

Article

Oracles in Decentralized Finance: Attack Costs, Profits and Mitigation Measures

Ayana T. Aspembitova * and Michael A. Bentley

Euler Labs, London EC1V 2NX, UK

* Correspondence: ayana.aspembitova@euler.xyz

Abstract: Decentralized finance (DeFi) is by far the most popular application of blockchain technology. Despite the wide acceptance of new financial instruments and services, there are still many unexplored areas in the field. We dedicate this research to the understanding of one of the most crucial limitations of decentralized finance—oracles. DeFi protocols, as well as other blockchain applications, function in a closed environment and regularly need to fetch real-world information (e.g., assets' prices)—the tool used for this purpose is called an oracle. We review the existing oracle types in DeFi applications and focus our research on the least explored one: when another protocol, typically a decentralized exchange, serves as a price oracle. After explaining the mechanisms behind the decentralized exchanges, we introduce an algorithmic model that allows one to safely design a decentralized oracle and adjust crucial parameters. We believe that understanding and implementing the logic presented in the model can help to reduce the chances of price manipulations attacks, which are the most frequent incident types in DeFi.

Keywords: DeFi; oracle; automated market makers; decentralized exchange; lending protocol



Citation: Aspembitova, A.T.; Bentley, M.A. Oracles in Decentralized Finance: Attack Costs, Profits and Mitigation Measures. *Entropy* **2023**, *25*, 60. <https://doi.org/10.3390/e25010060>

Academic Editors: Stanisław Drożdż, Jarosław Kwapień and Marcin Wątorek

Received: 6 November 2022

Revised: 13 December 2022

Accepted: 22 December 2022

Published: 28 December 2022



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Blockchain-based smart contracts have been successfully growing, and their use cases are quite innovative and have attracted lots of interest valued in the billions of dollars. However, there is a fundamental limitation of decentralized applications—they execute in a closed environment and a bridge service (oracle) is needed when obtaining information outside of the blockchain. As decentralized applications evolve and mature, oracles play an increasingly prominent role in ensuring the safety across smart contracts. Despite the critical role that oracles play in decentralized applications, the research is still in its infancy. In [1], the authors performed a bibliometric analysis and demonstrated the alarming scarcity of the research dedicated to blockchain oracles. Moreover, in the recent study of DeFi incidents [2], the authors empirically showed that oracle manipulation attacks are the most frequent incident types in DeFi. Although there are tools that can detect the price manipulation attacks [3,4], and identify new vulnerabilities in real time, there is still a need for prevention measures. The lack of understanding of oracles mechanics and functions concerns not only academic research but more so the real users of decentralized applications.

Decentralized finance (DeFi) uses blockchain technology to provide financial instruments without intermediaries in a trustless and transparent manner [5]. DeFi covers a wide range of financial products, offering innovative alternatives to traditional financial products, such as stablecoins, exchanges, lending protocols, insurance and yield farming protocols.

Here, we provide an overview on why DeFi rests heavily on the use of oracles and how information from the outside world can be retrieved. Generally speaking, there is some *ground truth* information that resides outside of smart contracts, and smart contracts need it for the proper performance. To obtain such ground truth, smart contracts need reliable *data sources*—any entity that stores the ground truth information (databases, sensors or other smart contracts). Then, *data feeders* report off-chain data to an on-chain system.

The systematic explanation on the existing type of oracles in a blockchain is provided in [6]. As for the decentralized financial applications, the ground truth needed is the price of the assets listed in a smart contract. Although there are many types of oracles with different functions and characteristics, the oracles currently used in DeFi can be broadly divided into two main categories—decentralized trust-based oracles and decentralized exchanges used as oracles.

Decentralized trust-based oracles function as a smart contract and do not rely on a single source of information. Instead, they query multiple sources and aggregate the obtained information into a single output. The papers [7–9] provided a detailed review on the architecture, workflow and weak points of various decentralized oracles, such as Chainlink [10,11], Provable [12], Oraclize, etc. Some DeFi applications are fetching the price information directly from the decentralized exchanges by either getting the spot price or aggregating the prices over a certain window size. Using the spot price can be very dangerous because the price can be easily manipulated [13–15]. Therefore, more and more DeFi applications started using the TWAP (time-weighted average price) instead—the output price is calculated as a weighted average over a certain time period and, therefore, the cost of price manipulation of the TWAP oracle increases linearly with the length of the TWAP averaging window, reducing the chance of an oracle hack.

In this paper, we focus on the decentralized exchanges (DEXs) used as oracles for DeFi protocols. While trust-based oracles have attracted some attention from the researchers, using DEXs directly as oracles is still not well understood. In [16], the authors analyzed the cost of TWAP manipulation when an arithmetic mean is used for the aggregation and also considered the possibility of an MMEV attack. Decentralized exchanges utilize the concept of automated market makers (to be explained in detail in Section 2). Our main contributions consist of the following: we systematize the existing knowledge about using automated market maker (AMM)-based decentralized exchanges as oracles, we derive attack costs for the most popular cost functions used in DEXs, then we derive the relations between protocol-specific parameters and oracle-specific parameters that impact the safety of using the DEX-based oracle and, finally, we develop the algorithmic model that allows to assess the risks of using oracles in a given protocol. Overall, knowing the mechanics behind the oracles' work would give a comprehensive understanding on how attacks can be performed. Implementing the logic presented in the model below would give the quantitative estimate on the cost a potential attacker needs for a successful attack. Knowing the mechanism behind the price oracle and being able to precisely estimate the cost of a potential attack provides an additional layer of security to the protocols using DEX-based oracles.

