# Lien FairSwap White Paper

Lien Protocol*

Version 1.0: April 27, 2020

**Abstract**

In our first white paper (Lien Protocol 2020), we proposed a new type of stable coin, *iDOL*. The price stability of iDOL is ensured by speculators who are willing to take on the risk associated with the price fluctuation of Ether (ETH). This is achieved by those speculators holding a derivative token, named *Liquid Bond Token* (LBT). However, as LBT is a brand new class of derivative token, currently there is no market where the token is actively traded.

In this paper, we propose *Lien FairSwap*, a decentralized exchange (DEX) platform that provides an opportunity to trade cryptoassets based on the Constant Product Market Maker model. The platform also incorporates the idea of frequent batch auction and is designed to be robust against front-running. Our design also introduces a mechanism for dynamically adjusting the commission rates based on the market condition. While the platform will specifically serve as a market for trading LBTs within the Lien protocol, other decentralized finance (DeFi) protocols may also benefit from the use of the core mechanisms utilized by the platform.

*Contact Author: lien-protocol@protonmail.com

# 1 Introduction

## 1.1 Speculators: Essential Part of the iDOL Ecosystem

In April 2020, the Lien Protocol released the white paper on the stable coin, *iDOL*. The stability of the iDOL price (i.e. its fiat exchange rate) is supported by the investors who insure the fiat exchange rate risk for a profit.

To supply a stable coin, Lien initially splits Ether into two parts, the *Solid Bond Token* (SBT) and the *Liquid Bond Token* (LBT). The LBT holder undertakes all the exchange-rate risk as long as the USD/ETH exchange rate remains higher than a certain threshold. Thanks to this "tranching", SBT becomes an "almost risk-free" asset, whose value denominated in USD is insensitive to changes in the USD/ETH exchange rate. Our stable coin, the iDOL token, is a representative money backed by a basket of SBTs, each of which is provided with unique configurations. See our white paper (Lien Protocol 2020) for more details on the iDOL ecosystem.

The iDOL token is minted when a user deposits an SBT to the iDOL contract. To deposit an SBT, the user must first create it by tranching Ether, which then generates an SBT and an LBT. LBT is an attractive asset for speculators as it allows them to engage in leveraged trading. Furthermore, leveraged trading with LBT is less risky than leveraging through debt finance because the speculator does not have to provide a collateral to conduct the leveraged transaction. Accordingly, there is no margin call involved even if the exchange rate (i.e. the price of ETH) falls significantly.

Given that a majority of people participating in the cryptocurrency market are holding cryptocurrencies while hoping to make a profit through price appreciation, we can expect to see large demand for LBTs coming from those speculators who want to hold LBTs to engage in leveraged trading. However, since LBT is a new type of derivative token, currently there is no market available where it is actively traded.

The iDOL ecosystem needs involvement of speculators who are motivated to hold LBTs.

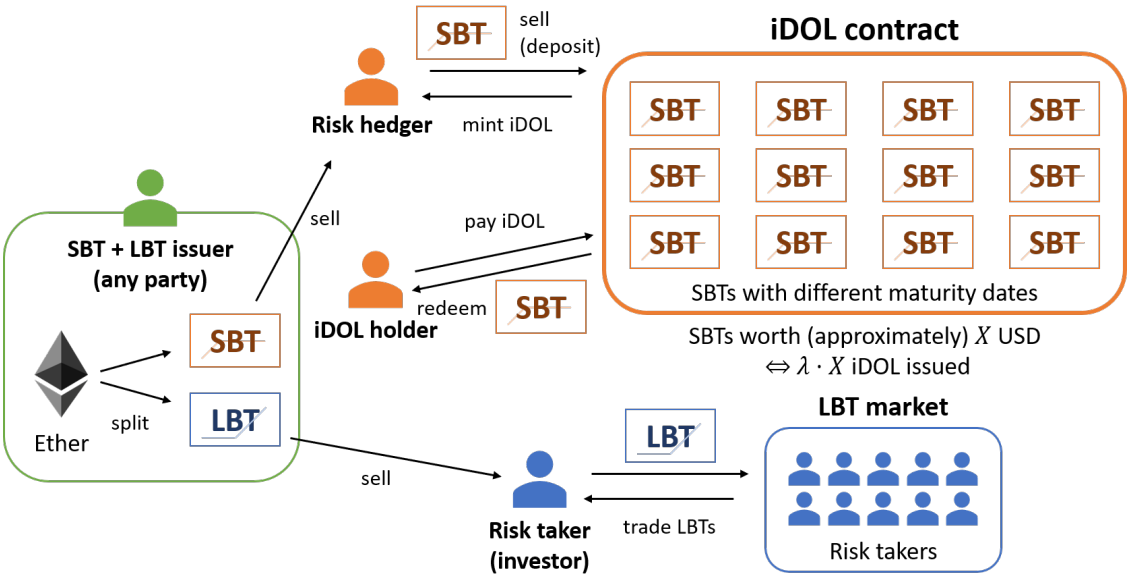
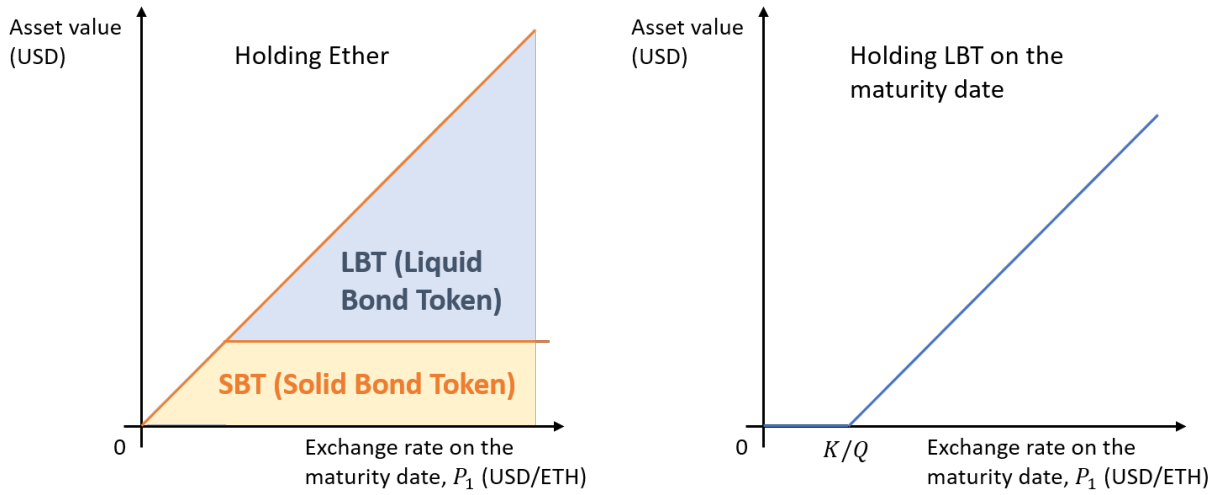



Figure 1: The iDOL ecosystem

For this reason, we provide a marketplace for trading LBTs in addition to a system for issuing a stable coin (iDOL). Here, because we are a team of anonymous individuals, the way we try to cultivate "trust" in the system is by developing the marketplace in the form of a *decentralized exchange* (DEX), a smart contract that allows users to trade cryptoassets. Extending the idea of the *Constant Product Market Maker model*, we develop a secure, efficient, and low-cost marketplace for trading LBTs. Although the marketplace is designed to provide liquidity for LBTs, it can also be applied to the market design of any other token as well.

