# $\mathcal{LTV}$:
# Curatorless Leveraged Tokenized Vault with a Constant Target Loan-To-Value Ratio

Andrey Sobol    Vladyslav Burtsevych    Illia Abrosimov

**Abstract**

The proposed system is a Curatorless Leveraged Tokenized Vault (LTV) with a Constant Target Loan-To-Value (LTV) ratio. This vault operates without a central curator and allows users to deposit and withdraw funds while receiving tokenized shares representing their holdings. The architecture is based on two interconnected EIP4626 vaults. To ensure alignment with the target LTV, an auction-based stimulus system is employed, which incentivizes users to participate in rebalancing actions through rewards or fees. This approach also integrates basic level of MEV protection to guard against frontrunning and maintain system integrity.

# Contents

# 1 Title explanation

The system being introduced is an $\mathcal{LTV}$: Curatorless Leveraged Tokenized Vault with a Constant Target Loan-To-Value Ratio.

- **Vault** is a protocol where users can deposit and withdraw money.
- **Tokenized**[1][2] means that a user gets transferable shares when they deposit and send to the vault shares during withdrawals.
- **Leveraged** means that the vault uses two types of assets: one is used as collateral, and the other is borrowed. The vault deposits the collateral asset into a lending protocol and borrows the other asset. This creates leverage by using borrowed funds in addition to the original deposit, allowing the user to have higher exposure for collateral assets than the initial deposit.
- **Constant Target Loan-To-Value ratio**[3] means that the vault has a constant value as a target for LTV and tries to keep LTV as close to the target as possible.
- **Curatorless** means that the vault can operate without a curator. In other words, it can be fully autonomous.
- $\mathcal{LTV}$ means **L**everaged **T**okenized **V**ault and **L**oan-**T**o-**V**alue ratio at the same time. From this point forward, LTV will exclusively denote the Loan-to-Value ratio, omitting its application as a Leveraged Tokenized Vault.

# 2 Motivation

Leverage vaults are commonly used in DeFi to open and maintain leveraged positions in lending protocols, simplifying the user experience. As of the date of publication, more than 20 vaults with a total TVL exceeding 100,000 ETH have been deployed in the Ethereum ecosystem, utilizing over 9 lending protocols as leverage sources. These include vaults for leveraged yield farming strategies (e.g., Instadapp Lite ETH v2 vault [4], CIAN LST vaults [5]), as well as leveraged long/short vaults designed for trading (e.g., Index Coop)

However, the strategy of such vaults is executed via manual curation. As a result, vaults with such architecture struggle to scale on multiple asset pairs due to the manual effort required for position monitoring, risk assessment, and optimization. Additionally, they carry counterparty risks, such as human error and lack of transparency, forcing users to trust curators without verification of optimal strategy execution.

This paper introduces the design of curatorless leverage vaults, offering scalability through a permissionless architecture, similar to Uniswap V2 [6] pools. This allows anyone to deploy a leverage vault for any pair of assets, eliminating the need for manual oversight.

We consider curatorless leverage vaults to become a new DeFi primitive that will be reused as a basic building block for more complex products (e.g. leverage aggregators with automatic refinancing mechanisms). It can also be used as a convenient tool for interest rate arbitrage including a cross-chain one, due to the tokenization of leverage positions.

In the current paper, we focus on leverage vaults for correlated asset pairs (e.g. ETH LST[7] [8] to ETH, or yield-bearing stablecoins to stablecoins[9] [10]). Leverage for non-correlated pairs requires additional soft liquidation mechanisms which are planned for future work.

# 3 Design intuition

## 3.1 Overview

Leveraged vault is designed to manage **collective leverage position** for a selected **pair of assets** at a selected **target leverage** level using a selected **lending protocol**.

The leveraged position is achieved via recursive borrowing [11] [12] [13] [14] [15] [16]. When a borrowing asset is borrowed, it is converted into a collateral asset, deposited into a lending protocol as collateral, which allows borrowing more of the borrowing asset, converting it again into collateral, and so on. This way we can achieve leverage exposure. However, the vault does not directly participate in the execution of recursive borrowing. It only provides the interface and incentives for users to maintain the leverage position.

Each time the user interacts with a vault (depositing or withdrawing funds), it changes the ratio of collateral assets to the borrowed asset [3], thereby changing the leverage level. To push LTV as close to the target as possible, the vault uses **auctions** [17] [18] with the **incentives** system. If user action pushes the vault LTV towards the target - it will be rewarded. If user action pushes the vault LTV away from the target LTV - it will be charged with an additional fee and the vault will open an auction that will align leverage to a target level once executed.

Assuming some level of efficiency in the market (due to a lot of MEV searchers existing) [19] [20] we can expect such a vault to always keep the target leverage level by adjusting the position after every user interaction. Therefore, the design of the auctions and incentives system became the most crucial part of the vault design and was meticulously described in this paper and formally proven.

## 3.2 Desired properties

- Implementation of two interconnected EIP4626 [2] vaults, featuring four core functions: deposit, withdraw, mint, and redeem for two distinct assets: collateral and borrow.

- Maintenance of a constant target LTV [3], ensuring that after each operation, the LTV remains aligned with the predefined target, thus enabling automatic rebalancing.

- Rebalancing costs are exclusively borne by users interacting with the vault, rather than all share holders, minimizing the potential for vault value depletion.

- The system is designed for bidirectional (deposits and withdrawals) operations with minimal limitations, facilitating flexible asset management.

- Incorporation basic level of MEV protection [19] [20] measures to guard against frontrunning activities.

## 3.3 Equation of global balance

$$\frac{borrow + \Delta borrow}{collateral + \Delta collateral} = targetLTV$$

This is the main equation of the system. When a user performs an operation, they must satisfy this equation. Here, $\Delta$ represents the change from the user interaction, while variables without delta represent the current state of the vault.

To consistently return to the $targetLTV$, auction mechanisms, and incentives are employed.

## 3.4 Auction stimulus system

To be MEV [19] protected, incentives must be implemented through an auction system rather than direct incentives. Therefore, auctions are used. The auction system is designed for rebalancing the vault to maintain a constant $LTV$. Auctions [17] [18] are triggered when there is a deviation from the $targetLTV$. Users receive incentives (rewards) for participating in the auctions and helping to rebalance the vault.

Almost every interaction with the vault will impact the auction system. If a user wants to make a move that leads to a better $LTV$ (closer to $targetLTV$), they will receive an incentive. If a user wants to make a move that leads to a worse $LTV$, they will pay a fee. This is how the system maintains the $targetLTV$.

### 3.5 Rebalance flow example

Here is an example that showcases rebalance flow in one of the most common cases. We assume LST-ETH:ETH price is 1:1.

1. Given LST-ETH/ETH x4 leverage vault (4x leverage means that $targetLTV$ is 0,75) in the ideal state ($LTV$ equals $targetLTV$).

   Vault state:

   | collateral | debt | LTV | Taget LTV |
   |---|---|---|---|
   | 40 LST-ETH | 30 ETH | 0,75 | 0,75 |

2. The user deposits 1 ETH into the vault

   (a) The vault will immediately repay 1 ETH debt in the lending protocol.

   (b) Within the same transaction, the vault will open an auction for exchanging $\sim$ 4 ETH into $\sim$ 4 LST-ETH to return to the target LTV (The user will be charged a fee that will be used as an incentive for auction).

   Vault state:

   | collateral | debt | LTV | Taget LTV |
   |---|---|---|---|
   | 40 LST-ETH | $\sim$ 29 ETH | 0,66 | 0,75 |

3. The auction is executed by arbitrage bots At the time of auction execution the vault borrowed $\sim$ 4 ETH from the lending protocol, sold it for $\sim$ 4 LST-ETH with the incentive to arbitrage bot, and deposited obtained $\sim$ 4 LST-ETH to the lending protocol.
   Vault return to the ideal state (real LTV equals target LTV)

   Vault state:

   | collateral | debt | LTV | Taget LTV |
   |---|---|---|---|
   | $\sim$ 44 LST-ETH | $\sim$ 33 ETH | 0,75 | 0,75 |

# 4 Logical notation framework

## 4.1 Boolean values

In this paper, we adopt a binary representation for logical values, where the value 0 is used to represent a $false$ statement, and the value 1 is used to represent a $true$ statement. This convention will be consistently followed throughout the paper to facilitate the formalization of logical expressions and simplify the computational interpretation of Boolean functions.

That means that these statements are correct:

$$(7 > 5) = true = 1$$

$$(6 < 5) = false = 0$$

## 4.2 Operator *sum* of logical values

In this paper, we can *sum* logical values. For example:

$$(5 < 6) + (6 < 7) + (7 < 8) = 3$$

The result of this *sum* is equal to 3 because all three statements are true.

## 4.3 Operator *and*

In our notation, we utilize the *and* operator (denoted by $\wedge$ ) to represent the Boolean conjunction. The conjunction operation results in $true$ only when both operands are $true$ and $false$ otherwise.

## 4.4 Operator *or*

In our notation, we utilize the *or* operator (denoted by $\vee$ ) to represent the Boolean disjunction. The disjunction operation results in $true$ when at least one of the operands is $true$ and $false$ otherwise.