The paper is structured as follows. First, we review the most popular AMM-based decentralized exchanges, demonstrate their logic and the cost functions used for asset pricing. In the appendices, we derive the cost of the attacks for each type of AMM pricing function discussed in Section 2. Then, in Section 3, we discuss various aggregation methods that can be used in DEX-based oracles and show how they can be impacted by the price manipulation attack. Section 4 aggregates all the information obtained above and provides a step-by-step algorithm on how to mitigate attacks related to the DEX-based oracles on the example of a lending protocol. We simulate various attack scenarios to the lending protocol on two types of AMM cost functions—a constant product and stableswap. Finally, we conclude all the findings and discuss the future directions of this research in the last section.

2. Automated Market Makers

To understand the safety of using the DEX-based oracle, we need to first be familiar with how DEXs work and understand the mathematics behind it—the cost function utilized by the DEX. In this section, we first explain the mechanism of the decentralized exchange protocol and then review the popular AMM cost functions and demonstrate how they are used to price assets in a DEX.

An AMM-based decentralized exchange consists of pools of different assets (liquidity pools). Liquidity to these pools is provided by people who wish to gain income from the

transaction fees (liquidity providers). Each pool can have few assets (currently most of the pools have two assets) and users who want to exchange assets (traders) interact directly with the given pool to swap asset x to asset y .

In centralized exchanges, the price discovery happens by matching the sell and buy orders from various counterparties. In contrast to it, decentralized exchanges are based on the automated market-making mechanism (AMM). The AMM utilizes the cost function that discovers the price algorithmically—this function only allows counterparties to exchange the assets for the prices along the trajectory determined by the AMM formula and quantities of the available assets. Although the implementation of AMM functions to price assets in decentralized exchanges is quite novel, the idea of agents automatically placing bets and following prescribed rules is not new and has been implemented in many areas to aggregate the information—the prediction of building openings [17], sport matches [18], etc. Overall, the idea of automated market making is to define algorithmic rules for agents within the system to place their bets on a certain subject, aggregate them and derive a single function (*conservation function*) from the outcome. In DEXs, this process goes a little different. First, the conservation function is defined and then agents (traders and liquidity providers) match their trades, and whenever there is a trade that goes beyond the expectations of the cost function, it is punished by the algorithm and, therefore, it discourages agents to behave (trade) differently than prescribed by the conservation function. Although in DEXs agents are not algorithmic bots but real people, they do act in a way as was expected algorithmic bots to act to preserve the cost function.

In [19], Othman introduced five desideratas (desirable properties) for cost functions—monotonicity, convexity, bounded loss, translation invariance and positive homogeneity. As it was proven by [20], it is impossible for the cost function to satisfy all five properties. Therefore, all the cost functions utilized by AMMs satisfy only a few properties, while others are relaxed.

2.1. Logarithmic Market Scoring Rule

The first automated market maker for prediction markets was introduced by Hanson [21,22]. It has been quite popular due to its simple analytical form and satisfying the main desirable properties for cost functions (convexity, bounded loss and translation invariance).

The Logarithmic Market Scoring Rule (LMSR) conservation function for n assets is defined as:

$$C(x) = b \log\left(\sum_{i=1}^n \exp(x_i/b)\right) \quad (1)$$

where $b > 0$ is the liquidity parameter, it is strictly positive, constant and it is defined before the pricing of assets. b parameter controls the liquidity in the market—the higher the b , the less the price is shifted when assets are added. Moreover, it translates into the bigger maximum loss because the market maker's worst-case loss is the function of b which is $b \log n$.

The derivative of the cost function $C(x)$ is the price function in the LMSR:

$$p_i(x) = \frac{\exp(\frac{x_i}{b})}{\sum_j \exp(\frac{x_j}{b})} \quad (2)$$

The LMSR is used in many settings, such as auctions, prediction markets, rating markets, etc. In decentralized finance, the LMSR has not been widely used for a few reasons: first, the LMSR does not satisfy the liquidity sensitivity property; second, it is quite easy and cheap to compromise the price of an asset when the LMSR is used as a cost function.

By allowing the parameter b to be the function of the outstanding quantities instead of being constant, the LMSR becomes liquidity sensitive—the Liquidity-Sensitive Logarithmic Market Scoring Rule (LS-LMSR), introduced by Othman [23]:

$$C(x) = b(x) \log\left(\sum_{i=1}^n \exp(x_i/b(x))\right) \quad (3)$$

where function b is as follows:

$$b(x) = \alpha \sum_i x_i \quad (4)$$

where α is the parameter that is strictly positive and set before the pricing of assets. The possible maximum commission (also called vigorish v) depends on the α parameter and does not exceed v when α is set as follows [24]:

$$\alpha = \frac{v}{n \log n} \quad (5)$$

where n is the number of outcomes (assets in the pool for AMM-based DEX). Depending on what is the desired maximum commission v , the optimal parameter α can be easily calculated.

In decentralized finance, the LS-LMSR is used in applications such as Augur [25] and Gnosis [26].

2.2. Constant Product Market

Constant product AMM (CPAMM) cost function for n assets is defined as follows:

$$C(x) = \prod_{i=1}^n x_i \quad (6)$$

where $C(x)$ set as a constant.