## 1.2 LBT as a Leveraged Token

In the iDOL ecosystem, you can create a stable asset by splitting Ether into two derivative tokens: *Solid Bond Token* (SBT) and *Liquid Bond Token* (LBT). First, $Q$ ETH is deposited to a smart contract for the issuance of SBT and LBT. Then, on the maturity date, the contract returns a payout of $K$ USD to the SBT holder whenever possible, and returns the rest to the LBT holder. Specifically, when the USD/ETH exchange rate on the maturity date is $P_1$, the smart contract returns $\min\{K/P_1, Q\}$ ETH to the SBT holder, and distributes




(a) Decomposition of Ether

(b) Payout of LBT

Figure 2: By tranching Ether, we can create a stable asset (SBT) and a leveraged token (LBT).

the rest of the deposit, $Q - \min\{K/P_1, Q\} = \max\{Q - K/P_1, 0\}$ ETH, to the LBT holder. In other words, LBT functions as an European call option that gives its holder the right to buy $Q$ ETH at $K$ USD on the maturity date.

The value of SBT in USD is kept stable: as long as $Q \geq K/P_1$ or, equivalently, $P_1 \geq K/Q$, the value of the SBT will be $K/P_1$ ETH $= K$ USD on the maturity date. Then, we can create a basket of SBTs and have it back up the value of iDOL, the stable coin. For further details, read the iDOL white paper (Lien Protocol 2020).

The stability of the SBT value is based upon the instability of the LBT value. That is, all the risks related to the price volatility of Ether are undertaken by the LBT holder, which then makes the value of SBT stable. We can think of LBT as a leveraged derivative token, the value of which is associated with Ether but fluctuates much more than Ether.

While there already exist various ways to conduct leveraged trading in the cryptocurrency market, the characteristics of the LBT token described above will make it an attractive vehicle with which to speculate on the price development of Ether.

**Price** LBT holders can collect a "premium" from SBT holders or iDOL holders — because SBT holders and iDOL holders are risk hedgers who wish to hedge against the volatility of the price of Ether, they should be willing to pay an "insurance fee" to LBT holders who can take on the exchange-rate risk for a profit. By contrast, if an investor tries to establish a leveraged position through margin trading in a traditional market, he must *pay* commissions rather than being *paid* to undertake risks.

**No Margin Call** No debt finance is needed when you hold an LBT. For this reason, no margin call will be involved even if the price of Ether drops significantly. Therefore, the value of LBT never becomes negative even in the worst case scenario: you can simply release the token if it becomes worthless.

**No Counterparty Risk** To generate SBTs and LBTs, one needs to deposit Ether to a smart contract. After the derivative contract is run, it will automatically distribute the value of these tokens into their respective holders on the maturity date. Hence, there is no counterparty risk from the LBT holder's point of view.

It is also worth noting that leveraging through LBT would also contribute to financial stability in general — highly leveraged margin trading is one of the principal factors that could cause a financial crisis. When an asset value declines, the margin deposits are wiped out, forcing the investor to add to the deposit. Then, the investor has to sell more of his assets to keep liquidity, and the selling pressure causes the asset price to decline even further. However, LBT is free from this problem because its value does not rely on debt finance at all.

# 2 Preliminary: Constant Product Market

## 2.1 How it works

In the Constant Product Market Maker model, a market maker, also called a *liquidity provider*, provides an opportunity to exchange two assets. In a normal implementation, one of the assets is Ether (ETH) and the other is a financial asset token whose ownership is managed on the Ethereum blockchain (e.g. stable coins issued by various DeFi projects). We plan to open marketplaces for exchanging ETH, iDOL, and LBT with support for various maturity dates. However, for the sake of discussion, we will take a hypothetical token "ABC" and look at how the liquidity provider within this marketplace can allow us to "buy" and "sell" the ABC token in exchange for Ether.

The design of Lien FairSwap is based on the Constant Product Market Maker model, proposed in a reddit post by Vitalik Buterin.[1] Later, several decentralized finance (DeFi) projects implemented Buterin's idea and provided various DEX platforms based on this model, Uniswap being a primary example.

In contrast to the system designs adopted by traditional stock market exchanges, the asset price (e.g. the ETH/ABC exchange rate) is not "pre-specified" within the Constant Product Market Maker model. Instead, the marketplace algorithmically "adjusts" the price based on the inputs from buyers and sellers so as to eliminate arbitrage opportunities.

When a liquidity provider opens a marketplace, the provider pools these two traded assets (ETH and ABC). Any user can submit a buy order (i.e. send ETH to receive ABC) or a sell order (i.e. send ABC to receive ETH) in the marketplace. When a buy order is made, the user puts ETH into the ETH pool and takes ABC from the ABC pool. When the marketplace is closed, the liquidity provider redeems all the ETH and ABC tokens from the pool. Orders are written as transactions which will be sent through the Ethereum blockchain.

[1] https://www.reddit.com/r/ethereum/comments/55m04x/lets_run_onchain_decentralized_exchanges_the_way/, accessed on April 10th, 2020. See also Lu (2017) and Angeris, Kao, Chiang, Noyes, and Chitra (2019).

Now, assume that $X$ ETH and $Y$ ABC are currently pooled. The price of the ABC token (denominated in ETH) is determined by the ratio of $X$ to $Y$. To simplify, we ignore the commissions paid by users to the liquidity provider for now and focus on the basic idea behind how the Constant Product Market Maker model works.

Let us suppose that a user sends a buy order to the marketplace. To place a buy order, the user must transfer to the marketplace the amount of "money" (ETH) he wants to spend. Here, when the marketplace receives $\Delta x$ ETH from the user, it determines the amount of the ABC tokens the user can receive, $\Delta y$ ABC, by solving the following equation:

$$(X + \Delta x)(Y - \Delta y) = XY.$$

As you can see, the product of the two factors remains constant, which is why this model is called the *Constant Product* Market Maker model. The marketplace always holds constant the product of $X$ and $Y$, or the amount of ETH and ABC held by the marketplace. Solving the above equation for $\Delta y$, we have:

$$\Delta y = \frac{\Delta x \cdot Y}{X + \Delta x}.$$

Hence, the price (i.e. the price of ABC denominated in ETH) the user needs to pay to obtain the ABC token will be:

$$\frac{\Delta x}{\Delta y} = \frac{X + \Delta x}{Y} \text{ (ETH/ABC)}.$$

This value increases as $\Delta x$ increases. In other words, the more ABC the user wants to buy, the more ETH he has to pay. Hence, the *ask price*, which is the best possible price proposed by the market maker to a buyer of the ABC token, is $X/Y$ ETH/ABC.