## 4.5 Operator $\neg$

In our notation, we utilize the $\neg$ operator to represent the negation of a logical value. The negation operation inverts the logical value of the operand, transforming $true$ into $false$ and $false$ into $true$.

## 4.6 Sign of factors

In the inequality $factorA \times factorB > 0$ indicates that both factors, $factorA$ and $factorB$, share the same sign. In the inequality $factorA \times factorB < 0$, it indicates that the factors, $factorA$ and $factorB$, have opposite signs.

# 5 Constants

## 5.1 LTV constants

- $targetLTV$: The target Loan-to-Value ratio (LTV)[3] represents the ratio of the borrowed value to the value of the collateral. The goal is to maintain the LTV in lending protocol at the ratio specified in $targetLTV$.
- $minProfitLTV$: The minimum profitable LTV is the lowest LTV the vault can temporarily accommodate during an auction.
- $maxSafeLTV$: The maximum safe LTV is the highest LTV the vault can temporarily accommodate during an auction.

$$0 < minProfitLTV < targetLTV < maxSafeLTV < 1$$

## 5.2 Slippage

*slippage*: The slippage coefficient is a factor multiplied by the price to stimulate auction participant engagement and increase the probability of auction execution.

$$0 < slippage \ll 1$$

## 5.3 Auction size

$amountOfSteps$: the total number of steps in the auction process, where each step corresponds to a single block.

# 6 Price oracles

## 6.1 Time settings

- $t$: The time that ticks with every block
- $i$: The time that ticks with every user interaction.

## 6.2 Collateral and borrow prices

$PriceCollateral$ and $PriceBorrow$ are the prices of the collateral and borrow assets relative to a common value. Common value refers to the universal currency used to compare the prices of different assets.

$PriceCollateralOracle_t \in PriceCollateral$, $PriceBorrowOracle_t \in PriceBorrow$ are the prices of the collateral asset to common value and borrow asset to common value at time $t$ from the oracle [21][22].

$PriceCollateralDex_t \in PriceCollateral$, $PriceBorrowDex_t \in PriceBorrow$ are the prices of the collateral asset to common value and borrow asset to common value at time $t$ from the abstract DEX[6][23].

$SetPricesCollateralDex_t$ and $SetPricesBorrowDex_t$ are sets of prices of the collateral to common value and borrow to common value at time $t$ from the different DEXs.

$$SetPricesCollateralDex_t = \{PriceCollateralDex_t^* ...\}$$

$$SetPricesBorrowDex_t = \{PriceBorrowDex_t^* ...\}$$

### 6.3 Price impact into slippage

$$slippageCollateral = slippage \times mpc$$

$$mpc_t = max(max(SetPricesCollateralDex_t), PriceCollateralOracle_t)$$

$$slippageBorrow = slippage \times mpb$$

$$mpb_t = max(max(SetPricesBorrowDex_t), PriceBorrowOracle_t)$$

$slippageCollateral$, $slippageBorrow$ select the maximum among all DEX prices and the Oracle price at the time. The rationale behind using the maximum value is likely to ensure that the collateral or borrow is valued conservatively during slippage calculation, protecting against a depeg that might occur if one price source is temporarily higher than the market consensus.

If the price is calculated using at least one DEX and the Oracle, $slippageCollateral$ and $slippageBorrow$ select different price sources in the common case (except the case when all prices are equal). Because the source with the maximum price for the borrow asset will at the same time provide the minimum price for the collateral asset.

### 6.4 Lending protocol interest rate repayment and return

$\Delta BorrowRepayment_t$ is the amount of borrowed funds that the vault will pay to the lending protocol at time $t$. $\Delta CollateralReturn_t$ is the amount of collateral that the vault will get from the lending protocol at time $t$.

$$\Delta BorrowRepayment_t \geq 0$$

$$\Delta CollateralReturn_t \geq 0$$

$\Delta BorrowRepayment_t$ and $\Delta CollateralReturn_t$ represent different ways of expressing the interest rate in a lending protocol.

# 7 State

## 7.1 Auction

$auctionStepProportion$ is the proportion of the auction step that is used to calculate the protocol and the user rewards.

$$auctionStepProportion = \frac{auctionStep}{amountOfSteps}$$

$$auctionStep = \begin{cases} currentBlock - startAuction & \neg stuck \\ amountOfSteps & stuck \end{cases}$$

$startAuction$ is the block number when the auction starts. $startAuction$ can be rewritten afterward in case of merge.

$stuck$ is a boolean value that indicates whether an auction is stuck.

## 7.2 Borrow

The $borrow$ is what the protocol will owe to the lending protocol after the full execution of the auction, and all fee payments.

The $realBorrow$ is what the protocol owes to the lending protocol. The $realBorrow$ value must always be greater than or equal to 0.

$$realBorrow \geq 0$$

The $realBorrow$ can only fall below zero in one scenario: when an individual (referred to as the Good Samaritan) repays the vault's borrow and even contributes additional funds to it. In such cases, we assume that our protocol will treat the borrow as zero in lending protocols where applicable. Although this scenario will influence the $borrow$ value.

$futureBorrow$ is the value of borrowing that will change when the auction is fully executed. If $futureBorrow > 0$, the $realBorrow$ will increase. If $futureBorrow < 0$, the $realBorrow$ will decrease.

$futureRewardBorrow$ is the value of borrowing that will be distributed as a reward when an auction is fully executed. The $futureRewardBorrow$ will always be greater than or equal to 0.

$$futureRewardBorrow \geq 0$$

The $borrow$ can be represented as the sum of $realBorrow$, $futureBorrow$ and $FutureRewardBorrow$.

$$borrow = realBorrow + futureBorrow + futureRewardBorrow$$

$protocolFutureRewardBorrow$ is the value of borrowing that will be sent as protocol fee when the auction is fully executed. $protocolFutureRewardBorrow$ will always be greater than or equal to 0.

$$protocolFutureRewardBorrow \geq 0$$

$userFutureFeeBorrow$ is the value of borrowing that will be sent as a user reward when the auction is fully executed. $userFutureFeeBorrow$ will always be greater than or equal to 0.

$$userFutureFeeBorrow \geq 0$$

$futureRewardBorrow$ is always equal to the sum of $protocolFutureFeeBorrow$ and $userFutureFeeBorrow$.

$$\begin{aligned} futureRewardBorrow = \; & protocolFutureRewardBorrow + \\ & + \; userFutureRewardBorrow \end{aligned}$$

$$protocolFutureRewardBorrow = futureRewardBorrow \times (1 - auctionStepProportion)$$

$$userFutureRewardBorrow = futureRewardBorrow \times auctionStepProportion$$

$$stuck = currentBlock - startAuction > amountOfStep$$

### 7.3 Borrow assets

$realBorrowAssets_t$ is the amount of borrowed assets that the vault has at time $t$ according to lending protocol output.

$$realBorrowAssets_t = realBorrowAssets_{t-1} + \Delta BorrowRepayment_t$$

$$realBorrow_t = realBorrowAssets \times PriceBorrow_t$$

$futureBorrowAssets$ represents the amount of $futureBorrow$ in the borrowed assets.

$$futureBorrow_t = futureBorrowAssets_t \times PriceBorrow_t$$

$futureRewardBorrowAssets$ represents the amount of $futureRewardBorrow$ in the borrowed assets.

$$futureRewardBorrow_t = futureRewardBorrowAssets_t \times PriceBorrow_t$$

### 7.4 Collateral

The *collateral* is what the protocol will have as collateral in the lending protocol after the full execution of the auction and all fee payments. The *collateral* value must always be greater than or equal to 0.

$$collateral \geq 0$$

The *realCollateral* is what the protocol has as collateral in the lending protocol. The *realCollateral* value must always be greater than or equal to 0.

$$realCollateral \geq 0$$

$futureCollateral$ is the value of the collateral that will be adjusted when the auction is fully executed. If $futureCollateral > 0$, $realCollateral$ will increase. If $futureCollateral < 0$, $realCollateral$ will decrease.