Constant product AMM has many advantages that makes it suitable to be used in DEXs—it is simple to code into the smart contract, it is a convex function which meets the principles of supply and demand and it is also liquidity sensitive. Although it has been shown in [27] that prices in such a DEX can be inaccurate during volatile markets, this cost function still remains the most popular and being utilized by large DEXs, such as Uniswap [28,29].

Decentralized exchange pools consist of two tokens and Equation (6) becomes the following:

$$x \times y = k \quad (7)$$

where k is the constant, x is the amount of the first token and y is the amount of the second token.

The price for each token in a pool can be calculated by simply dividing the number of tokens in one reserve to the number of tokens in another. A more detailed review of constant product markets is given in [30,31].

2.3. Combination of Constant Sum and Constant Product Markets

In DeFi, there are assets that have the same value, for example, a different version of USD (USDC, USDT, etc.). Because the ratio between asset x and asset y in such pools is stable and close to 1, they are called *stableswap* pools. The pricing formula for stableswap pools was developed by the Curve team [32]. Essentially, this is a combination of the constant product market pricing formula $xy = k$ and the linear invariant $x + y = C$. The rationale behind adding the linear invariant term to the constant product formula is to achieve the closer peg 1:1 and allow lower slippage for stableswap pools. When using only a linear invariant formula, tokens are always traded at 1:1 with zero slippage; however, this might lead to the depleting of the pool's one token. Using only a constant product formula leads to larger slippage and a less stable peg. Therefore, the combination of these two curves allows to keep the pool balanced while providing a more stable peg.

The final stableswap curve formula looks as in Equation (8). For the full explanation and derivation, refer to the paper [33].

$$2^2 A(x + y) + D = 2^2 AD + \frac{D^3}{2^2 xy} \quad (8)$$

where A is the amplification factor for the linear invariant curve—the larger the A , the closer the curve to the linear. D is the total amount of tokens in the pool.

To calculate the price for the token, one needs to express the curve for y from Equation (8), and the derivative of that expression stands for the price. Stableswap AMM is widely used in many DEX pools that have the same price for both tokens.

3. Aggregation Methods

In this paper, we focus on oracles for DeFi applications that get price information directly from the decentralized exchanges by aggregating the output prices over a certain time period. Every time there is a new swap (trade) in the DEX, the price is updated in oracle and then the time-weighted average is calculated. To mitigate the possible effect of the price manipulation within one or a few blocks, one would prefer to use the time-weighted and/or liquidity-weighted average price.

In this section, we discuss various aggregation methods and show the impact of a price manipulation attack on each of them.

3.1. Arithmetic Mean Time-Weighted Average Price

The arithmetic mean TWAP over n price updates is calculated as follows:

$$TWAP = \frac{\sum_{i=1}^n t_i p_i}{\sum_{i=1}^n t_i} \quad (9)$$

where t_i is the time elapsed between the price update i and next price update $i + 1$, and p_i is the price during that period. n is the averaging window.

We estimate the effect of manipulation on the TWAP price when the attacker wants to consistently manipulate the spot price for m times of price updates within the averaging window n .

$$TWAP_m = \frac{\sum_{i=1}^{n-m} t_i p_i + \sum_{j=n-m+1}^n t_j p_j}{\sum_{i=1}^n t_i} \quad (10)$$

assuming that attack would happen in the last m blocks. From here, we would like to estimate the p_j —how big should the manipulated price be that the attacker should target in order to achieve the desired effect on $TWAP_m$.

$$\sum_{j=0}^m t_j p_j = TWAP_m \times \sum_{i=1}^n t_i - \sum_{i=1}^{n-m} t_i p_i \quad (11)$$

In the case when the attacker does not want to be exposed to arbitrageurs and wants to manipulate the price within one block $m = 1$, the manipulated price will be as follows:

$$p_j = \frac{TWAP_m \times \sum_{i=1}^n t_i - \sum_{i=1}^{n-1} t_i p_i}{t_j} \quad (12)$$

Although using the TWAP instead of a spot price is safer in terms of avoiding the malicious price manipulations, the output from the averaging might not be accurate.

3.2. Geometric Mean Time-Weighted Average Price

The geometric mean TWAP over n blocks can be calculated as the n th root of the product of the spot price on each block:

$$TWAP = \left(\prod_{i=1}^n p_i \right)^{\frac{1}{n}} \quad (13)$$

If the attacker wants to manipulate the geometric mean TWAP by manipulating the price over m blocks, then the target TWAP will be calculated as follows:

$$TWAP_m = (p^{n-m} \times q^m)^{\frac{1}{n}} \quad (14)$$

An attacker wanting to manipulate the TWAP to some particular oracle price $TWAP_m$ over m blocks will need to know what spot price q they need to move the normal spot price p to in each of those blocks. It can be calculated by rearranging Equation (14):

$$q = \sqrt[m]{\frac{TWAP_m^n}{p^{n-m}}} \quad (15)$$

This equation shows that it is surprisingly difficult to move the geometric mean TWAP from the wider market spot price when manipulated blocks are few in number relative to unmanipulated blocks. That is, the spot price must be moved a significant distance from its wider market price in order to have even a modest impact on the geometric mean TWAP.

3.3. Median Time-Weighted Average Price

Using median time-weighted average prices as oracles has been discussed in [34], although they have not been widely implemented in practice yet. Theoretically, because the median is unaffected by the effect of outliers, it could be a solution to avoid single-block manipulation attacks, especially in pools with small liquidity. For an attacker to influence the oracle's final output price, they would need to control the last m block prices for at least half of the period of the window size.