We can derive the best price proposed to a seller in a similar manner. If the user transfers $\Delta y$ ABC to be sold in the marketplace, the amount of ETH provided by the marketplace

($\Delta x$ ETH) can be calculated with the following equation:

$$(X - \Delta x)(Y + \Delta y) = XY.$$

Solving this equation for $\Delta x$, we have:

$$\Delta x = \frac{X \cdot \Delta y}{Y + \Delta y}.$$

Hence, the price the marketplace pays to the user is:

$$\frac{\Delta x}{\Delta y} = \frac{X}{Y + \Delta y}$$

This value decreases as $\Delta y$ increases. As a result, the more ABC the user wants to sell, the lower the price of each token will be. Consequently, the *bid price*, which is the best possible price proposed by the market maker to a seller, is $X/Y$ ETH/ABC.

If we ignore the commissions paid to the validators and the liquidity provider, the Constant Product Market Maker model always allows for buying and selling the ABC token at the price of $X/Y$ ETH/ABC. If the price of the ABC token differs from $X/Y$ ETH/ABC outside this market, the user can take advantage of the arbitrage opportunity, resulting in the ratio of $X$ to $Y$ being fixed immediately, provided that the market is efficient. Accordingly, the price proposed in this model, i.e. $X/Y$ ETH/ABC, will always be adjusted to the broader market price. Therefore, the Constant Product Market Maker model can always propose a fair price to the users. Since $X/Y$ can take any value between 0 and $+\infty$, there is no risk of the ratio being "stuck" in a certain range or value. Hence, the marketplace can keep operating even when drastic changes in price occur.

Practically, the liquidity provider should take some commissions from users. Similar to market makers in the traditional stock exchange markets, the liquidity provider is providing a market for trading, thereby creating value for its users. If the users were allowed to benefit

from this option for free, the liquidity provider would incur losses, there being no incentive for people to provide liquidity. To address this problem, the liquidity provider should take some commissions every time a trade occurs.

*Remark* 1. When a liquidity provider launches a marketplace, he should choose $X$ and $Y$ in such a way that $X/Y$ coincides with the ETH/ABC exchange rate of the outside market. Otherwise, arbitragers immediately take advantage of the mispricing and exploit the liquidity provider until $X/Y$ reaches the exchange rate accepted in a broader market.

## 2.2 Front-Running

Although the Constant Product Market Maker model comes equipped with sound theoretical properties, its ideal implementation is not trivial due to several technological limitations within a blockchain system. If we naïvely implement the Constant Product Market Maker model, an arbitrager may exploit some of the inherent inefficiencies in DEX platforms.[2]

Let us take Uniswap as an example. In Uniswap, both buy and sell orders are processed one by one — every time the marketplace finds a new order, it updates the market price and adjusts the balance of the pool. Typically, multiple orders are contained in one block, and these orders (transactions) are sorted by the block creator. Uniswap processes transactions serially, in a deterministic order (a transaction that is located at the top of the block is processed earlier). This is the simplest implementation of the Constant Product Market Maker model that was described in the previous subsection.

To understand the vulnerability of Uniswap, we should be aware of the following properties associated with blockchain systems in general. First, transactions on a blockchain are processed in a discrete manner. The Ethereum network appends a new block every 10 to 20 seconds when its difficulty adjustment algorithm functions properly, which means that transactions are never processed immediately — even in the best case scenario, a user needs

[2] Daian, Goldfeder, Kell, Li, Zhao, Bentov, Breidenbach, and Juels (2019) study the activities of DEX arbitrage bots.

to wait for one new block to be added to the blockchain before his transaction can be processed.[3] Second, pending transactions are publicly visible. All the pending transactions are stored in a mempool, from which validators pick transactions to create a new block. Given that anyone is allowed to work as a validator in a public blockchain network, people can easily check the details of those pending transactions. As a result, they can observe the orders sent to Uniswap along with their details. Finally, users can control the order of transaction processing. Usually, the validators in the Ethereum network put transactions in descending order based on their gas prices, allowing people to select the relative position of their transactions within a new block by configuring the gas prices. Additionally, the validators can freely change the order of the transactions contained in the new block he created, if he so chooses.

These specifications of the Ethereum blockchain make it easy to exploit the Uniswap system; an arbitrager can take advantage of those characteristics to make a profit through *front-running*.

There arises an opportunity for front-running anytime a user (who is different from the arbitrager) conducts a high value transaction. For example, suppose that the ask price ($= X/Y$ ETH/ABC) proposed after the previous block had been processed was $P_0$ and, observing this, a user submitted a large buy order. The order, written as an Ethereum transaction request, is initially stored in the mempool. Once an arbitrager finds the transaction request, he can send buy orders and sell orders, both of which contain (approximately) the same volume. Therefore, after both transactions are processed, the arbitrager does not hold any position for the ABC token.

The arbitrager then adjusts the gas price in such a way that the transactions are processed in the following order: (i) the arbitrager's buy order, (ii) the user's buy order, and finally (iii) the arbitrager's sell order. Since the arbitrager's buy order is processed first, the ask price proposed to the arbitrager, $P_1 (> P_0)$, is close to the ask price of the previous block,

[3] Sometimes, the user would need to wait more because the validators cannot accommodate all transactions into the new block if there are many pending transactions.

$P_0$. Since the arbitrager made a large buy order, after (i) is processed, the price will have increased. Hence, the user is presented with a higher, and less desirable, price, $P_2 (> P_1 > P_0)$. Consequently, when the arbitrager's sell order, i.e. (iii), is processed, the arbitrager can benefit from a higher price than the original one — if the arbitrager sends a sufficiently small volume (slightly smaller than the volume of the user's buy order), the arbitrager can make a profit of $P_3$ ($P_2 > P_3 > P_1$) per token. Hence, the arbitrager can buy the ABC tokens at $P_1$ and sell them at $P_3$, effectively enjoying a "free lunch". On the other hand, the user ends up buying the ABC token at a higher price because of the arbitrager's activity. A platform that allows such an exploitation does not appeal to users because they might unwillingly lose their money.

## 2.3 Price Slippage Limit

The front-running problem happens because the price is adjusted too often compared with the frequency with which a block arrives. This allows an attacker to manipulate the price so as to exploit normal users. In the above example, when the user sent a buy order, he observed $P_0$ as the ask price. However, the actual price proposed to the user ($P_2$) turned out to be different from (and higher than) $P_0$ because transactions that had been processed earlier and were contained in the same block had influenced the price. This is called *price slippage*.

Uniswap mitigates the front-running problem by allowing users to cancel orders when the price slippage results in too large a price discrepancy. For example, in the above example, the user can refuse to buy the ABC tokens when the price is higher than $(1 + \epsilon)P_0$, where $\epsilon > 0$ is the maximum price slippage that can be allowed. Once the slippage limit is specified, the arbitrager is discouraged from buying a large amount of tokens because he cannot make a profit when the user's buy order is canceled. Even though arbitrage opportunity remains, the arbitrager has to act within the following constraint: $(1 + \epsilon)P_0 > P_2 > P_3 > P_1 (> P_0)$. Accordingly, if $\epsilon$ is close to zero, the profit an arbitrager can make as well as the loss the

user incurs will become close to zero.

This solution has a drawback. To exclude the front-running opportunities, the users must not tolerate price slippage. However, price slippage always occurs whenever new orders are processed. In an extreme case, all the successive transaction requests in the same block might be canceled after the first transaction is processed due to the first transaction causing price slippage. In such a system, the marketplace would be able to process only one transaction per each block. This means that only one user can make a trade for every 10 to 20 seconds. Accordingly, such a system fails to provide enough liquidity to the market.