$futureRewardCollateral$ is the value of the collateral that will be distributed as a reward when an auction is fully executed. $futureRewardCollateral$ is always less than or equal to 0.

$$futureRewardCollateral \leq 0$$

The *collateral* can be represented as the sum of $realCollateral$, $futureCollateral$ and $futureRewardCollateral$.

$$collateral = realCollateral + futureCollateral + futureRewardCollateral$$

$futureRewardCollateral$ is always equal to the sum of $protocolFutureRewardCollateral$ and $userFutureRewardCollateral$.

$$\begin{aligned} futureRewardCollateral = \ & protocolFutureRewardCollateral + \\ + \ & userFutureRewardCollateral \end{aligned}$$

$$protocolFutureRewardCollateral = futureRewardCollateral \times (1 - auctionStepProportion)$$

$$userFutureRewardCollateral = futureRewardCollateral \times auctionStepProportion$$

### 7.5 Collateral assets

$realCollateralAssets_t$ is the amount of collateral assets that the vault has at time $t$ according to lending protocol output.

$$realCollateralAssets_t = realCollateralAssets_{t-1} + \Delta CollateralReturn_t$$

$$realCollateral_t = realCollateralAssets \times PriceCollateral_t$$

$futureCollateralAssets$ represents the amount of $futureCollateral$ in the collateral assets.

$$futureCollateral_t = futureCollateralAssets_t \times PriceCollateral_t$$

$futureRewardCollateralAssets$ represents the amount of $futureRewardCollateral$ in the collateral assets.

$$futureRewardCollateral_t = futureRewardCollateralAssets_t \times PriceCollateral_t$$

### 7.6 Connection between rewards

If $futureRewardBorrow > 0$, then $futureRewardCollateral = 0$.

$$futureRewardBorrow > 0 \Rightarrow futureRewardCollateral = 0$$

If $futureRewardCollateral < 0$, then $futureRewardBorrow = 0$.

$$futureRewardCollateral < 0 \Rightarrow futureRewardBorrow = 0$$

Both can also be equal to 0.

$$(futureRewardBorrow > 0 \land futureRewardCollateral < 0) = false$$

## 7.7 Origin

The origin state that exists in the state of the lending protocol:

- $realBorrowAssets$
- $realCollateralAssets$

The origin state that exists in the smart contract state:

- $futureBorrowAssets$
- $futureCollateralAssets$
- $futureRewardBorrowAssets$
- $futureRewardCollateralAssets$
- $startAuction$

All other state variables are calculated on the fly in each time step $t$.

# 8 State transition variables

## 8.1 Borrow

$\Delta borrow$ is the amount of change in borrowing within the protocol. $\Delta borrow$ can be positive or negative.

$\Delta userBorrow$ is the aggregated value of the user's borrowed value. $\Delta userBorrow$ can be positive or negative.

$\Delta protocolFutureRewardBorrow$ is the value of borrowing that will be sent as a protocol fee. $\Delta protocolFutureRewardBorrow$ will always be less than or equal to 0.

$$\Delta protocolFutureRewardBorrow \leq 0$$

$$\Delta borrow = \Delta userBorrow + \Delta protocolFutureRewardBorrow$$

$\Delta realBorrow$: The $\Delta realBorrow$ represents the amount the user wants to borrow from the protocol. If $\Delta realBorrow > 0$, the user wants to withdraw and borrow; if $\Delta realBorrow < 0$, the user wants to deposit and decrease protocol borrow.

$\Delta futureBorrow$ represents the change in borrowing that needs to be applied to the auction amount.

$\Delta userFutureRewardBorrow$ is the amount of borrowing that will go to the user as a reward for the auction execution. $\Delta userFutureRewardBorrow$ will always be less than or equal to 0. Note that in this case, the reward for the user is not the same as the user's profit. To determine the real profit a user will receive, refer to the internal assumptions section.

$$\Delta userFutureRewardBorrow \leq 0$$

$\Delta futurePaymentBorrow$ is the value of borrowing that will be paid by the user for creating the auction. $\Delta userFutureRewardBorrow$ will always be greater than or equal to 0.

$$\Delta futurePaymentBorrow \geq 0$$

$$\begin{aligned}\Delta userBorrow =& \Delta realBorrow+\\ &+\Delta futureBorrow+\\ &+\Delta userFutureRewardBorrow+\\ &+\Delta futurePaymentBorrow\end{aligned}$$

## 8.2 Collateral

$\Delta collateral$ represents the change in the value of collateral within the protocol. $\Delta collateral$ can be positive or negative.

$\Delta userCollateral$ is the aggregate value of the user's collateral value. $\Delta userCollateral$ can be positive or negative.

$\Delta protocolFutureRewardCollateral$ is the value of collateral that will be sent as a protocol fee. $\Delta protocolFutureRewardCollateral$ will always be greater than or equal to 0.

$$\Delta protocolFutureRewardCollateral \geq 0$$

$$\Delta collateral = \Delta userCollateral + \Delta protocolFutureRewardCollateral$$

$\Delta realCollateral$: $\Delta realCollateral$ represents the value of collateral the user wants to change in the protocol. If $\Delta realCollateral > 0$, the user wants to deposit collateral; if $\Delta realCollateral < 0$, the user wants to withdraw collateral.

$\Delta futureCollateral$ represents the change in collateral that needs to be applied to the auction value.

$\Delta userFutureRewardCollateral$ is the value of the collateral that will be given to the user as a reward for auction execution. $\Delta userFutureRewardCollateral$ will always be greater than or equal to 0. Note that in this case, the reward for the user is not the same as the user's profit. To determine the real profit a user will receive, refer to the internal assumptions section.

$$\Delta userFutureRewardCollateral \geq 0$$

$\Delta futurePaymentCollateral$ is the value of the collateral that will be paid by the user for creating the auction. $\Delta futurePaymentCollateral$ will always be less than or equal to 0.

$$\Delta futurePaymentCollateral \leq 0$$

$$\begin{aligned}\Delta userCollateral = &\Delta realCollateral + \\ &+ \Delta futureCollateral + \\ &+ \Delta userFutureRewardCollateral + \\ &+ \Delta futurePaymentCollateral\end{aligned}$$

## 8.3 Shares and fees

$\Delta shares$ represent the change in shares that the user sends into the vault or receives into the vault. If $\Delta shares > 0$, the user will receive shares; if $\Delta shares < 0$, the user will send shares.

$$\Delta shares = \Delta userCollateral - \Delta userBorrow$$

$\Delta fee$ is the amount of fee that the vault will receive. $\Delta fee$ will always be greater than or equal to 0.

$$fee \geq 0$$

$$\Delta fee = \Delta protocolFutureRewardCollateral - \Delta protocolFutureRewardBorrow$$

## 8.4 $i + 1$ step transition

Every variable in the state can be represented as the sum of its previous state and the change in the variable.

$$realBorrow_{i+1} = realBorrow_i + \Delta realBorrow$$

$$realCollateral_{i+1} = realCollateral_i + \Delta realCollateral$$

$$futureBorrow_{i+1} = futureBorrow_i + \Delta futureBorrow$$

$$futureCollateral_{i+1} = futureCollateral_i + \Delta futureCollateral$$

$$\begin{aligned} futureRewardBorrow_{i+1} = \; & futureRewardBorrow_i + \\ & + \Delta futurePaymentBorrow + \\ & + \Delta userFutureRewardBorrow + \\ & + \Delta protocolFutureRewardBorrow \end{aligned}$$

| | |
|---|---|
| $\Delta futurePaymentBorrow \geq 0$ | $\Delta userFutureRewardBorrow \leq 0$ |
| | $\Delta protocolFutureRewardBorrow \leq 0$ |

Table 1: $futureRewardBorrow$ flow

$$
\begin{aligned}
futureRewardCollateral_{i+1} = & futureRewardCollateral_i + \\
& + \Delta futurePaymentCollateral + \\
& + \Delta userFutureRewardCollateral + \\
& + \Delta protocolFutureRewardCollateral
\end{aligned}
$$

| | |
|---|---|
| $\Delta futurePaymentCollateral \leq 0$ | $\Delta userFutureRewardCollateral \geq 0$ |
| | $\Delta protocolFutureRewardCollateral \geq 0$ |

Table 2: $futureRewardBorrow$ flow

## 8.5 Merging auction

Condition for merging an auction:

$$
\begin{aligned}
merge = & futureBorrow \times \Delta futureBorrow > 0 \wedge \\
& \wedge futureCollateral \times \Delta futureCollateral > 0
\end{aligned}
$$

If this condition is $false$, the auction is not merged; we do not change $startAuction_{i+1}$.

During the merging process, we assess the existing and new auctions using their respective $auctionWeight$ and $\Delta auctionWeight$.

$$
auctionWeight = \begin{cases} futureRewardBorrow & \Delta futurePaymentBorrow \neq 0 \\ futureRewardCollateral & \Delta futurePaymentCollateral \neq 0 \end{cases}
$$

$$
\Delta auctionWeight = \begin{cases} \Delta futurePaymentBorrow & \Delta futurePaymentBorrow \neq 0 \\ \Delta futurePaymentCollateral & \Delta futurePaymentCollateral \neq 0 \end{cases}
$$

$$
auctionStep_{i+1} = \left\lfloor \frac{auctionStep_i \times auctionWeight}{auctionWeight + \Delta auctionWeight} \right\rceil
$$

$$
\Delta startAuction_{i+1} = \begin{cases} currentBlock - auctionStep_{i+1} & merge \\ startAuction_i & \neg merge \end{cases}
$$

$startAuction_{i+1}$ can be adjusted retroactively.

# 9 7 systems of linear equations

## 9.1 Cases recitation

Our mathematical framework can be represented as 7 systems of linear equations, with each system of equations corresponding to a distinct case. These individual systems of equations are referred to as "cases" within the system.

- **cna**: **C**ase: **N**o **A**uction. In this case, the auction is not touched.
- **cmcb**: **C**ase: **M**erge (auction) **C**ollateral (to) **B**orrow.
- **cecb**: **C**ase: **E**xecute (auction) **C**ollateral (to) **B**orrow.
- **ceccb**: **C**ase: **E**xecute (auction and) **C**reate (new auction) **C**ollateral (to) **B**orrow.
- **cmbc**: **C**ase: **M**erge (auction) **B**orrow (to) **C**ollateral.
- **cebc**: **C**ase: **E**xecute (auction) **B**orrow (to) **C**ollateral.
- **cecbc**: **C**ase: **E**xecute (auction and) **C**reate (new auction) **B**orrow (to) **C**ollateral.