From the economic point of view, storing price time series over a certain period to calculate the median could be very expensive in terms of the gas cost in the Ethereum blockchain. In alternative blockchains with a different technical design and cheaper gas cost, this could be possible if the pros of using the median TWAP outweigh the cons. In DeFi, median TWAP oracles have been implemented in the Euler Finance protocol as an alternative price source to the geometric mean TWAP [34].

4. Algorithmic Model to Estimate the Safety of TWAP Oracle

We have reviewed AMM cost functions in decentralized exchanges that are often used as price oracles. Moreover, we looked at the most popular methods for price aggregation in oracles. Now, we would like to systematize everything into the algorithmic model that allows to estimate the safety of any DEX-based TWAP oracle. Inputs in the model are parameters of the protocol using TWAP oracle and parameters of AMM that is being used as oracle. Outputs of the model are the attack cost AC and capital C needed to provide to the protocol under attack to be able to profit from it. Overall, the model output only tells how much funds a potential attacker needs to gain the profit from price manipulation. To assess the economic feasibility of such an attack, one would need to perform additional independent analysis, for instance, if model output tells that attack would cost 100 USD, then it is economically feasible for many people and, therefore, it is not safe. If model estimates AC and C to be big so only a few people can technically perform an attack, then one can conclude that chances of oracle attack are low. Outputs from the model can serve as a starting point to decide on the safety of TWAP-based oracle.

We first introduce the general algorithmic model that allows to estimate the feasibility of price manipulation attack. Then, we explain how the model can be implemented based on the example of lending protocol—this is the most popular use case of TWAP oracles. We provide a brief explanation of how lending protocols work, demonstrate how model can be used and simulate various attack scenarios for constant product and stableswap AMM-based TWAP oracles.

4.1. Algorithmic Model

Algorithm 1 shows step by step how to estimate the safety of TWAP oracle. Model outputs are attack cost AC and minimum collateral C an attacker needs to profit from their manipulation. Knowing these parameters allows us to estimate the economic feasibility of such an attack and, therefore, to decide whether it is safe to implement the given DEX as an oracle. Input parameters needed for the model can be divided as protocol specific, oracle specific (averaging window size WS) and DEX specific (liquidity L). These parameters can be easily found in protocol's web-page and in DEXs page (L). To find values of Δx (number of tokens needed to move the price to the target value) and Δy (number of tokens to receive in return after the swap), one needs to know the type of AMM that DEX uses to price assets. In Appendices A and B, we have derived equation for Δx and Δy for constant product and stableswap AMMs. Overall, all these parameters can be precisely found and no assumptions need to be made. There is one more parameter that is crucial to take into account—the number of blocks m during which attacker will try to manipulate the price. This can not be known upfront but its value affects the cost of an attack significantly. One would suppose that the value of m should be small because any deviation from the spot price would be noticed by arbitrageurs and set back to the real value, not allowing an attacker to manipulate for many blocks. However, in pools with infrequent trading activity, the arbitrage opportunity might go unnoticed for longer times. Moreover, there is a chance of multi-block MEV-style attack, where an attacker could cooperate with the miner to mine a few blocks in a row. This style of attack combined with the oracle price manipulation makes the oracle attack cost cheaper. In the examples and simulations below, we assume no MMEV attack and frequent trading activity in a pool.

Algorithm 1: Model to estimate the safety of DEX-based oracle

Input: Protocol risk parameters, $WS, m, \Delta x, \Delta y, L$

Output: AC, C

- 1 Calculate minimum deviation from the spot price needed to profit based on protocol-specific parameters.
- 2 Calculate target manipulation price from oracle $TWAP_m$.

$$TWAP_m = TWAP_s + TWAP_s \times \epsilon$$

- 3 Calculate how big should be the manipulation price p_m to achieve needed $TWAP_m$. For this:
 - (I) Estimate the oracle window size WS .
 - (II) Decide on number of blocks m to manipulate the price.
 - (III) Depending on aggregation method, calculate p_m using Equation (12) or Equation (14).
- 4 Calculate attack cost AC based on AMM-specific parameters (liquidity L) and p_m found above. Find Δx and Δy according to the AMM type, as shown in Appendices A and B.

$$AC = \Delta x - \Delta y$$

- 5 Calculate the minimum capital needed to obtain profit $Profit > 0$ depending on protocol's risk parameters.
 - 6 Estimate the economic feasibility of an attack based on the attack cost AC and collateral C values.
-

4.2. Lending Protocols

Lending protocols (also called money markets, credit protocols or protocols for loanable funds) are a market that matches borrowers and lenders—users—who wish to gain interest on their savings, deposit their funds to the lending protocol and then it allows borrowers to lend available assets paying certain interest rate. Detailed explanation on how lending protocols work was provided in the paper [35]. Overall, lending protocols have attracted a lot of interest and become very popular among DeFi community—Ethereum-based lending protocols such as Aave [36], Compound [37], dYdX [38] and MakerDAO [39]. Credit protocols are one of the most popular use cases for AMM-based DEX data to be used as a price feed. Chainlink type of price databases are not always available for relatively new blockchains. In such cases, DEXs, acting as the only option for nascent chains, are used as substitutes for more robust oracle solutions. Considering the large TVL (total value locked in protocol) associated with the popularity of credit protocols and their growing functionality and complexity, it is vital to understand the safe settings of AMM pools that are used as a price information source.