# 3 Lien FairSwap

## 3.1 Frequent Batch Auction + Constant Product Market Maker Model

The front-running problem is not exclusive to DEX platforms and the Constant Product Market Maker model. The problem also exists in a traditional stock exchange that uses an order book. In fact, it has been reported that some high-frequency traders are exploiting normal users through high-speed algorithmic trading. (Lewis 2014 details the rise of high-frequency trading in the US market.)

A *frequent batch auction* proposed by Budish, Cramton, and Shim (2015) does not process orders sequentially, i.e. one by one. Instead, time is treated as a discrete parameter instead of a continuous measure, and orders are processed through *batch auctions* rather than sequentially. While traders have the ability to place their orders in an interval of milliseconds, the system itself does not allow transactions to be processed in milliseconds; the system might hold an auction every tenth of a second, for example. While the speed with which normal traders submit their orders is slower than high-frequency traders, an interval of a tenth of a second would give normal traders some room for making more reasonable decisions, thereby reducing opportunities for front-running.

The practice of front-running could cause more serious issues in a blockchain system. Since block time is much slower than transaction speeds observed in traditional financial markets, we do not actually need high-speed Internet connection at all in order to conduct front-running on a DEX platform. To cut into a line of orders, it suffices to set a higher gas price than the existing transactions. Accordingly, if the system processes transactions sequentially, anyone can easily execute front-running by operating a bot. Hence, to establish a safe and efficient DEX, it makes all the more sense to introduce a solution like frequent batch auction. In addition, due to the discrete nature of a blockchain system, it is impossible to process each transaction instantaneously, even if we want to design the system that way.

In this project, we establish Lien FairSwap, a DEX platform which incorporates the idea of frequent batch auction and which, we believe, is an improvement over the Constant Product Market Maker model. As in a Constant Product Market Maker model based system, the price is determined based on the balance of the ETH and ABC tokens in the pool in addition to the volumes of buy and sell orders. The difference between our model and the original Constant Product Market Maker model as implemented in Uniswap is that we do not process transactions serially. Rather, we form a batch of transactions and process them simultaneously. Just like a frequent batch auction, Lien FairSwap is designed to be robust against front-running.

## 3.2 Exchange Box

Because Lien FairSwap adopts the frequent batch auction mechanism, it does not process orders one by one. Instead, Lien FairSwap introduces an *exchange box*, which is a batch of all orders included in several neighboring blocks, and clears these orders simultaneously. Orders included in the same exchange box are cleared with the same price (more precisely, buyers will have to pay a slightly higher price than the one put forward by sellers because they must pay commissions to the liquidity provider. The same applies on the seller's side).

Batching eliminates the possibility of manipulating transaction order. Lien FairSwap does

not factor in the order of transactions — to the extent that transactions are included in the same exchange box, the order by which they have been sent does not matter. Therefore, arbitragers cannot exploit normal users by sandwiching transactions, making the front-running activities unprofitable.

It is worth emphasizing that an exchange box includes multiple blocks. Hence, even the validators cannot perfectly control which transactions are included into which exchange box unless he has the ability to create multiple blocks in a row, which is highly unlikely.

### 3.3 Clearing Price

From here, we will describe how the price and quantity are determined for each trade. The summary of the clearing procedure for exchange boxes is described in Subsection 3.6.

Since multiple transactions are contained in an exchange box, we must clear both buy and sell orders simultaneously. We therefore need to extend the clearance rule described in Subsection 2. Additionally, we will also incorporate the commissions into the model to provide a more precise picture of the market design.

Suppose that the liquidity provider currently has $X$ ETH and $Y$ ABC in the pool. Then, let us assume that we have buy orders whose volume is $\Delta X$ ($\geq 0$) ETH in total, and sell orders whose volume is $\Delta Y$ ($\geq 0$) ABC in total. Later, we will explain how we derive $\Delta X$ and $\Delta Y$ from the list of orders in an exchange box.

The buyers and sellers will be presented with the same exchange rate, $P$ ETH/ABC. Hence, the buyers get $\Delta X/P$ ABC in total whereas the sellers receive a total of $\Delta Y \cdot P$ ETH. The exchange rate $P$ is determined in such a way as to satisfy the following formula:

$$(X + \Delta X - \Delta Y \cdot P)(Y + \Delta Y - \Delta X/P) = XY. \tag{1}$$

The equation (1) requires that the multiple of the amount of the ETH tokens and the ABC tokens held by the liquidity provider be constant. The liquidity provider initially has $X$ ETH

and $Y$ ABC in the pool. In this exchange box, the buyers additionally send $\Delta X$ ETH to the pool (net of commissions), increasing the pool's supply. At the same time, since the sellers also send $\Delta Y$ ABC to receive the ETH tokens, the liquidity provider must transfer $\Delta Y \cdot P$ ETH to the sellers. In this exchange box, the sellers send $\Delta Y$ ABC (net of commissions), and the liquidity provider must deliver $\Delta X/P$ ABC to the buyers.

Solving (1), we obtain the following two solutions:

$$P = \frac{\Delta X}{\Delta Y} \quad , \quad \frac{X + \Delta X}{Y + \Delta Y}.$$

The former solution, $P = \Delta X/\Delta Y$, completely ignores the role of the liquidity provider and determines the price purely by comparing the amount of the buy and sell orders contained in the current exchange box. This is not a desirable solution. We cannot expect all exchange boxes to have the same balance between buy and sell orders. If the balance between $\Delta X$ and $\Delta Y$ in a given exchange box differs significantly from the one in another exchange box, either buyers or sellers in that exchange box will end up paying or receiving an unfair price. In addition, if either $\Delta X$ or $\Delta Y$ is zero, the resulting price will end up being an impractical value (i.e. zero or infinity). For this reason, we cannot adopt this price as the token's exchange rate.

The latter solution, $P = (X + \Delta X)/(Y + \Delta Y)$, corresponds to a generalization of the pricing in the basic Constant Product Market Maker model; when there is only one order included in the current exchange box, the price will be equal to the one derived in Section 2. If the liquidity provider has a sufficiently large pool available, then the values of $X$ and $Y$ will be much larger than the values of $\Delta X$ and $\Delta Y$, and the price will be stabilized around $X/Y$. As such, we adopt the latter solution.

## 3.4 Commissions and Aggregation Rule

To incentivize the liquidity provider to open a market, the marketplace should provide a reward in the form of commissions. Lien FairSwap subtracts commissions in each order.

Let $I$ and $J$ be the set of buy and sell orders, respectively, in the current exchange box. For each $i \in I$, $x_i \in \mathbb{R}_{++}$ denotes the amount of the ETH tokens submitted by the buyer $i$, and $y_j \in \mathbb{R}_{++}$ signifies the amount of ABC tokens provided by the seller $j$.

