2023 to 7th September 2023. The latest commit reviewed and ran through the Certora Prover was f14d926.

The scope of this verification is the aAAVE token code which includes the following contracts:

- AToken.sol
- ATokenWithDelegation.sol
- IncentivizedERC20.sol
- MintableIncentivizedERC20.sol
- ScaledBalanceTokenBase.sol

The Certora Prover proved the protocol implementation is correct with respect to formal specifications written by the Certora team.

The resulting specification files are available on Aave's public git repository.

## Disclaimer

# Overview of the verification

## Assumptions and Simplifications

We made the following assumptions during the verification process:

- `getReserveNormalizedIncome` return a constant value of a whole RAY.
- Due to computational complexities, all properties of `AToken` are proven assuming `finalizeTransfer` and `handleAction` are never called.
- We unroll loops. Violations that require executing a loop more than three times will not be detected.
- When verifying contracts that make external calls, we assume that those calls can have arbitrary side effects outside of the contracts but do not affect the state of the contract being verified. This means that some reentrancy bugs may not be caught.

## Verification Conditions

### Notation

✓ indicates the rule is formally verified on the latest reviewed commit. Footnotes describe any simplifications or assumptions used while verifying the rules (beyond the general assumptions listed above).

This document shows verification conditions as logical formulas or Hoare triples of the form {p} C {q}. A verification condition given by a logical formula denotes an invariant that holds if every reachable state satisfies the condition.

Hoare triples of the form {p} C {q} hold if any non-reverting execution of program C that starts in a state satisfying the precondition p ends in a state satisfying the postcondition q. The notation {p} C@withrevert {q} is similar but applies to reverting and non-reverting executions. Preconditions and postconditions are similar to the Solidity require and assert statements.

Formulas relate the results of method calls. In most cases, these methods are getters defined in the contracts, but in some cases, they are getters we have added to our harness or definitions provided in the rules file. Undefined variables in the formulas are treated as arbitrary: the rule is checked for every possible value of the variables.

## Formal Properties

1. The `totalSupply` of the token is the sum of all users' balances. ✓

2. `Permit` sets the allowance as the user intended. ✓
3. Every possible operation changes the balance of at most two users. ✓
4. Mint to user `u` amount of `x` tokens, increases his `balanceOf()` the underlying asset by `x` and aToken `totalsupply` increase. ✓
5. Mint is additive. Performing a mint of an amount `y` following by a mint of amount `x` is the same as minting `x+y`. ✓
6. Burn scaled amount of Atoken from `user` and transfers amount of the underlying asset to `to`. ✓
7. Burn is additive. Performing a burn of an amount `y` following by a burn of amount `x` is the same as burning `x+y`. ✓
8. Burning one user atokens should have no effect on other users that are not involved in the action. ✓
9. Minting ATokens for a user should have no effect on other users that are not involved in the action. ✓
10. Successful permit function increases the nonce of owner by 1 and also changes the allowance of owner to spender. ✓
11. Verify that `metaDelegateByType` can only be called with a signed request. ✓
12. Verify that it's impossible to use the same arguments to call `metaDalegate` twice. ✓
13. Changing a delegate of one type doesn't influence the delegate of the other type. ✓
14. Verifying voting power increases/decreases while not being a voting delegatee yourself. ✓
15. Verifying proposition power increases/decreases while not being a proposition delegatee yourself. ✓
16. If an account is not receiving delegation of power (one type) from anybody, and that account is not delegating that power to anybody, the power of that account must be equal to its token balance. ✓
17. Verify correct voting power on token transfers, when both accounts are not delegating. ✓
18. Verify correct proposition power on token transfers, when both accounts are not delegating. ✓

19. Verify correct voting power after Alice delegates to Bob, when both accounts were not delegating. ✓
20. Verify correct proposition power after Alice delegates to Bob, when both accounts were not delegating. ✓
21. Verify correct voting power after a token transfer from Alice to Bob, when Alice was delegating and Bob wasn't. ✓
22. Verify correct proposition power after a token transfer from Alice to Bob, when Alice was delegating and Bob wasn't. ✓
23. Verify correct voting power after Alice stops delegating, when Alice was delegating and Bob wasn't. ✓
24. Verify correct proposition power after Alice stops delegating, when Alice was delegating and Bob wasn't. ✓
25. Verify correct voting power after Alice delegates. ✓
26. Verify correct proposition power after Alice delegates. ✓
27. Verify correct voting power after Alice transfers to Bob, when only Bob was delegating. ✓
28. Verify correct proposition power after Alice transfers to Bob, when only Bob was delegating. ✓
29. Verify correct proposition power after Alice transfers to Bob, when both Alice and Bob were delegating. ✓
30. Verify correct voting power after Alice transfers to Bob, when both Alice and Bob were delegating. ✓
31. Verify that an account's delegate changes only as a result of a call to the delegation functions. ✓
32. Verify that an account's voting and proposition power changes only as a result of a call to the delegation,transfer,mint,burn functions. ✓
33. Verify that only `delegate()` and `metaDelegate()` may change both voting and proposition delegates of an account at once. ✓
34. Verifies that delegating twice to the same delegate changes the delegate's voting power only once. ✓
35. transfer and `transferFrom()` change voting/proposition power identically. ✓

36. Test that transferFrom works correctly: - Balances are updated if not reverted. - If reverted, it means the transfer amount was too high, or the recipient is 0. ✓
37. Balance of address 0 is always 0. ✓
38. Contract calls don't change token total supply. ✓
39. Allowance changes correctly as a result of calls to `approve`, `transfer`, `increaseAllowance`, `decreaseAllowance`. ✓
40. Transfer from a to b doesn't change the sum of their balances. ✓
41. Transfer using `transferFrom()` from a to b doesn't change the sum of their balances. ✓
42. Transfer from `msg.sender` to alice doesn't change the balance of other addresses. ✓
43. Transfer from alice to bob using `transferFrom()` doesn't change the balance of other addresses. ✓
44. Balance of an address, who is not a sender or a recipient in transfer functions, doesn't decrease as a result of contract calls. ✓
45. User's delegation flag is switched on iff user is delegating to an address other than his own own or 0. ✓
46. Sum of delegated voting balances and undelegated balances is equal to total supply. ✓
47. Sum of delegated proposition balances and undelegated balances is equal to total supply. ✓
48. Transfers don't change voting delegation state. ✓