In lending protocols, any user can anonymously borrow funds, but to be able to do so, they first need to provide some collateral asset. To ensure the safety and the solvency of protocol, the Loan-to-Value (LTV) parameter is used—this parameter shows how much a user can borrow relative to their collateral value (all loans in lending protocols are *overcollateralized*. For example, if user deposited 100 USD worth of collateral C and LTV parameter is 80%, then they can borrow up to 80% worth of the other asset B. More detailed explanations of lending protocols and their risk parameters can be found in [35–37,39].

In practice, lending protocols are the most frequent target for oracle manipulation attacks. An attacker tries to artificially increase their collateral value by compromising the oracle price information to be able to borrow more.

We assume a scenario where the attacker artificially increases the value of their collateral to be able to borrow more than their actual collateral value allows. In this case, attacker's profit can be formulated as follows:

$$\text{Profit} = (C \times \text{LTV} + C \times \text{LTV} \times \epsilon) - C \quad (16)$$

Here, ϵ is the target price manipulation fraction and C is the value of collateral—for convenience and normalization purposes, we consider it not as an absolute value but relative to the pool liquidity. This normalization allows us to generalize findings and give parameter recommendations for any pools regardless the size:

$$C = \frac{\text{Collateral}}{\text{PoolLiquidity}} \quad (17)$$

Because the attacker would need to give up their collateral in order to realize their profit from manipulation, we subtract the actual value of their collateral from the profit. From Equation (16), it is clear that the lower the LTV parameter, the more difficult it is to get the profit from an attack and the higher the ϵ should be. We can derive the value of the target manipulation price from the Equation (16)—we set the Profit = 0 and calculate the ϵ as:

$$\epsilon \geq \frac{1}{\text{LTV}} - 1 \quad (18)$$

Figure 1 shows the minimum manipulation target ϵ an attacker needs to achieve for the attack to be profitable given a certain LTV.

Next, after we know the minimum price target needed to make the attack profitable, we can calculate the total cost of an attack using the equations derived in Appendices A and B.

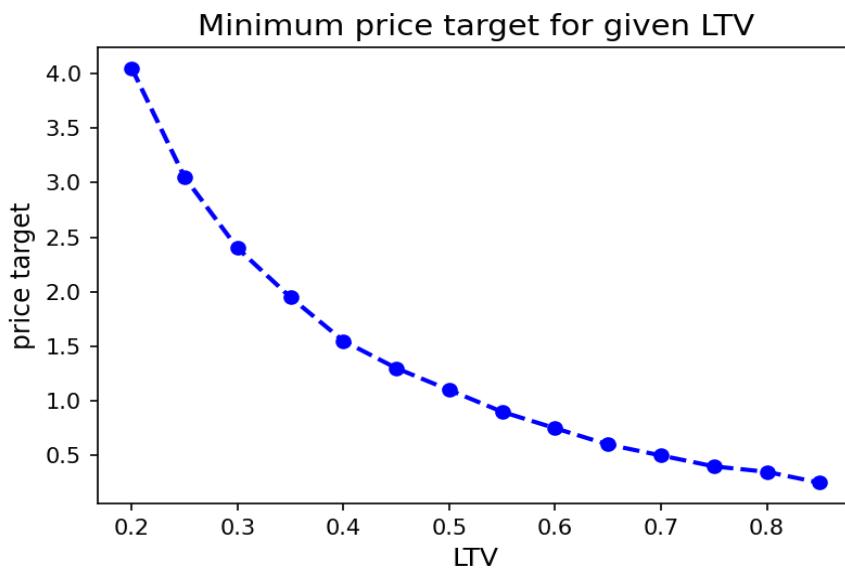


Figure 1. Minimum price target needed for given LTV to obtain the profit from the attack.

4.3. Attack Scenarios to Lending Protocol Using Constant Product AMM-Based TWAP Oracle

Knowing both the profit and attack cost equations, it is straightforward to simulate various attacks to lending protocols that are using any type of AMM as oracle. We looked at the two most popular types of AMM used in decentralized exchanges—constant product and stableswap—to obtain the full understanding about attacker's profits and manipulation capital needed.

In this section, we demonstrate how to calculate the cost of an attack under certain conditions using the terms and explanations shown above. Assumptions used in this example are as follows:

- Loan to Value of the target asset equals 40%.
- TWAP window equals 30 min.
- Time without arbitrage equals 1 min.

Note that we are not making any assumptions regarding pool liquidity given how Equation (16) was defined, which allows us to make calculations, irrespective of the pool liquidity.

Using the attack cost formula and simulating scenarios with varying attacker's collateral C , price target ϵ and pool's liquidity L , we arrive at the following profitability matrix shown in Figure 2, where the space in red indicates a loss (negative profit), the space in blue indicates a positive profit for the attacker and the white area indicates zero-profit scenarios.

From the results shown in Figure 2, we see that the attacker can theoretically reach a profit under almost every combination of events. Moreover, we see that an attack can be profitable by adjusting the requirements to the collateral and the manipulation target ϵ . The lower the ϵ , the higher the collateral the attacker needs to provide for the attack to become profitable and vice versa. These results make us question whether there is such a combination of ϵ and collateral that allows an attacker to obtain profit from an attack with minimum resources?

Figure 3 shows that, effectively, we can retrieve the attack's minimum cost through a specific combination of ϵ and collateral provided; let us call it an *optimal target*. For the example covered in this section, this point happens at collateral being around five times larger than the pool's liquidity and manipulation target ϵ being 4.7. Most importantly, the figure below shows that the total capital needed for a profitable attack is 9.3 times the liquidity in the pool used for the AMM.