The marketplace subtracts a fixed percentage, $\gamma \in [0, 1)$, from the total amount and then aggregates all the transactions to calculate the values of $\Delta X$ and $\Delta Y$ as follows:

$$\Delta X = (1-\gamma) \cdot \sum_{i \in I} x_i,$$
$$\Delta Y = (1-\gamma) \cdot \sum_{j \in J} y_j.$$

The buyer $i$ receives $(1-\gamma)x_i/P$ ABC in exchange for $x_i$ ETH. As such, the actual price the buyer will pay is:

$$\frac{x_i}{(1-\gamma)x_i/P} = \frac{P}{1-\gamma} = \frac{1}{1-\gamma} \cdot \frac{X + \Delta X}{Y + \Delta Y} \quad \text{ETH/ABC}.$$

Similarly, the seller $j$ receives $(1-\gamma)y_i \cdot P$ ETH in exchange for $y_i$ ABC. Therefore, the actual price the seller will receive is:

$$\frac{(1-\gamma)y_i P}{y_i} = (1-\gamma) \cdot P = (1-\gamma) \cdot \frac{X + \Delta X}{Y + \Delta Y} \quad \text{ETH/ABC}.$$

Commissions are returned to the pool after the current exchange box is cleared. Thus, although the product is kept at a constant value after the clearance, its value will increase due to those commissions being added to the pool.

*Remark* 2. When LBT is included in a trading pair, we should not keep the commission rate $\gamma$ constant. Instead, we should adjust $\gamma$ for every exchange box based on the price volatility

of LBT. See Section 4 for further details.

### 3.5 Price Slippage Limit and Rationing

Thus far, we have assumed that all the users accept any price slippage. This assumption is impractical. Since the clearing price depends on the profile of the orders contained in a given exchange box, users cannot perfectly foresee the price they will pay or receive. If a user finds that the price slippage (i.e. the difference between the actual price $((X + \Delta X)/(Y + \Delta Y))$ and the benchmark price he observed $(X/Y)$) is too large, then he may want to cancel the transaction ex post. As such, even though our market design excludes the opportunity for front-running through a batching mechanism, users should also have the ability to cancel orders to avoid a large price slippage.

In Lien FairSwap, users can make two different types of orders: *limit order* and *market order*. A limit order is a type of order to buy or sell a token at a specific price (or at a better price than the specific price). In the traditional order-book system, a user is allowed to specify any price. In Lien FairSwap, however, the only price the user is allowed to specify is the latest benchmark price, $X/Y$ ETH/ABC. If the clearing price deviates so much from $X/Y$ ETH/ABC as to render a trade disadvantageous, the limit order will be canceled.[4] On the other hand, a market order is executed at a current market price, as long as the price does not change dramatically.

Now, let $t$ and $T$ be the *tolerance parameters* for limit and market orders, respectively. Suppose also that $P_0 = X/Y$ is the benchmark price and $P$ the clearing price. As buyers try to avoid a higher price, a buy limit order is guaranteed to be canceled if $P > (1 + t)P_0$ and a buy market order is guaranteed to be canceled if $P > (1 + T)P_0$. Likewise, because sellers do not like selling at a lower price, a sell limit order is guaranteed to be canceled if $P < (1 + t)^{-1}P_0$ while a sell market order is guaranteed to be canceled if $P < (1 + T)^{-1}P_0$. Note that, as the design of Lien FairSwap is robust against front-running even if we do not

[4]Such orders are not carried over to the next exchange box.

limit price slippages, we do not have to choose small values for $t$ and $T$ in order to minimize front-running. Accordingly, we can select relatively large values for $t$ and $T$, and therefore, Lien FairSwap has the advantage over Uniswap in being able to process transactions much faster, etc.

The actual process of price determination is more involved because order cancellation itself influences the price by changing the values of $\Delta X$ and $\Delta Y$. To obtain an efficient outcome, we must establish a reasonable way to ration orders in order to maximize the trade volume while respecting the users' tolerances for price slippage.

Orders will be rationed in the following manner. First, let $\Delta X_L$, $\Delta X_M$, $\Delta Y_L$, and $\Delta Y_M$ be the total volumes of buy limit orders, buy market orders, sell limit orders, and sell market orders, respectively (net of commissions paid to the liquidity provider). We then sum up the volumes of these four different orders separately.

In the beginning, Lien FairSwap checks whether all orders can be accommodated into the current exchange box, with the initial price, $P^*$, set as follows:

$$P^* = \frac{X + \Delta X_L + \Delta X_M}{Y + \Delta Y_L + \Delta Y_M}.$$

If all users accept $P^*$, i.e. if $(1+t)^{-1}P_0 < P^* < (1+t)P_0$ is satisfied, then the exchange box is cleared at $P^*$.

However, when the order volumes are unbalanced, $P^*$ may deviate significantly from $P_0$, the benchmark price. In such a case, Lien FairSwap will adjust the price by rationing the long-side orders. For example, let us assume that there are too many buy orders, and therefore, $P^*$ becomes larger than $(1+t)P_0$. When this occurs, buyers who made limit orders do not accept $P^*$ and we must therefore cancel (some of) their orders. Here, after canceling some limit orders, the price goes down and may become equal to $(1+t)P_0$.

Once the marketplace finds that $P^* > (1+t)P_0$, it computes the clearing price ($P^-$) that

would be established if all the buy limit orders were canceled as follows:

$$P^- = \frac{X + \Delta X_M}{Y + \Delta Y_L + \Delta Y_M}.$$

If $P^- \leq (1+t)P_0$, we can adjust the clearing price to $(1+t)P_0$ by cancelling some buy limit orders. We find the volume of those buy limit orders to be processed, $\Delta X'_L \in [0, \Delta X_L]$ , by solving the following equation for $\Delta X'_L$:

$$(1+t)P_0 = \frac{X + \Delta X'_L + \Delta X_M}{Y + \Delta Y_L + \Delta Y_M},$$

which gives us the following value:

$$\Delta X'_L = (1+t)P_0(Y + \Delta Y_L + \Delta Y_M) - X - \Delta X_M.$$

In this case, the clearing price will become $(1+t)P_0$, and all the buy market orders, sell limit orders, and sell market orders will be cleared. Note that only $\Delta X'_L/\Delta X_L$ percent of the buy limit orders are processed, and the remaining buy limit orders are canceled. All buyers who submitted limit orders will incur the effect of cancellation equally — if the buyer $i$ sent a buy limit order whose volume is $x_i$ ETH, then $x_i \cdot (1 - \Delta X'_L/\Delta X_L)$ ETH is returned to the buyer $i$ due to cancellation. The rest of the order is processed normally and the buyer $i$ receives $(1-\gamma) \cdot (\Delta X'_L/\Delta X_L) \cdot x_i/[(1+t)P_0]$ ABC at the price of $(1+t)P_0$.

If $P^-$ is higher than $(1+t)P_0$, all the buy limit orders should be canceled. All of the remaining orders either (i) benefit from a higher price (for all the sell orders) or (ii) adapt to a higher price (for buy market orders). Hence, the exchange box is usually cleared at the price of $P^-$, after cancelling all buy limit orders.