Each auction case can be systematically classified using two primary criteria. The first criterion distinguishes between cases "borrow to collateral" and "collateral to borrow". The second criterion differentiates among three operations: "merge", "execute", and "execute and create". The intersection of these two criteria yields six distinct cases. Additionally, there exists one case where the auction is not involved or touched - **cna**.

| | Merge | Execute | Execute and create |
|---|---|---|---|
| Collateral to borrow | **cmbc** | **cecb** | **ceccb** |
| Borrow to collateral | **cmbc** | **cebc** | **cecbc** |

Table 3: Classification of the 6 auction cases where the auction is touched

## 9.2 Case expressions

The following expressions define the conditions for each of the 7 cases.

### 9.2.1 *cna* constraint

The condition *cna* will be true if and only if $\Delta futureBorrow = 0$ and $\Delta futureCollateral = 0$, meaning that both the $futureBorrow_{i+1}$ and $futureCollateral_{i+1}$ remain unchanged.

$$cna = (\Delta futureBorrow = 0 \wedge \Delta futureCollateral = 0)$$

#### 9.2.2 *cmcb* constraint

The condition *cmcb* will be true if and only if the following 4 conditions are met: $futureBorrow \leq 0$, $futureCollateral \leq 0$, $\Delta futureBorrow < 0$, and $\Delta futureCollateral < 0$.

$$\begin{aligned} cmcb = (\ & futureBorrow \leq 0 \wedge \\ \wedge\ & futureCollateral \leq 0 \wedge \\ \wedge\ & \Delta futureBorrow < 0 \wedge \\ \wedge\ & \Delta futureCollateral < 0) \end{aligned}$$

$futureBorrow \leq 0$ and $futureCollateral \leq 0$ means that the case is "merge" type.

$\Delta futureBorrow < 0$ and $\Delta futureCollateral < 0$ means that the current case is "collateral to borrow" type.

#### 9.2.3 *cecb* constraint

The condition *cecb* will be true if and only if the following 6 conditions are met:

$$\begin{aligned} cecb = (\ & futureBorrow + \Delta futureBorrow \geq 0 \wedge \\ \wedge\ & futureCollateral + \Delta futureCollateral \geq 0 \wedge \\ \wedge\ & futureBorrow > 0 \wedge \\ \wedge\ & futureCollateral > 0 \wedge \\ \wedge\ & \Delta futureBorrow < 0 \wedge \\ \wedge\ & \Delta futureCollateral < 0) \end{aligned}$$

The first 4 conditions mean that the case is "execute" type.

$\Delta futureBorrow < 0$ and $\Delta futureCollateral < 0$ means that the current case is "collateral to borrow" type.

### 9.2.4 *ceccb* constraint

The condition *ceccb* will be true if and only if the following 6 conditions are met:

$$\begin{aligned} ceccb = (\ & futureBorrow + \Delta futureBorrow < 0 \wedge \\ & \wedge\ futureCollateral + \Delta futureCollateral < 0 \wedge \\ & \wedge\ futureBorrow > 0 \wedge \\ & \wedge\ futureCollateral > 0 \wedge \\ & \wedge\ \Delta futureBorrow < 0 \wedge \\ & \wedge\ \Delta futureCollateral < 0) \end{aligned}$$

The first 4 conditions mean that the case is "execute and create" type.

$\Delta futureBorrow < 0$ and $\Delta futureCollateral < 0$ means that the current case is "collateral to borrow" type.

### 9.2.5 *cmbc* constraint

The condition *cmcb* will be true if and only if the following 4 conditions are met: $futureBorrow \geq 0$, $futureCollateral \geq 0$, $\Delta futureBorrow > 0$, and $\Delta futureCollateral > 0$.

$$\begin{aligned} cmbc = (\ & futureBorrow \geq 0 \wedge \\ & \wedge\ futureCollateral \geq 0 \wedge \\ & \wedge\ \Delta futureBorrow > 0 \wedge \\ & \wedge\ \Delta futureCollateral > 0) \end{aligned}$$

$futureBorrow \geq 0$ and $futureCollateral \geq 0$ means that the case is "merge" type.

$\Delta futureBorrow > 0$ and $\Delta futureCollateral > 0$ means that the current case is "borrow to collateral" type.

### 9.2.6 *cebc* constraint

The condition *cebc* will be true if and only if the following 6 conditions are met:

$$\begin{aligned} cebc = (\ & futureCollateral + \Delta futureCollateral \leq 0 \wedge \\ & \wedge\ futureBorrow + \Delta futureBorrow \leq 0 \wedge \\ & \wedge\ futureBorrow < 0 \wedge \\ & \wedge\ futureCollateral < 0 \wedge \\ & \wedge\ \Delta futureBorrow > 0 \wedge \\ & \wedge\ \Delta futureCollateral > 0) \end{aligned}$$

The first 4 conditions mean that the case is "execute" type.
$\Delta futureBorrow > 0$ and $\Delta futureCollateral > 0$ means that the current case is "borrow to collateral" type.

### 9.2.7 *cecbc* constraint

The condition *cecbc* will be true if and only if the following 6 conditions are met:

$$\begin{aligned} cecbc = (\ & futureCollateral + \Delta futureCollateral > 0 \wedge \\ & \wedge\ futureBorrow + \Delta futureBorrow > 0 \wedge \\ & \wedge\ futureBorrow < 0 \wedge \\ & \wedge\ futureCollateral < 0 \wedge \\ & \wedge\ \Delta userFutureBorrow > 0 \wedge \\ & \wedge\ \Delta userFutureCollateral > 0) \end{aligned}$$

The first 4 conditions mean that the case is "execute and create" type.
$\Delta futureBorrow > 0$ and $\Delta futureCollateral > 0$ means that the current case is "borrow to collateral" type.

### 9.3 Exclusive constraint activation principle

In this exceptional case, the following condition holds true: among the constraints denoted by $cna$, $cmcb$, $cecb$, $ceccb$, $cmbc$, $cebc$, and $cecbc$, exactly one is active at any given time, while all others remain inactive. This implies that the system is governed by a unique exclusivity principle, whereby only one constraint is valid at a time.

Mathematically, we can express this relationship as:

$$cna + cmcb + cecb + ceccb + cmbc + cebc + cecbc = 1$$

This equation asserts that the sum of all constraints is always equal to one, ensuring that no more than one constraint is true simultaneously, while the remaining constraints must necessarily be false.

## 10 Case declarations

In each scenario, distinct interdependencies emerge among the following variables:

- $\Delta futureBorrow$
- $\Delta futureCollateral$
- $\Delta futurePaymentBorrow$
- $\Delta userFutureRewardBorrow$
- $\Delta protocolFutureRewardBorrow$
- $\Delta futurePaymentCollateral$
- $\Delta userFutureRewardCollateral$
- $\Delta protocolFutureRewardCollateral$

## 10.1 $cna$ declaration

In the $cna$ case, $\Delta futureBorrow$ is equal to $\Delta futureCollateral$ because the auction is not touched.

All other variables should be equal to zero in this case.

$$cna \Rightarrow \Delta futureBorrow = \Delta futureCollateral$$

$$cna \Rightarrow \Delta futurePaymentBorrow = 0$$

$$cna \Rightarrow \Delta userFutureRewardBorrow = 0$$

$$cna \Rightarrow \Delta protocolFutureRewardBorrow = 0$$

$$cna \Rightarrow \Delta futurePaymentCollateral = 0$$

$$cna \Rightarrow \Delta userFutureRewardCollateral = 0$$

$$cna \Rightarrow \Delta protocolFutureRewardCollateral = 0$$

## 10.2 *cmcb* declaration

In the *cmcb* case, $\Delta futureBorrow$ is equal to $\Delta futureCollateral$ because the auction is merged.

When merging the auction, the user should pay $-\Delta futureBorrow \times borrowSlippage$, or in other words, $\Delta futurePaymentBorrow$.

All other variables should be equal to zero in this case.

$$cmcb \Rightarrow \Delta futureBorrow = \Delta futureCollateral$$

$$cmcb \Rightarrow \Delta futurePaymentBorrow = -\Delta futureBorrow \times borrowSlippage$$

$$cmcb \Rightarrow \Delta userFutureRewardBorrow = 0$$

$$cmcb \Rightarrow \Delta protocolFutureRewardBorrow = 0$$

$$cmcb \Rightarrow \Delta futurePaymentCollateral = 0$$

$$cmcb \Rightarrow \Delta userFutureRewardCollateral = 0$$

$$cmcb \Rightarrow \Delta protocolFutureRewardCollateral = 0$$

## 10.3 *cecb* declaration

In the *cecb* case, $\Delta futureCollateral$ is equal to $\Delta futureBorrow \times \frac{futureCollateral}{futureBorrow}$ because the auction is executed.

When executing the auction, the user should get a reward $userFutureRewardCollateral \times \frac{\Delta futureCollateral}{futureCollateral}$, or in other words, $\Delta userFutureRewardCollateral$.