The optimal target found above and amount of resources needed to reach that point can serve as a reference when deciding on the safety of an oracle.

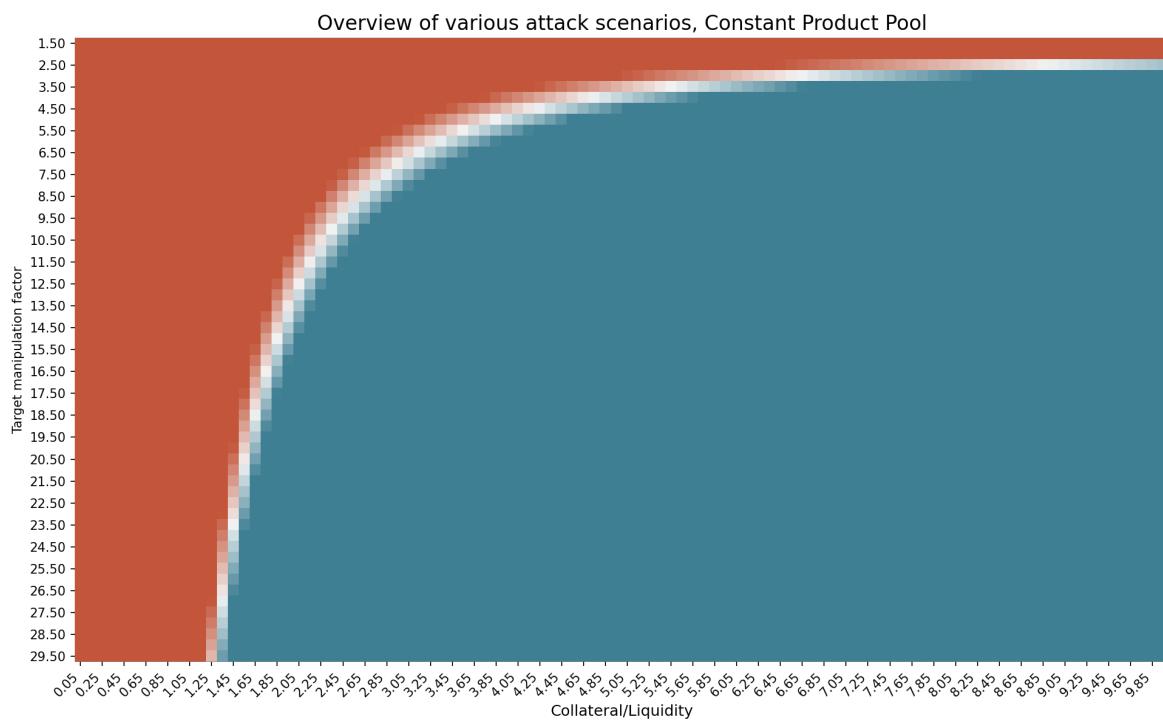


Figure 2. Overview of various attack scenarios in constant product market. The x-axis shows the amount of an attacker’s collateral in terms of liquidity, and the y-axis is the target manipulation price ϵ . The space in red shows the non-profitable attack scenarios (when attack cost exceeds the profit). Blue areas show profitable attacks, while the space in white shows when $\text{Profit} - \text{AttackCost}$ is close to zero. LTV = 0.4 for all scenarios.

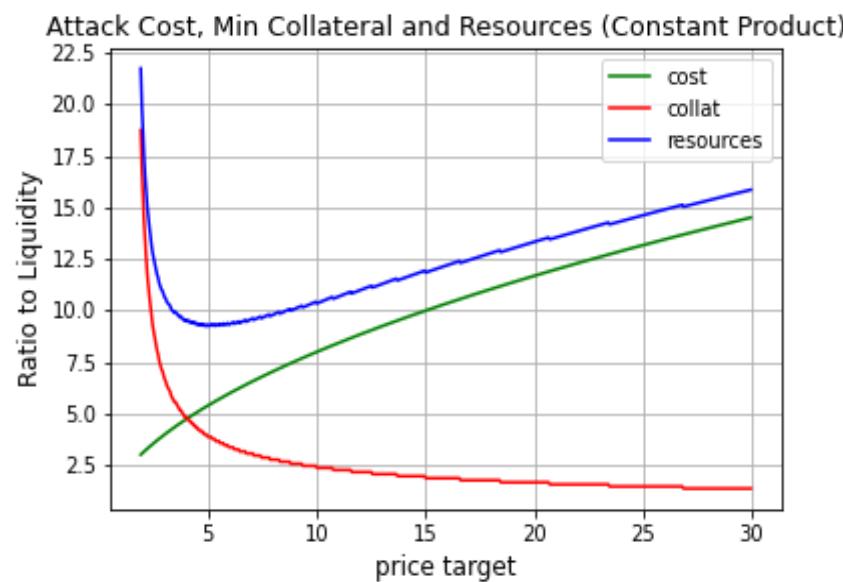


Figure 3. Attack cost, minimum collateral needed and the total resources needed for the profitable attack.

4.4. Attack Scenarios to Lending Protocol Using Stableswap AMM-Based TWAP Oracle

In the previous section, we showed how parameters need to be set for the constant product AMM oracle. In this section, we look at the attack cost and profit when using the stableswap pool as an oracle.

With the attack cost calculated for the stableswap in Appendices A and B, we can run simulations as in the previous section and produce the profitability matrix. Note that the

assumptions used within this section (except for the amplification factor, which is unique to the stableswap AMM) are the same as those used in the previous section. The following figure shows the profitability matrix using a stableswap with an amplification factor A of 30.