However, as a precaution, we also cancel some market orders when the price change is too abrupt. Specifically, if $P^-$ is larger than $(1+T)P_0$, we cancel market orders until the price reaches $(1+T)P_0$. The volume of buy market orders to be processed, $\Delta X'_M \in [0, \Delta X_M]$, can

be derived from the following equation:

$$(1+T)P_0 = \frac{X + \Delta X'_M}{Y + \Delta Y_L + \Delta Y_M},$$

or equivalently:

$$\Delta X'_M = (1+T)P_0(Y + \Delta Y_L + \Delta Y_M) - X.$$

In this case, all the sellers trade at the price of $(1+T)P_0$. All the buy limit orders are canceled, and $(1 - \Delta X'_M/\Delta X_M)$ percent of the buy market orders is also canceled. However, $\Delta X'_M/\Delta X_M$ percent of the buy market orders is processed at the price of $(1+T)P_0$. Consequently, if the buyer $i$ places a buy market order of volume $x_i$ ETH, then $x_i(1 - \Delta X'_M/\Delta X_M)$ ETH will be returned due to cancellation, and $(1-\gamma) \cdot (\Delta X'_M/\Delta X_M) \cdot x_i/[(1+T)P_0]$ ABC will also be returned as part of the order that has been processed.

The similar logic applies to the case where $P^* < P_0$ — we ration some sell orders to increase the price. The entire mechanism for price determination will be illustrated with a flowchart in the next subsection.

## 3.6 Summary of the Clearing Procedure

**Input** Assume that the liquidity provider currently has $X$ ETH and $Y$ ABC in the pool. Let $I_L$, $I_M$, $J_L$, and $J_M$ be the set of users who made buy limit orders, buy market orders, sell limit orders, and sell market orders, respectively. For $i \in I_L \cup I_M$, $x_i \in \mathbb{R}_+$ denotes the volume of the buyer $i$'s order. For $j \in J_L \cup J_M$, $y_j \in \mathbb{R}_+$ denotes the volume of the seller $j$'s order.

**Step 1: Aggregation** The marketplace initially calculates the commission rate $\gamma$ following the protocol.[5] Then, the marketplace subtracts commissions for the liquidity provider and

[5]For some trading assets, we can use a fixed value for $\gamma$. However, when LBT is included in a trading pair, we should adjust $\gamma$ dynamically while taking into account the market volatility. See Section 4.

aggregates the orders separately as follows:

$$\Delta X_L \leftarrow (1-\gamma)\sum_{i\in I_L} x_i, \quad \Delta X_M \leftarrow (1-\gamma)\sum_{i\in I_M} x_i,$$
$$\Delta Y_L \leftarrow (1-\gamma)\sum_{i\in J_L} y_j, \quad \Delta Y_M \leftarrow (1-\gamma)\sum_{i\in J_M} y_j.$$

**Step 2: Threshold Prices** The marketplace calculates the following threshold prices:

$$\begin{aligned} P_0 &\leftarrow \frac{X}{Y}, \\ P^+ &\leftarrow \frac{X+\Delta X_L+\Delta X_M}{Y+\Delta Y_M}, \\ P^* &\leftarrow \frac{X+\Delta X_L+\Delta X_M}{Y+\Delta Y_L+\Delta Y_M}, \\ P^- &\leftarrow \frac{X+\Delta X_M}{Y+\Delta Y_L+\Delta Y_M}. \end{aligned}$$

**Step 3: Rationing and Price Adjustment** The marketplace determines the clearing price $P$ and the rationed trade volumes ($\Delta\hat{X}_L$, $\Delta\hat{X}_M$, $\Delta\hat{Y}_L$, and $\Delta\hat{Y}_M$). We will then have the following seven scenarios for rationing and price adjustment:

**Case 1:** $P_0 \geq (1+T)P^+$**.** No buy order is rationed: $\Delta\hat{X}_L \leftarrow \Delta X_L$ and $\Delta\hat{X}_M \leftarrow \Delta X_M$. All the sell limit orders are rationed: $\Delta\hat{Y}_L \leftarrow 0$. The sell market orders are also partially rationed:

$$\Delta\hat{Y}_M \leftarrow \Delta Y'_M = \frac{1+T}{P_0}(X+\Delta X_L+\Delta X_M) - Y.$$

The clearing price is set to the lower limit: $P \leftarrow (1+T)^{-1}P_0$.

**Case 2:** $(1+T)P^+ > P_0 \geq (1+t)P^+$**.** No buy order is rationed: $\Delta\hat{X}_L \leftarrow \Delta X_L$ and $\Delta\hat{X}_M \leftarrow \Delta X_M$. Sell market orders are not rationed either: $\Delta\hat{Y}_M \leftarrow \Delta Y_M$. However, all the sell limit orders are rationed: $\Delta\hat{Y}_L \leftarrow 0$. The clearing price is $P \leftarrow P^+$.

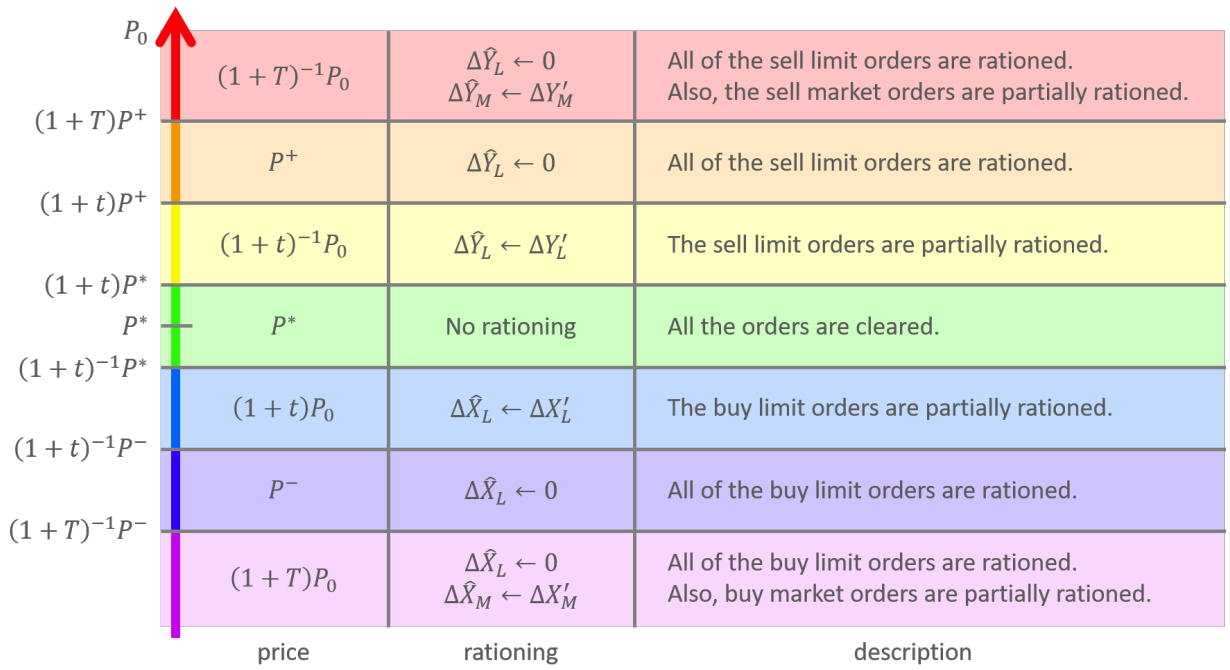



Figure 3: Rationing and price adjustment for different price levels, with $P_0$ used as the benchmark price.