When executing the auction, the protocol should get a reward $protocolFutureRewardCollateral \times \frac{\Delta futureCollateral}{futureCollateral}$, or in other words, $\Delta protocolFutureRewardCollateral$.

All other variables should be equal to zero in this case.

$$cecb \Rightarrow \Delta futureCollateral = \Delta futureBorrow \times \frac{futureCollateral}{futureBorrow}$$

$$cecb \Rightarrow \Delta futureBorrow = \Delta futureCollateral \times \frac{futureBorrow}{futureCollateral}$$

$$cecb \Rightarrow \Delta futurePaymentBorrow = 0$$

$$cecb \Rightarrow \Delta userFutureRewardBorrow = 0$$

$$cecb \Rightarrow \Delta protocolFutureRewardBorrow = 0$$

$$cecb \Rightarrow \Delta futurePaymentCollateral = 0$$

$$cecb \Rightarrow \Delta userFutureRewardCollateral = userFutureRewardCollateral \times \\ \times \frac{\Delta futureCollateral}{futureCollateral}$$

$$cecb \Rightarrow \Delta protocolFutureRewardCollateral = protocolFutureRewardCollateral \times \\ \times \frac{\Delta futureCollateral}{futureCollateral}$$

### 10.4 *ceccb* declaration

In the *ceccb* case, $\Delta futureCollateral$ is equal to
$futureBorrow + \Delta futureBorrow - futureCollateral$
because the auction is execution.

When merging the auction, the user should pay
$-(\Delta futureBorrow + futureBorrow) \times borrowSlippage$,
or in other words, $\Delta futurePaymentBorrow$.

When executing the auction, the user should get a reward
$-userFutureRewardCollateral$, or in other words, $\Delta userFutureRewardCollateral$.

When executing the auction, the protocol should get a reward
$-protocolFutureRewardCollateral$, or in other words, $\Delta protocolFutureRewardCollateral$.

All other variables should be equal to zero in this case.

$$ceccb \Rightarrow \Delta futureCollateral = futureBorrow + \Delta futureBorrow - futureCollateral$$

$$ceccb \Rightarrow \Delta futureBorrow = futureCollateral + \Delta futureCollateral - futureBorrow$$

$$ceccb \Rightarrow \Delta futurePaymentBorrow = -(\Delta futureBorrow + futureBorrow) \times \\ \times borrowSlippage$$

$$ceccb \Rightarrow \Delta userFutureRewardBorrow = 0$$

$$ceccb \Rightarrow \Delta protocolFutureRewardBorrow = 0$$

$$ceccb \Rightarrow \Delta futurePaymentCollateral = 0$$

$$ceccb \Rightarrow \Delta userFutureRewardCollateral = -userFutureRewardCollateral$$

$$ceccb \Rightarrow \Delta protocolFutureRewardCollateral = -protocolFutureRewardCollateral$$

### 10.5 *cmbc* declaration

In the *cmbc* case, $\Delta futureBorrow$ is equal to $\Delta futureCollateral$ because the auction is merged.

When merging the auction, the user should pay
$-\Delta futureCollateral \times collateralSlippage$,
or in other words, $\Delta futurePaymentCollateral$.

All other variables should be equal to zero in this case.

$$cmbc \Rightarrow \Delta futureBorrow = \Delta futureCollateral$$

$$cmbc \Rightarrow \Delta futurePaymentBorrow = 0$$

$$cmbc \Rightarrow \Delta userFutureRewardBorrow = 0$$

$$cmbc \Rightarrow \Delta protocolFutureRewardBorrow = 0$$

$$cmbc \Rightarrow \Delta futurePaymentCollateral = -\Delta futureCollateral \times collateralSlippage$$

$$cmbc \Rightarrow \Delta userFutureRewardCollateral = 0$$

$$cmbc \Rightarrow \Delta protocolFutureRewardCollateral = 0$$

## 10.6 *cebc* declaration

In the *cebc* case, $\Delta futureCollateral$ is equal to $\Delta futureBorrow \times \frac{futureCollateral}{futureBorrow}$ because the auction is executed.

When executing the auction, the user should get a reward $userFutureRewardBorrow \times \frac{\Delta futureBorrow}{futureBorrow}$, or in other words, $\Delta userFutureRewardBorrow$.

When executing the auction, the protocol should get a reward $protocolFutureRewardBorrow \times \frac{\Delta futureBorrow}{futureBorrow}$, or in other words, $\Delta protocolFutureRewardBorrow$.

All other variables should be equal to zero in this case.

$$cebc \Rightarrow \Delta futureCollateral = \Delta futureBorrow \times \frac{futureCollateral}{futureBorrow}$$

$$cebc \Rightarrow \Delta futureBorrow = \Delta futureCollateral \times \frac{futureBorrow}{futureCollateral}$$

$$cebc \Rightarrow \Delta futurePaymentBorrow = 0$$

$$cebc \Rightarrow \Delta userFutureRewardBorrow = userFutureRewardBorrow \times \\ \times \frac{\Delta futureBorrow}{futureBorrow}$$

$$cebc \Rightarrow \Delta protocolFutureRewardBorrow = protocolFutureRewardBorrow \times \\ \times \frac{\Delta futureBorrow}{futureBorrow}$$

$$cebc \Rightarrow \Delta futurePaymentCollateral = 0$$

$$cebc \Rightarrow \Delta userFutureRewardCollateral = 0$$

$$cebc \Rightarrow \Delta protocolFutureRewardCollateral = 0$$

### 10.7 *cecbc* declaration

In the *cecbc* case, $\Delta futureCollateral$ is equal to $futureBorrow + \Delta futureBorrow - futureCollateral$ because the auction is executed.

When merging the auction, the user should pay $-(\Delta futureCollateral + futureCollateral) \times collateralSlippage$, or in other words, $\Delta futurePaymentCollateral$.

When executing the auction, the user should get a reward $-userFutureRewardBorrow$, or in other words, $\Delta userFutureRewardBorrow$.

When executing the auction, the protocol should get a reward $-protocolFutureRewardBorrow$, or in other words, $\Delta protocolFutureRewardBorrow$.

All other variables should be equal to zero in this case.

$$cecbc \Rightarrow \Delta futureCollateral = futureBorrow + \Delta futureBorrow - futureCollateral$$

$$cecbc \Rightarrow \Delta futureBorrow = futureCollateral + \Delta futureCollateral - futureBorrow$$

$$cecbc \Rightarrow \Delta futurePaymentBorrow = 0$$

$$cecbc \Rightarrow \Delta userFutureRewardBorrow = -userFutureRewardBorrow$$

$$cecbc \Rightarrow \Delta protocolFutureRewardBorrow = -protocolFutureRewardBorrow$$

$$cecbc \Rightarrow \Delta futurePaymentCollateral = -(\Delta futureCollateral + futureCollateral) \times \\ \times collateralSlippage$$

$$cecbc \Rightarrow \Delta userFutureRewardCollateral = 0$$

$$cecbc \Rightarrow \Delta protocolFutureRewardCollateral = 0$$

## 11 Case unification

Combining all 7 cases into one system of equations using conditions and binary transformation of logical values we got the following system of equations:

$$
\begin{aligned}
&\Delta futureBorrow = \\
&\qquad = (cna + cmcb + cmbc + ceccb + cecbc) \times \Delta futureCollateral \; + \\
&\qquad + (cecb + cebc) \times \Delta futureCollateral \times \frac{futureBorrow}{futureCollateral} + \\
&\qquad + (ceccb + cecbc) \times (futureCollateral - futureBorrow)
\end{aligned}
$$

$$
\begin{aligned}
&\Delta futureCollateral = \\
&\qquad = (cna + cmcb + cmbc + ceccb + cecbc) \times \Delta futureBorrow \; + \\
&\qquad + (cecb + cebc) \times \Delta futureBorrow \times \frac{futureCollateral}{futureBorrow} + \\
&\qquad + (ceccb + cecbc) \times (futureBorrow - futureCollateral)
\end{aligned}
$$

$$
\begin{aligned}
&\Delta futurePaymentBorrow = \\
&\qquad = cmcb \times -\Delta futureBorrow \times borrowSlippage \; + \\
&\qquad + ceccb \times -(\Delta futureBorrow + futureBorrow) \times borrowSlippage
\end{aligned}
$$

$$
\begin{aligned}
&\Delta userFutureRewardBorrow = \\
&\qquad = cebc \times userFutureRewardBorrow \times \frac{\Delta futureBorrow}{futureBorrow} + \\
&\qquad + cecbc \times -userFutureRewardBorrow
\end{aligned}
$$

$$
\begin{aligned}
&\Delta protocolFutureRewardBorrow = \\
&\qquad = cebc \times protocolFutureRewardBorrow \times \frac{\Delta futureBorrow}{futureBorrow} + \\
&\qquad + cecbc \times -protocolFutureRewardBorrow
\end{aligned}
$$

$$
\begin{aligned}
&\Delta futurePaymentCollateral = \\
&\qquad = cmbc \times -\Delta futureCollateral \times collateralSlippage \; + \\
&\qquad + cecbc \times -(\Delta futureCollateral + futureCollateral) \times collateralSlippage
\end{aligned}
$$

$$\begin{aligned}
&\Delta userFutureRewardCollateral = \\
&\qquad = cecb \times userFutureRewardCollateral \times \frac{\Delta futureCollateral}{futureCollateral} + \\
&\qquad + ceccb \times -userFutureRewardCollateral
\end{aligned}$$

$$\begin{aligned}
&\Delta protocolFutureRewardCollateral = \\
&\qquad = cecb \times protocolFutureRewardCollateral \times \frac{\Delta futureCollateral}{futureCollateral} + \\
&\qquad + ceccb \times -protocolFutureRewardCollateral
\end{aligned}$$

# 12 Custom auction math

For the auction, the $targetLTV$ constant rule is not followed. The custom math for the auction is described here.