From Figure 4, it can be seen that the profitability space for an attack is larger in a stableswap AMM than a CPAMM. In other words, manipulating a stableswap-based TWAP is cheaper than a CPAMM-based TWAP oracle using the same assumptions.

Figure 5 shows the minimum cost of performing a profitable attack, indicating a significantly lower point of minimum cost for an attack in a stableswap AMM than in a constant product AMM. Moreover, another difference is the considerably slower growth rate of the attack cost as the manipulation target increases, which makes the total cost of the attack stagnate as the manipulation target increases. In contrast with constant product AMM, the total cost of an attack keeps increasing. For a stableswap AMM, this results in relatively cheap attack opportunities.

As a final note, a stableswap pool can be relatively stable (in terms of price) at a very unbalanced state (in terms of underlying reserves). At the extreme, we could have a situation where the pool is very close to the “knee” of the pricing curve (where the constant sum (linear) part of pricing curve meets the constant product part), and manipulation attacks become increasingly easier to perform given the aggressive nature of the stableswap curve. In other words, we cannot assume that the attack will take place from a 50:50 state or anything closer to that. The more unbalanced the pool at the start of the manipulation attack, the less resources needed to conduct the attack. Therefore, we do not recommend using stableswap pools as an oracle. Please also refer to Appendix B, where various attack scenarios in stableswap pool were shown under different LTV values—it is clear that this type of pool is much cheaper to manipulate comparing with the constant product market. Moreover, as we can see from Figure 5, once the *optimal price target* is reached, attacker does not need significantly more resources to manipulate price higher and to obtain even higher returns from the attack.

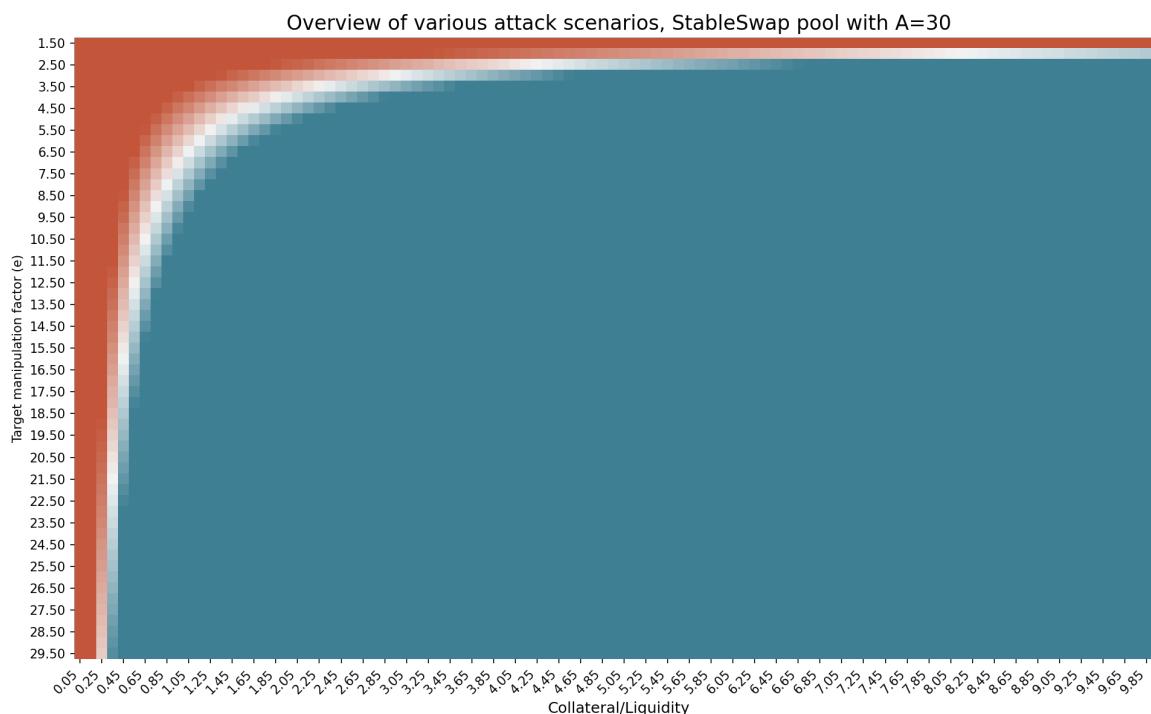


Figure 4. Overview of various attack scenarios in stableswap market. The x-axis shows the amount of attacker's collateral in terms of liquidity, and y-axis is the target manipulation price ϵ . The space in red shows the non-profitable attack scenarios (when attack cost exceeded the profit). Blue areas show profitable attacks, while the space in white shows when $\text{Profit} - \text{AttackCost}$ is close to zero. LTV = 0.4 for all scenarios.