**Case 3:** $(1+t)P^+ > P_0 \geq (1+t)P^*$**.** No buy order is rationed: $\Delta\hat{X}_L \leftarrow \Delta X_L$ and $\Delta\hat{X}_M \leftarrow \Delta X_M$. Sell market orders are not rationed either: $\Delta\hat{Y}_M \leftarrow \Delta Y_M$. The sell limit orders are partially rationed:

$$\Delta\hat{Y}_M \leftarrow \Delta Y'_M = \frac{1+t}{P_0}(X + \Delta X_L + \Delta X_M) - Y - \Delta Y_M.$$

The clearing price is $P \leftarrow (1+t)^{-1}P_0$.

**Case 4:** $(1+t)P^* > P_0 \geq (1+t)^{-1}P^*$**.** No order is rationed at all: $\Delta\hat{X}_L \leftarrow \Delta X_L$, $\Delta\hat{X}_M \leftarrow \Delta X_M$, $\Delta\hat{Y}_L \leftarrow \Delta Y_L$ and $\Delta\hat{Y}_M \leftarrow \Delta Y_M$. The clearing price is $P \leftarrow P^*$.

**Case 5:** $(1+t)^{-1}P^* > P_0 \geq (1+t)^{-1}P^-$**.** No sell order is rationed: $\Delta\hat{Y}_L \leftarrow \Delta Y_L$ and $\Delta\hat{Y}_M \leftarrow \Delta Y_M$. Buy market orders are not rationed either: $\Delta\hat{X}_M \leftarrow \Delta X_M$. The buy limit

orders are partially rationed:

$$\Delta\hat{X}_L \leftarrow \Delta X'_L = (1+t)P_0(Y + \Delta Y_L + \Delta Y_M) - X - \Delta X_M.$$

The clearing price is $P \leftarrow (1+t)P_0$.

**Case 6:** $(1+t)^{-1}P^- > P_0 \geq (1+T)^{-1}P^-$**.** No sell order is rationed: $\Delta\hat{Y}_L \leftarrow \Delta Y_L$ and $\Delta\hat{Y}_M \leftarrow \Delta Y_M$. Buy market orders are not rationed either: $\Delta\hat{X}_M \leftarrow \Delta X_M$. However, all the buy limit orders are rationed: $\Delta\hat{X}_L \leftarrow 0$. The clearing price is $P \leftarrow P^-$.

**Case 7:** $(1+T)^{-1}P^- > P_0$**.** No sell order is rationed: $\Delta\hat{Y}_L \leftarrow \Delta Y_L$ and $\Delta\hat{Y}_M \leftarrow \Delta Y_M$. All the buy limit orders are rationed: $\Delta\hat{X}_L \leftarrow 0$. The buy market orders are also partially rationed:

$$\Delta\hat{X}_M \leftarrow \Delta X'_M = (1+T)P_0(Y + \Delta Y_L + \Delta Y_M) - X.$$

The clearing price is set to the upper limit: $P \leftarrow (1+T)P_0$.

**Step 4: Clearing Individual Orders** Based on the prices and volumes calculated so far, the marketplace determines how much is returned to whom.

**Buy Limit Orders** Each buyer $i \in I_L$, who made a limit order, receives $(1-\Delta\hat{X}_L/\Delta X_L)\cdot x_i$ ETH (returned due to cancellation) and $(1-\gamma)\cdot(\Delta\hat{X}_L/\Delta X_L)\cdot x_i/P$ ABC.

**Buy Market Orders** Each buyer $i \in I_M$, who made a market order, receives $(1-\Delta\hat{X}_M/\Delta X_M)\cdot x_i$ ETH (returned due to cancellation) and $(1-\gamma)\cdot(\Delta\hat{X}_M/\Delta X_M)\cdot x_i/P$ ABC.

**Sell Limit Orders** Each seller $j \in J_L$, who made a limit order, receives $(1-\gamma)\cdot(\Delta\hat{Y}_L/\Delta Y_L)\cdot y_j \cdot P$ ETH and $(1-\Delta\hat{Y}_L/\Delta Y_L)\cdot y_j$ ABC (returned due to cancellation).

**Sell Market Orders** Each seller $j \in J_M$, who made a market order, receives $(1-\gamma) \cdot (\Delta\hat{Y}_M/\Delta Y_M) \cdot y_j \cdot P$ ETH and $(1-\Delta\hat{Y}_M/\Delta Y_M) \cdot y_j$ ABC (returned due to cancellation).

**Step 5: Updating the pool volume** After all the trades are executed, the liquidity provider will be holding the following amounts of ETH and ABC in the pool:

$$X + (1+\gamma)\left(\Delta\hat{X}_L + \Delta\hat{X}_M\right) - P \cdot \left(\Delta\hat{Y}_L + \Delta\hat{Y}_M\right) \quad \text{ETH},$$

$$Y + (1+\gamma)\left(\Delta\hat{Y}_L + \Delta\hat{Y}_M\right) - P^{-1} \cdot \left(\Delta\hat{X}_L + \Delta\hat{X}_M\right) \quad \text{ABC}.$$

The marketplace sets those values as the values of $X$ and $Y$ for the next round, respectively, and completes the clearance of the current exchange box.

### 3.7 Closing on the Maturity Date

Being a call option, an LBT has a maturity date. Hence, the marketplace should close on the maturity date.[6] On the maturity date, the marketplace is closed automatically, and no trade can be conducted after the market closure. The protocol automatically returns the ETH and ABC tokens in the pool to the liquidity provider and completes its role.

## 4 Concerns Specific to iDOL/LBT Tradings

### 4.1 Time Decay

One could argue that the Constant Product Market Maker model is not suitable for an asset that has a maturity date and whose value systematically changes over time. Indeed, the value of the call option (i.e. LBT) decreases as its maturity date nears. Hence, the users of Lien FairSwap can extract profits from the liquidity provider.

[6] This is the most fundamental reason why we need a new, customized DEX platform specifically designed for LBTs. Most of the existing DEX platforms, including Uniswap, cannot support assets with maturity dates.

We do not consider this problem to be significant. In the Constant Product Market Maker model, the liquidity provider supports the trading pairs for assets that do not have a maturity date in addition to those assets that do have a maturity date; we can expect the liquidity provider to remain profitable as long as the commission rate for each of the supported assets is set to an appropriate level.

## 4.2 Commission Rate Adjustment

The value of an option contract offered by a liquidity provider to its users is influenced by the price volatility of the underlying asset. If its volatility is high, there is an increasing chance of mispricing by the liquidity provider, creating an opportunity for large arbitrage gains. As a result, the liquidity provider can be exposed to huge losses when there is excessive volatility in the underlying asset.

The LBT trading on Lien FairSwap could be susceptible to this problem. Suppose that a liquidity provider is opening a marketplace for trading LBT and iDOL (whose value is pegged to USD). Here, the value of the call option (LBT) can change in a non-linear fashion against the value of the underlying asset (Ether). As such, even when the price volatility of ETH (the volatility of the ETH/USD exchange rate) stays fairly constant, the price of LBT can fluctuate more than the price of the underlying asset (ETH). This means that LBT may see huge volatility when the ETH price is in a certain range, and this can be detrimental to the liquidity provider.

To ensure the profit for the liquidity provider, we dynamically adjust the commission for every exchange box, using the price volatility of LBT as a parameter upon which to determine the values of option contracts provided by the liquidity provider.