$\Delta realBorrow^{\alpha}$ and $\Delta realCollateral^{\alpha}$ represent the amount of collateral and borrow a user receives after the auction execution.

$$\Delta realBorrow^{\alpha} = \Delta futureBorrow^{\alpha} + \Delta userRewardBorrow^{\alpha}$$

$$\Delta realCollateral^{\alpha} = \Delta futureCollataral^{\alpha} + \Delta userRewardCollateral^{\alpha}$$

## 12.1 Auction calculation

The proportion of borrow and collateral in the auction is described by the following equation:

$$\frac{\Delta futureBorrow^{\alpha}}{\Delta futureCollataral^{\alpha}} = \frac{futureBorrow}{futureCollataral}$$

The stimulus for this auction is $\Delta userRewardBorrow^{\alpha}$ and $\Delta userRewardCollateral^{\alpha}$.

$$\Delta userRewardBorrow^{\alpha} = userRewardBorrow \times \frac{\Delta futureBorrow^{\alpha}}{futureBorrow}$$

$$\Delta userRewardCollateral^{\alpha} = userRewardCollateral \times \frac{\Delta futureCollataral^{\alpha}}{futureCollataral}$$

## 12.2 Auction custom fee

The difference between a common fee is that the fee will be paid not in shares but in collateral or borrow assets.

$$\Delta protocolRewardCollateral^{\alpha} = protocolRewardCollateral \times \frac{\Delta futureCollataral^{\alpha}}{futureCollataral}$$

$$\Delta protocolRewardBorrow^{\alpha} = protocolRewardBorrow \times \frac{\Delta futureBorrow^{\alpha}}{futureBorrow}$$

### 12.3 Auction $i+1$ state transition

Auction $i+1$ state transition is very similar to regular math.

$$realBorrow_{i+1} = realBorrow_i + \Delta realBorrow^{\alpha} + \\ + \Delta protocolFutureRewardBorrow^{\alpha}$$

$$realCollateral_{i+1} = realCollateral_i + \Delta realCollateral^{\alpha} + \\ + \Delta protocolFutureRewardCollateral^{\alpha}$$

$$futureBorrow_{i+1} = futureBorrow_i + \Delta futureBorrow^{\alpha}$$

$$futureCollateral_{i+1} = futureCollateral_i + \Delta futureCollateral^{\alpha}$$

$$futureRewardBorrow_{i+1} = futureRewardBorrow_i + \\ + \Delta userFutureRewardBorrow^{\alpha} + \\ + \Delta protocolFutureRewardBorrow^{\alpha}$$

$$futureRewardCollateral_{i+1} = futureRewardCollateral_i + \\ + \Delta userFutureRewardCollateral^{\alpha} + \\ + \Delta protocolFutureRewardCollateral^{\alpha}$$

### 12.4 Auction assets variables

Asset variables are very similar to regular math.

$$\Delta protocolRewardBorrow_t^{\alpha} = \Delta protocolRewardBorrowAssets_t^{\alpha} \times PriceBorrow_t$$

$$\Delta protocolRewardCollateral_t^{\alpha} = \Delta protocolRewardCollateralAssets_t^{\alpha} \times PriceCollateral_t$$

$$\Delta realBorrow_t^{\alpha} = \Delta realBorrowAssets_t^{\alpha} \times PriceBorrow_t$$

$$\Delta realCollateral_t^{\alpha} = \Delta realCollateralAssets_t^{\alpha} \times PriceCollateral_t$$

# 13 Smart contract functions

## 13.1 $maxBorrow$ and $maxCollateral$ limits

Before any function, except $borrowAuction$ and $collateralAuction$, should check that $borrow$ and $collateral$ are within the limits.

$$borrow + \Delta borrow < maxBorrow \land collateral + \Delta collateral < maxCollateral$$

## 13.2 Apply state changes

If there are no errors, after any function except $borrowAuction$ and $collateralAuction$, the smart contract should follow these steps:

- Print or burn new $sharesAssets$
- Transfer $realBorrowAssets$ from the user or to the user
- Transfer $realCollateralAssets$ from the user or to the user
- Transfer $fee$ to the fee collector if it exists
- Save all variables

## 13.3 Low-level function *exchange*

The main idea of the low-level function *exchange* is to provide the ability to perform deposit and withdrawal operations with the best exchange rate.

Implicit input for the *exchange* function:

$$futureBorrow_{i+1}^{exchange} = 0$$

$$futureCollateral_{i+1}^{exchange} = 0$$

Explicit input for the *exchange* function: $sharesAssets$, which can be greater than zero, equal to zero, or less than zero.

Calculated using the system of equations, the output is $realBorrowAssets$, $realCollateralAssets$ (both can be greater than zero, equal to zero, or less than zero), and all other variables.

$$\begin{aligned} sharesAssets^{exchange} &> 0 \lor \\ \lor\, sharesAssets^{exchange} &< 0 \lor \\ \lor\, sharesAssets^{exchange} &= 0 \end{aligned}$$

$$\begin{aligned} realBorrowAssets^{exchange} &> 0 \lor \\ \lor\, realBorrowAssets^{exchange} &< 0 \lor \\ \lor\, realBorrowAssets^{exchange} &= 0 \end{aligned}$$

$$\begin{aligned} realCollateralAssets^{exchange} &> 0 \lor \\ \lor\, realCollateralAssets^{exchange} &< 0 \lor \\ \lor\, realCollateralAssets^{exchange} &= 0 \end{aligned}$$

## 13.4 EIP4626 functions

### 13.4.1 Classification

EIP-4626 is a standard for tokenized Vaults in Ethereum that is primarily designed for yield-bearing tokens. It provides a common interface for tokenized vaults.

Functions of EIP-4626 for borrowing subvault:

- $deposit$
- $withdraw$
- $mint$
- $redeem$

Functions of EIP-4626 for collateral subvault:

- $deposit^{collateral}$
- $withdraw^{collateral}$
- $mint^{collateral}$
- $redeem^{collateral}$

### 13.4.2 $realBorrow$ limits

In the functions $deposit$, $mint$, $deposit^{collateral}$, and $mint^{collateral}$, it is necessary to ensure that the new real LTV exceeds $minProfitLTV$.

$$minProfitLTV \leq \frac{realBorrow + \Delta realBorrow}{realCollateral + \Delta realCollateral}$$

In the functions $withdraw$, $redeem$, $withdraw^{collateral}$, and $redeem^{collateral}$, it is necessary to ensure that the new real LTV remains below $maxSafeLTV$:

$$\frac{realBorrow + \Delta realBorrow}{realCollateral + \Delta realCollateral} \leq maxSafeLTV$$

## 13.5 EIP4626 borrow subvault functions

In this section, the implementation of four EIP4626 functions is described: *deposit*, *withdraw*, *mint*, and *redeem*.

*previewDeposit*, *previewWithdraw*, *previewMint*, *previewRedeem*, *maxDeposit*, *maxWithdraw*, *maxMint*, and *maxRedeem* will be implemented in the same way as *deposit*, *withdraw*, *mint*, and *redeem* functions, but without any changes to the state. There's no need to describe them separately; they are trivial.

For the collateral subvault, the $preview*$ and $max*$ functions will be omitted for the same reasons mentioned earlier.

### 13.5.1 Implicit input *realCollateral*

The implicit input for all four functions is $realCollateral$ equal to 0.

$$realCollateral^{deposit,withdraw,mint,redeem} = 0$$

### 13.5.2 Function *deposit*

The explicit input for the *deposit* function is $-realBorrowAssets$.

$$realBorrowAssets^{deposit} < 0$$

Calculated using a system of equations output: $sharesAssets$.

$$\text{sharesAssets} < 0 \Rightarrow \text{sharesAssets}^{deposit} = 0$$

#### 13.5.3 Function *withdraw*

The explicit input for the *withdraw* function is $realBorrowAssets$.

$$realBorrowAssets^{withdraw} > 0$$

Calculated using a system of equations output: $-sharesAssets$.

$$\text{sharesAssets} > 0 \Rightarrow \text{sharesAssets}^{withdraw} = 0$$

#### 13.5.4 Function *mint*

The explicit input for the *mint* function is $sharesAssets$.

$$sharesAssets^{mint} > 0$$

Calculated using a system of equations output: $-realBorrowAssets$.

#### 13.5.5 Function *redeem*

The explicit input for the *redeem* function is $-sharesAssets$.

$$sharesAssets^{redeem} < 0$$

Calculated using a system of equations output: $realBorrowAssets$.

