Specification

The main goal of integration is to provide the ability to deposit stETH into AAVE and allow to use it as collateral. Borrowing of the stETH (both stable and variable) is not supposed. The motivation behind this design is to encourage using stETH as collateral rather than borrowing it. stETH is pegged steadily to ETH, so using it as collateral involves low liquidation risks.

The stETH is implemented as a rebasing token. In normal conditions balances of users update one per day with Oracle report. Under the hood stETH stores balances of users as holder's shares in the total amount of ether controlled by the Lido protocol. stETH has pair of methods to convert inner shares into the balances and vice versa:

/// @return the amount of ether that corresponds to_sharesAmount token shares. function

getPooledEthByShares (uint256 _sharesAmount)

public

view

returns

(uint256) ;

/// @return the amount of shares that corresponds to_ethAmount protocol-controlled ether. function

getSharesByPooledEth (uint256 _ethAmount)

public

view

returns

(uint256) ; The aSTETH, similarly to regular aTokens, is a rewards-generating token that is minted and burned upon deposits and withdraws in the LendingPool. The aSTETH value is pegged to the value of the corresponding deposited token at a 1:1 ratio and can be safely stored, transferred, or traded. All interest collected by the aSTETH reserve (from rebasing and AAVE rewards) is distributed to aTokens holders directly by continuously increasing their wallet balance (in case of negative rebases of stETH it might decrease).

The aSTETH implementation guarantees the following always ensured:

- At any time, a user can deposit X stETH to mint X aSTETH *
- Total aSTETH supply increases by X.
- At any time, a user can burn x aSTETH for x stETH *
- The total aSTETH supply decreases by x.
- At any time, userA can transfer X aSTETH to userB *
- userA’s aSTETH balance reduces by X.
- userB’s aSTETH balance increases by X.
- The total aSTETH supply exactly remains the same.
- When stETH rebases, aSTETH rebases as well.
- Say there are 1000 stETH locked in the reserve. Consider the below situations: 1. Common case: happens positive rebase, and stETH total supply increases by 1%: - totalSupply of aSTETH token becomes equal to 1010 aSTETH. - balance of each aSTETH holder increases by 1% also. 2. Rare case: happens negative rebase, and stETH total supply decreases by 1%: - totalSupply of aSTETH token becomes equal to 990 aSTETH. - balance of each aSTETH holder decreases by 1% also.

* Actual amount of token will be less or equal to X because of integer operations rounding of underlying token rebase rate and AAVE interest rate. However, the actual rounding error will not exceed a couple of WEI at any time.

AStETH Token

To implement the above logicAStETH contract modifies the implementation of default aToken but keeps it as close as possible to the original contract. Same as defaultAToken contract it inherits fromVersionedInitializable ,IncentivizedERC20 contracts and implementsIAToken interface.

Default aToken implements the ERC20 interface but has two specific methods:

- scaledBalanceOf(user)
-
    - Returns thescaled balance
- of user as auint256

- . The scaled balance is the balance of the underlying token of the user (amount deposited), divided by the current liquidity index at the moment of the update. scaledBalance = amountDeposited/currentLiquidityIndex
- This essentially 'marks' when a user has deposited in the reserve pool and can be used to calculate the user's current compounded aToken balance.
- Example: - User A deposits 1000 DAI at the liquidity index of 1.1 - User B deposits another amount into the same pool - The liquidity index is now 1.2 - Therefore to calculate User A's current compounded aToken balance, the reverse operation should be performed: aTokenBalance = scaledBalance*currentLiquidityIndex
- scaledTotalSupply()
-
    - Returns the scaled total supply of the aToken asuint256
- .

But above approach can't be used with the stETH token without modifications because it doesn't take into consideration rebases of stETH.

If apply above equations to stETH as is, the staking profit will not be distributed across the aSTETH holders but will be accumulated on the balance of the aSTETH token.

To make rebases profit accountable,AStETH introduces an additional index -stETH rebasing index . The stETH rebasing index - express the rewards from rebases of stETH token in time. StETH rebasing index might be calculated as follows:

function

_stEthRebasingIndex ( )

returns

(uint256)

{ // Below expression returns how much ether corresponds // to 10 ** 27 shares. 10 ** 27 was taken to provide // same precision as AAVE's liquidity index, which // counted in RAY's (decimals with 27 digits). return stETH . getPooledEthByShares (WadRayMath . RAY) ; } With stETH rebasing index,AStETH allows to make rebases profit accountable, applying additional scaling when minting or burning of token happens:

function

mint (address user ,

uint256 amount ,

uint256 liquidityIndex)

{ . . . uint256 stEthRebasingIndex =

_stEthRebasingIndex ( ) ; _mint (user ,

_toInternalAmount (amount , stEthRebasingIndex , liquidityIndex)) ; . . . }

function

burn (address user ,

uint256 amount ,

uint256 liquidityIndex)

{ . . . uint256 stEthRebasingIndex =

_stEthRebasingIndex ( ) ; _burn (user ,

_toInternalAmount (amount , stEthRebasingIndex , liquidityIndex)) ; . . . }

function

_toInternalAmount (uint256 amount , uint256 stEthRebasingIndex , uint256 aaveLiquidityIndex)

internal

view

returns

(uint256)

{ return amount . mul (WadRayMath . RAY) . div (stEthRebasingIndex) . rayDiv (aaveLiquidityIndex) ; } Then, according to AAVE's definitions,scaledTotalSupply() andscaledBalanceOf() might be calculated as:

function

scaledTotalSupply ( )

returns

(uint256)

{ return _totalSupply . mul (_stEthRebasingIndex ( )) . div (WadRayMath . RAY) ; }

function

scaledBalanceOf (address user)

returns

(uint256)

{ return _balances [user] . mul (_stEthRebasingIndex ( )) . div (WadRayMath . RAY) ; } Additionally,AStETH contract introduces the following methods:

- internalBalanceOf(user)
-
    - returnsinternal balance
- of the user. The internal balance is the balance of the underlying token of the user (sum of deposits of the user), divided by the current liquidity index at the moment of the update and by the current stETH rebasing index.
- internalTotalSupply()
-
    - Returns the internal total supply of the aSTETH.

function

internalTotalSupply (address user)

returns

(uint256)

{ return _totalSupply ; }

function

internalBalanceOf (address user)

returns

(uint256)

{ return _balances [user] ; }

StableDebtSTETH & VariableDebtSTETH Tokens

The current integration doesn't support borrowing, neither with variable nor with stable interest rates. Because of that, the StableDebtSTETH and VariableDebtSTETH contract extends defaultStableDebtToken andVariableDebtToken contracts accordingly, and overridemint() method with the stub, which reverts with errorCONTRACT_NOT_ACTIVE . This was done to make it impossible to use borrowing with aSTETH because default debt tokens are not compatible with theAStETH contract.

In the future, borrowing might be activated, by updating the implementation of debt tokens. ButStableDebtToken andVariableDebtToken contractsMUST NOT be used withAStETH because they don't take into consideration rebases of stETH token and will break the math of the integration.

Incentives Controller

At the launch of the stETH integration in the AAVE protocol, the incentives controller is not supposed to be used. If in the future Lido decides to add incentives to the integration, it might be done via updating the implementation of aSTETH token. The example of implementation ofIncentivesController forAStETH contract might be found here:https://github.com/lidofinance/aave-asteth-incentives-controller