## 4.3 The Option Value Provided by the Liquidity Provider

We first demonstrate how the value of an option contract provided by the liquidity provider is designed to change (approximately) linearly. Although this analysis also applies to other

assets in general, this is particularly important for trading pairs involving LBT given its volatile nature.

For every exchange box, the liquidity provider provides a call option and a put option to the users, allowing them to have the right to buy and sell an LBT with iDOL once the next exchange box is closed, typically several minutes later. While the precise "strike price" of an option depends on the number of orders involved, the users can generally trade at a price near the benchmark price, $P_0 = X/Y$ USD/LBT. Note that, although the "base asset" for this marketplace is iDOL (rather than USD), the exchange rate will be maintained at a constant value against USD as iDOL is a stable coin pegged to USD.

For simplicity, assume that the pricing of the previous exchange box was perfect in the sense that $P_0$ coincides with the global exchange rate; this assumption would be (approximately) true as long as arbitragers are actively trading on Lien FairSwap. In this case, the options supplied by the liquidity provider are considered to be "at-the-money options" — both the current price and strike price are $P_0$.

Now, let $V$ be the value of the right to buy an LBT at the price of $P_0$ in the next exchange box. It is well-known that the value of an at-the-money option can be evaluated with a simple formula (see Brenner and Subrahmanyan 1988). Using this simple formula coupled with the Black-Scholes formula (Black and Scholes 1973), the value of our at-the-money option can be calculated as follows:

$$V = P_0 \left\{ N\left(-r\sqrt{t} + \frac{1}{2}\sigma_C\sqrt{t}\right) - e^{-rt} N\left(-r\sqrt{t} - \frac{1}{2}\sigma_C\sqrt{t}\right) \right\}, \tag{2}$$

where $t$ is the "time of expiration" (i.e. the time of closing the next exchange box), $r$ is the risk-free interest rate, $\sigma_C$ is the volatility of the USD/LBT exchange rate, and $N(\cdot)$ is the cumulative distribution of the standard normal distribution.

To proceed, we first conduct the following approximations:

$$N\left(-r\sqrt{t}+\frac{1}{2}\sigma_C\sqrt{t}\right)\approx N\left(\frac{1}{2}\sigma_C\sqrt{t}\right),$$
$$e^{-rt}N\left(-r\sqrt{t}-\frac{1}{2}\sigma_C\sqrt{t}\right)\approx N\left(-\frac{1}{2}\sigma_C\sqrt{t}\right).$$

The way we derive these approximate values is as follows. First, we can see that the risk-free rate is negligibly small compared with the price volatility of a cryptoasset. Hence, $\sigma_C/2$ is much larger than $r$, allowing us to ignore $r$ here. Second, since the expiration time $t$ is extremely short, we can say that $e^{-rt}\approx 1$.

Given these approximations, (2) simplifies to:

$$V\approx P_0\left\{N\left(\frac{1}{2}\sigma_C\sqrt{t}\right)-N\left(-\frac{1}{2}\sigma_C\sqrt{t}\right)\right\}$$

Since $t$ is small, $1/2\cdot\sigma_C\sqrt{t}$ and $-1/2\cdot\sigma_C\sqrt{t}$ are close to zero. Then, using Taylor's formula, we obtain:

$$V\approx P_0N'(0)\sigma_C\sqrt{t}, \tag{3}$$

where $N'$ is the density function of the standard normal distribution.

The formula (3) implies that the value of a call option supplied by the liquidity provider is proportional to the volatility of the USD/LBT exchange rate, $\sigma_C$.

## 4.4 How to Calculate LBT's Price Volatility

In this subsection, we explain how we calculate the volatility of the USD/LBT exchange rate. Since there is currently no price oracle for the USD/LBT exchange rate, we should compute it from the historical data for the USD/ETH exchange rate.

Let $S(t)$ be the USD/ETH exchange rate in time $t$. We assume that $S(t)$ follows

$$dS(t)=\mu S(t)dt+\sigma S(t)dW(t),$$

where $\mu$ is the drift rate of the USD/ETH exchange rate, $\sigma$ is the volatility of the USD/ETH exchange rate, and $W(t)$ is a Wiener process. Now, let $C(t, S(t))$ be the value of LBT at time $t$. The LBT endows the right to buy 1 ETH at the price of $K$ USD at time $T$. Then, $C(t, S(t))$ follows:

$$\begin{aligned} dC(t,S(t)) &= \frac{\partial C}{\partial t}(t,S(t))dt + \frac{\partial C}{\partial P}(t,S(t))dS(t) + \frac{1}{2}\frac{\partial^2 C}{\partial P^2}(t,S(t))(dS(t))^2 \\ &= \left(\frac{\partial C}{\partial t}(t,S(t)) + \frac{\partial C}{\partial P}(t,S(t))\mu S(t) + \frac{1}{2}\frac{\partial^2 C}{\partial P^2}(t,S(t))\sigma^2 S(t)^2\right)dt \\ &\quad + \frac{\partial C}{\partial P}(t,S(t))\frac{S(t)}{C(t,S(t))}\sigma \cdot C(t,S(t))dW(t). \end{aligned}$$

Accordingly, the volatility of LBT is given by:

$$\sigma_C = \frac{\partial C}{\partial P}(t,S(t))\frac{S(t)}{C(t,S(t))}\sigma.$$

Since LBT is a call option, its price, $C(t, S(t))$, is given by the Black-Scholes formula. It is well-known that its partial derivative for the base asset price (also known as the *delta for a call option*) is given by:

$$\frac{\partial C}{\partial P}(t,S(t)) = N\left(\frac{1}{\sigma\sqrt{T-t}}\left[\log\left(\frac{S(t)}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]\right).$$

Lien FairSwap is aware of the maturity date ($T$), strike price ($K$), and the current LBT price ($C(t, S(t))$) by design. The risk-free interest rate ($r$), the current USD/ETH exchange rate ($S(t)$), and its volatility ($\sigma$) can be computed from the data provided by a reliable oracle such as ChainLink. Using those data points, we calculate the volatility of LBT, $\sigma_C$, and adjust the commission rate linearly based on the changes in $\sigma_C$.

# References


Angeris, G., H.-T. Kao, R. Chiang, C. Noyes, and T. Chitra (2019): "An analysis of Uniswap markets," *arXiv preprint arXiv:1911.03380*.

Black, F. and M. Scholes (1973): "The pricing of options and corporate liabilities," *Journal of political economy*, 81, 637–654.

Brenner, M. and M. G. Subrahmanyan (1988): "A simple formula to compute the implied standard deviation," *Financial Analysts Journal*, 44, 80–83.

Budish, E., P. Cramton, and J. Shim (2015): "The high-frequency trading arms race: Frequent batch auctions as a market design response," *The Quarterly Journal of Economics*, 130, 1547–1621.

Daian, P., S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels (2019): "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," *arXiv preprint arXiv:1904.05234*.

Lewis, M. (2014): *Flash Boys: A Wall Street Revolt*, W. W. Norton & Company.

Lien Protocol (2020): "iDOL white paper," https://lien.finance/pdf/iDOLWP_v1.pdf.

Lu, A. (2017): "Building a decentralized exchange in Ethereum," https://blog.gnosis.pm/building-a-decentralized-exchange-in-ethereum-eea4e7452d6e, Accessed on April 16, 2020.