## 13.6 EIP4626 collateral subvault functions

### 13.6.1 Implicit input $realBorrow$

The implicit input for all four functions is $realBorrow$, which is equal to 0.

$$realBorrow^{deposit^{collateral}, withdraw^{collateral}, mint^{collateral}, redeem^{collateral}} = 0$$

### 13.6.2 Function $deposit^{collateral}$

The explicit input for the $deposit^{collateral}$ function is $realCollateralAssets$.

$$realCollateralAssets^{deposit^{collateral}} < 0$$

Calculated using a system of equations output: $sharesAssets$.

$$\text{sharesAssets} < 0 \Rightarrow \text{sharesAssets}^{deposit^{collateral}} = 0$$

### 13.6.3 Function $withdraw^{collateral}$

The explicit input for the $withdraw$ function is $-realCollateralAssets$.

$$realCollateralAssets^{withdraw^{collateral}} > 0$$

Calculated using a system of equations output: $-sharesAssets$.

$$\text{sharesAssets} > 0 \Rightarrow \text{sharesAssets}^{withdraw^{collateral}} = 0$$

### 13.6.4 Function $mint^{collateral}$

The explicit input for the $mint^{collateral}$ function is $sharesAssets$.

$$sharesAssets^{mint^{collateral}} > 0$$

Calculated using a system of equations output: $realCollateralAssets$.

### 13.6.5 Function $redeem^{collateral}$

The explicit input for the $redeem^{collateral}$ function is $-sharesAssets$.

$$sharesAssets^{redeem^{collateral}} < 0$$

Calculated using a system of equations output: $-realCollateralAssets$.

## 13.7 Auction function

### 13.7.1 Limitations for auction size for borrow

$$
\begin{aligned}
abl = {} & \Delta realBorrow^{\alpha} \geq -futureBorrow^{\alpha} - userRewardBorrow^{\alpha} \wedge \\
& \wedge futureBorrow > 0 \wedge \\
& \wedge \Delta realBorrow^{\alpha} < 0
\end{aligned}
$$

$$
\begin{aligned}
abg = {} & \Delta realBorrow^{\alpha} \leq -futureBorrow^{\alpha} - userRewardBorrow^{\alpha} \wedge \\
& \wedge futureBorrow < 0 \wedge \\
& \wedge \Delta realBorrow^{\alpha} > 0
\end{aligned}
$$

The main limitation for the borrow variables and auction size is described below:

$$ab = abl \vee abg$$

### 13.7.2 Limitations for auction size for collateral

$$
\begin{aligned}
acl = {} & \Delta realCollateral^{\alpha} \geq -futureCollateral^{\alpha} - userRewardCollateral^{\alpha} \wedge \\
& \wedge futureCollateral > 0 \wedge \\
& \wedge \Delta realCollateral^{\alpha} < 0
\end{aligned}
$$

$$
\begin{aligned}
acg = {} & \Delta realCollateral^{\alpha} \leq -futureCollateral^{\alpha} - userRewardCollateral^{\alpha} \wedge \\
& \wedge futureCollateral < 0 \wedge \\
& \wedge \Delta realCollateral^{\alpha} > 0
\end{aligned}
$$

The main limitation for the collateral variables and auction size is described below:

$$ac = acl \vee acg$$

### 13.7.3 Common auction size limitations

The final limitations for the auction size are applied to the functions $borrowAuction$ and $collateralAuction$:

$$ab \wedge ac$$

### 13.7.4 Borrow auction

The explicit input for the $borrowAuction$ function is $\Delta realBorrowAssets^{\alpha}$. Calculated output: $\Delta realCollateralAssets^{\alpha}$. Custom auction math is applied to the $borrowAuction$ function.

### 13.7.5 Collateral auction

The explicit input for the $collateralAuction$ function is $\Delta realCollateralAssets^{\alpha}$. Calculated output: $\Delta realBorrowAssets^{\alpha}$. Custom auction math is applied to the $collateralAuction$ function.

# 14 Security assumptions

## 14.1 Security assumption level 0

Math is correct. The universal rules of algebra are correct. We are not idiots and crazy people.

## 14.2 Environment

The designated blockchain for deployment exhibits robust security with no identified vulnerabilities.

The network maintains a singular, unified chain with no forks or reorganizations.

The network and blockchain uphold a commitment to censorship resistance, ensuring unimpeded communication and transactional integrity.

The third-party implementations exhibit comprehensive security integrity with no detected vulnerabilities and no malicious actors in the following areas:

- Lending protocol [11] [12] [13] [14] [15] [16]
- Oracle protocol [21] [22]
- DEX protocol [6] [23]

### 14.3 Internal assumptions

The variable $maxSafeLTV$ is sufficiently calibrated to avoid liquidation. Should there be uncertainty regarding its adequacy, a soft liquidation mechanism should be designed.

The parameter $minProfitLTV$ is optimized to sufficiently incentivize user engagement with the protocol.

The parameters $maxBorrow$ and $maxCollateral$ are strategically established to define protocol boundaries, mitigating excessive borrowing ratios and minimizing the risk of liquidation.

Within the range of blocks from 0 to $amoutOfSteps$, there will be at least one block where $userProfit$ will be sufficiently large to justify executing the current auction.

$$\begin{aligned} userProfit = & futureBorrow + \\ & + userFutureRewardBorrow + \\ & - futureCollateral - \\ & - userFutureRewardCollateral \end{aligned}$$

Outside the vault, there is enough liquidity to move it to the $targetLTV$, especially in the case of a Good Samaritan attack.

The user will execute a post-validation for the asset amount and initiate a reversal if there is any alteration in the protocol's state. To avoid MEV.

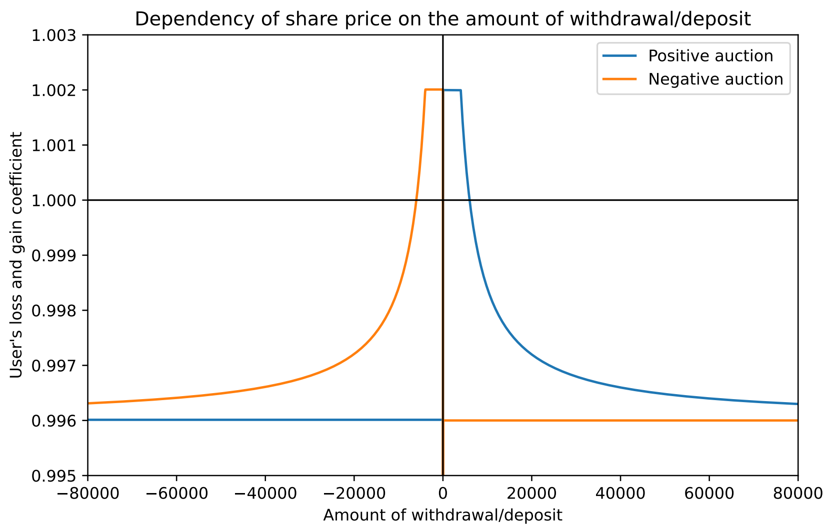
# 15 Results

## 15.1 Different stimuli for users in the case of positive and negative auctions

The implemented mathematical model shows different scenarios of economic incentives for the user. In the case of a positive auction, if $futureBorrow > 0$ and $futureCollateral > 0$, the user will be incentivized to deposit more assets.

In the case of a negative auction, if $futureBorrow < 0$ and $futureCollateral < 0$, the user will be incentivized to withdraw more assets.

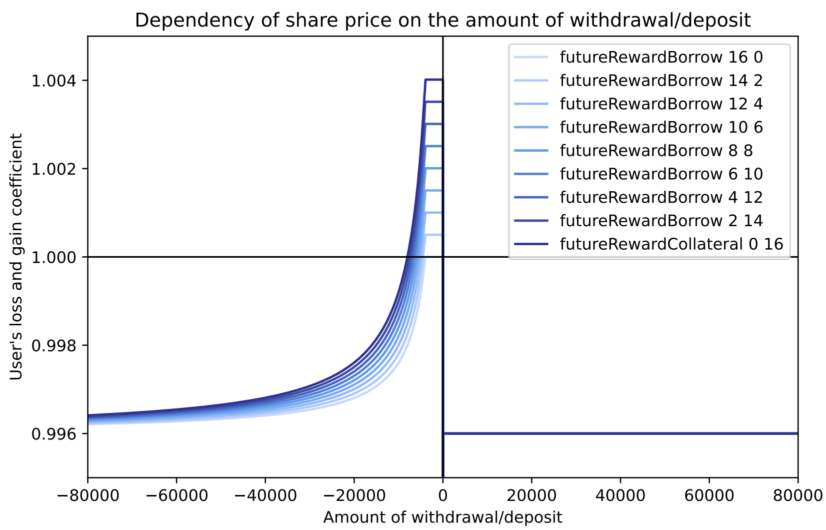
Until the auction is fully executed, the user will have a stable share price. After the auction is executed, the user will have a new share price. After that, the share price will tend to stabilize at the reverse operation's price.




| | Positive | Negative |
|---|---|---|
| $targetLTV$ | 0.75 | 0.75 |
| $realBorrow$ | 734000 | 765984 |
| $futureBorrow$ | 16000 | $-16000$ |
| $protocolFutureRewardBorrow$ | 0 | 8 |
| $userFutureRewardBorrow$ | 0 | 8 |
| $realCollateral$ | 984016 | 1016000 |
| $futureCollateral$ | 16000 | $-16000$ |
| $protocolFutureRewardCollateral$ | $-8$ | 0 |
| $userFutureRewardCollateral$ | $-8$ | 0 |

Table 4: Initial state of positive and negative auction

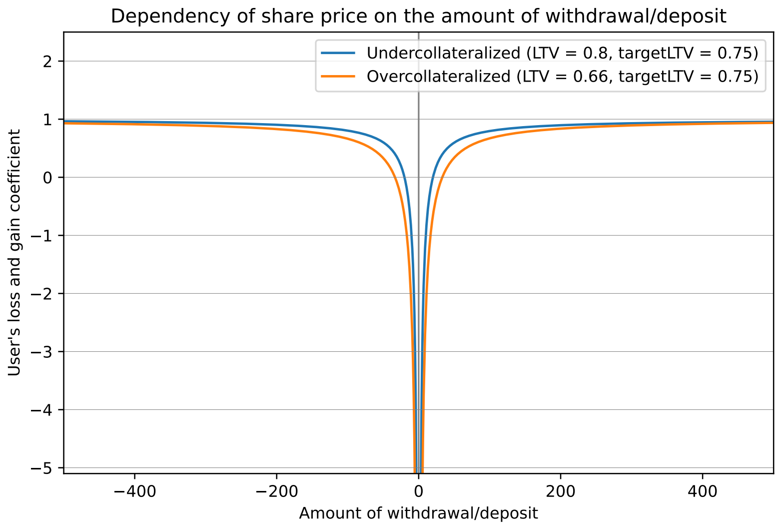
The stimulus will grow over time. With every new block, the user will have more and more stimulus to interact with the vault because the share price will remain stable until the auction is executed. However, the coefficient of the stimulus will become larger and larger until someone executes the auction.




The initial state is similar to Table 4. Only $protocolFutureRewardBorrow$ and $userFutureRewardBorrow$, are in the range of 0 to 16.

## 15.2 Undercollateralized and overcollateralized vault

If the vault is undercollateralized or overcollateralized, for small values of deposit and withdrawal, the user's gain will tend towards $-\infty$. This means that user interaction with the vault will be less efficient than buying or selling shares on the DEX market.




| | Undercollateralized | Overcollateralized |
|---|---|---|
| $targetLTV$ | 0.75 | 0.75 |
| $LTV$ | 0.8 | 0.66 |
| $borrow$ | 80000 | 66667 |
| $collateral$ | 100000 | 100000 |

Table 5: Initial state of the undercollateralized and overcollateralized vault

# 16 Future work

## 16.1 $targetLTV$ tradeoffs

To avoid the tendency towards $-\infty$ at small values of deposit and withdrawal, the following design choices can be made:

- Change $targetLTV$ to the range of $[minTargetLTV, maxTargetLTV]$.
- Enforce auctions only in the case of changes in $LTV$ moving in a worse direction.
- Set a function that considers deposit and withdrawal size and the effort required to reach the target LTV.

## 16.2 Protocol safety

To create a safe protocol in the case of non-correlated assets and unpredictable percentage rates, the following system should be designed:

- Soft liquidation to avoid liquidation risk.
- Algebraic or data-driven oracles for $maxBorrow$ and $maxCollateral$.

## 16.3 Slippage

To make slippage more efficient, slippage can be dynamic and can be changed:

- By applying some heuristics based on historical auction data.
- By voting of protocol token holders or vault token holders.

## 16.4 Auction

To improve the auction system, the following changes can be made:

- The auction function can be non-linear. There is a wide spectrum of possible functions.
- The auction can have parameters that will be changed by heuristics applied to historical data.
- The auction $auctionWeight$ can depend on $futureBorrow$ and $futureCollateral$, $\Delta futureBorrow$, $\Delta futureCollateral$, $protocolFutureRewardCollateral$, $protocolFutureRewardBorrow$, $userFutureRewardCollateral$, $userFutureRewardBorrow$, $\Delta futurePaymentBorrow$, $\Delta futurePaymentCollateral$.

### 16.5 Fee

Theoretically, the protocol can collect fees for the auction in one token, not in three. And a variety of different fees can be designed and applied:

- Maximum token growth fee
- Deposit and withdrawal fee
- Token transfer fee
- Token holding fee

### 16.6 Adaptive $LTV$ Management

To improve adaptability in response to market fluctuations and asset volatility, the protocol could implement:

- Dynamic $LTV$ thresholds that adjust based on real-time market conditions and percentage rates.
- An oracle-based condition for increasing or decreasing $LTV$.

### 16.7 Automated Liquidation Recovery

For the riskiest leveraged pairs, the protocol could implement an automated recovery system to address liquidation scenarios.

## 17 Conclusion

This paper introduces the Curatorless Leveraged Tokenized Vault (LTV) with a Constant Target Loan-to-Value ratio. The vault's architecture features a curatorless position rebalancing mechanism driven by auction incentives, ensuring the alignment of vault leverage with its target after each user interaction. Our analysis in the Results chapters demonstrates that the system performs as expected within its defined parameters.

The proposed design fulfills our requirements for a decentralized and permissionless architecture, but most importantly, a scalable system. By eliminating the need for manual curation, it enables the deployment of leverage vaults for any pair of correlated assets, as long as these assets have a leverage source (listed in a lending protocol). Furthermore, assuming the possibility of permissionless deployment of isolated lending pools, the entire process — from ERC-20 token creation to leverage token deployment — can be fully permissionless. This approach opens the ability to obtain leveraged exposure for any token, provided there is liquidity support. We believe that this design aligns deeply with DeFi principles, making Curatorless Leveraged Tokenized Vault a new building block in the DeFi ecosystem.

However, the proposed design still has room for improvement—refining the auction and slippage mechanics and enhancing protocol safety for diverse financial contexts. Additionally, we have deferred the design of leverage vaults for uncorrelated asset pairs, such as those used in trading strategies. This more complex approach will require the development of advanced soft liquidation mechanisms, which we intend to explore in future work.

## 18 Acknowledgments

The authors would like to thank Dima Bondar, Nick Schteringard, and Yurii Piadyk for their great ideas, valuable feedback, and suggestions.

Appreciation goes to the rest of the LTV protocol team (`https://ltv.finance`), who made the publication of this paper possible, and to Astroid Venture Studio (`https://astroid.xyz`) for sponsoring this research.

Special thanks to the developers of Aave and Compound for being pioneers of lending protocols in the DeFi ecosystem, and to Morpho Blue and Euler for their efforts in taking steps toward permissionless lending protocols.

## References


[1] EIP-20: ERC-20 Token Standard
https://eips.ethereum.org/EIPS/eip-20

[2] EIP-4626: Tokenized Vault Standard
https://eips.ethereum.org/EIPS/eip-4626

[3] Loan-to-value ratio
https://en.wikipedia.org/wiki/Loan-to-value_ratio

[4] Instadapp Lite
https://lite.guides.instadapp.io

[5] CIAN Yield Layer
https://docs.cian.app/getting-started-with-cian/yield-layer

[6] Uniswap V2
https://uniswap.org/whitepaper.pdf

[7] Rocket Pool 2.5 — Tokenised Staking
https://medium.com/rocket-pool/rocket-pool-2-5-tokenised-staking-48601d52d924

[8] Lido: Ethereum Liquid Staking
https://lido.fi/static/Lido:Ethereum-Liquid-Staking.pdf

[9] Curve stablecoin design
https://docs.curve.fi/assets/pdf/curve-stablecoin.pdf

[10] The Dai Stablecoin System
https://makerdao.com/whitepaper/DaiDec17WP.pdf

[11] Aave Protocol V2
https://github.com/aave/protocol-v2/blob/master/aave-v2-whitepaper.pdf

[12] Aave Protocol V3
https://github.com/aave/aave-v3-core/blob/master/techpaper/Aave_V3_Technical_Paper.pdf

[13] Compound Protocol
https://compound.finance/documents/Compound.Whitepaper.pdf

[14] Morpho Blue
https://github.com/morpho-org/morpho-blue/blob/main/morpho-blue-whitepaper.pdf

[15] Euler v1
https://docs-v1.euler.finance/getting-started/white-paper

[16] Ethereum Vault Connector (EVC)
https://evc.wtf/docs/whitepaper/

[17] Dutch Auction and First-Price Sealed-Bid Auction
http://www.econport.org/content/handbook/auctions/auctionsexperiments/auctionexpdutchandfirst.html

[18] Dutch auction
https://pi.math.cornell.edu/~mec/Winter2009/Spulido/Auctions/dutch.html

[19] Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges
https://arxiv.org/pdf/1904.05234.pdf

[20] Ethereum is a Dark Forest
https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest

[21] ChainLink. A Decentralized Oracle Network
https://research.chain.link/whitepaper-v1.pdf

[22] Uniswap Oracle
https://docs.uniswap.org/contracts/v2/concepts/core-concepts/oracles

[23] Curve StableSwap
https://docs.curve.fi/assets/pdf/stableswap-paper.pdf