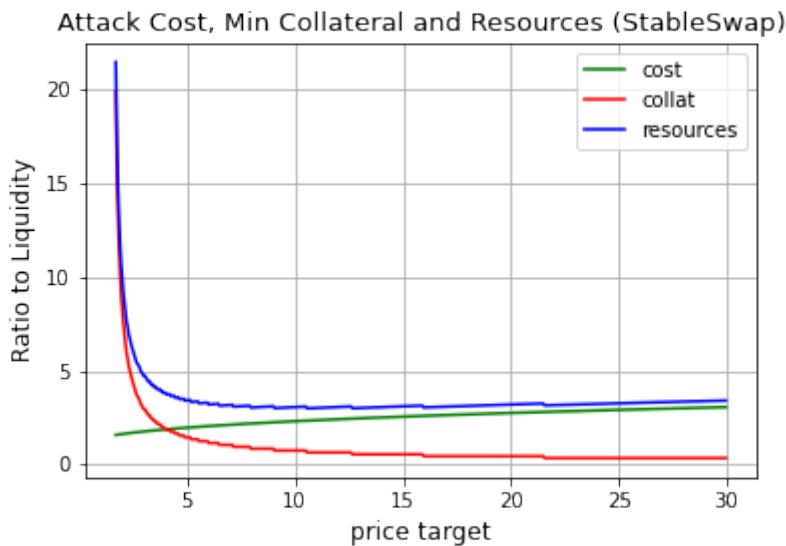


Figure 5. Attack cost, minimum collateral needed and the total resources needed for the profitable attack.

5. Conclusions and Discussion

DeFi protocols, as well as any other blockchain applications, function in a closed environment, and for their proper performance, a reliable data source (oracle) is needed. Currently, there are two different ways to fetch the data about assets' prices—either by using trust-based oracles (e.g., Chainlink) or by getting the prices directly from the decentralized exchange. In our research, we focused on understanding the mechanism of the latter option. Understanding the safety of a DEX-based oracle starts from the deep understanding on how DEXs work; nowadays, they function using the automated market-making (AMM) mechanisms and the asset's price discovery happens along the curve of the AMM cost function. We reviewed the most widely used AMM cost functions and derived the cost of an attack for them. The next step was to look at the various aggregation methods; because using the spot price directly from DEX can lead to cheap price manipulations, most of the DeFi applications aggregate historical spot prices over a certain window size to decrease the chance of an attack. Depending on the method implemented and the window size, the target manipulation price can be higher or lower. We have provided equations to estimate the target attack price based on the aggregation method. We then developed the algorithmic model to estimate the safety of a DEX-based oracle on the example of a lending protocol. A step-by-step algorithm considers protocol-specific, oracle-specific and DEX-specific parameters and provides the logic on how to proceed with deciding on the safety of an oracle. Although we used the lending protocol as an example of a DeFi application using a DEX-based oracle, the model we introduced can be easily generalized to other types of protocols by changing the protocol-specific parameter (*LTV* in our example).

Incidents that happen in the new field of decentralized finance often lead to the crisis of trust from users and have a large social impact on the entire industry. Despite the crucial role oracles play in decentralized finance, their underlying mechanics are still under-explored and poorly understood which resulted in several protocol exploits [13–15]. However, we see the growing interest from both academia and industry practitioners to improve the oracles' resistance to manipulation attacks—new AMM curves are being introduced [40–42], oracle research is growing and more protocols are aware of price manipulation attacks. There is still a lot of work that can be done to achieve the goal of a safe decentralized price oracle in every layer—protocols using oracles can improve their risk management strategies, the AMM cost function can contribute a lot to the safety of oracles, as we saw in Sections 4.3 and 4.4, where the different pricing curves result in the different costs of attack. Finding the optimal AMM cost function that would minimize the chances of manipulation

is not a trivial task, and new pricing curves are being proposed by academia [40,42] and implemented in practice [41]. We hope to see more work performed in this direction. More research can be conducted about information aggregation methods as well—for economic reasons, currently, protocols are using simple statistical methods such as the TWAP. Finding an optimal solution that is less sensitive to the outliers and at the same time has a high price precision and cheap gas cost is still an open question at the moment. Overall, oracles in decentralized finance remain one of the most important and under-researched topics in the field with a huge impact on the entire cryptocurrency system.

Author Contributions: Conceptualization, A.T.A. and M.A.B.; methodology, A.T.A. and M.A.B.; validation, A.T.A. and M.A.B.; formal analysis, A.T.A. and M.A.B.; investigation, A.T.A. and M.A.B.; resources, A.T.A. and M.A.B.; data curation, A.T.A.; writing—original draft preparation, A.T.A.; writing—review and editing, A.T.A. and M.A.B.; visualization, A.T.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We would like to acknowledge Delphi Labs and Jonathan Erlich for their help and fruitful discussions.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LTV	Loan to Value
AMM	Automated Market Maker
TWAP	Time-Weighted Average Price
DEX	Decentralized Exchange

Appendix A. Attack Cost Calculation for Constant Product Market

In constant product AMMs, the liquidity is defined as follows:

$$x \times y = k \quad (\text{A1})$$

If an attacker wants to move the spot price of an asset, they would need to swap against the pool, depending on whether they want to manipulate the price up or down. If we assume that they want to increase the price of y , they would need to sell some amount of x and receive some y tokens in return (adding x tokens from the pool, an attacker decreases its value). After the swap, the liquidity in the pool will be as follows:

$$(x + \Delta x)(y - \Delta y) = k = xy \quad (\text{A2})$$

From here, we can express the Δy —how much of token y attacker would receive after making a swap:

$$\Delta y = y \left(\frac{\Delta x}{x + \Delta x} \right) \quad (\text{A3})$$

Because the attacker wants to increase the price y by adding the Δx tokens and removing Δy tokens, we can express how big the change of manipulated price p_j would be:

$$p_j = \frac{x + \Delta x}{y - \Delta y} = \frac{x + \Delta x}{y - y \left(\frac{\Delta x}{x + \Delta x} \right)} = \frac{(x + \Delta x)^2}{xy} = \frac{(x + \Delta x)^2}{k} \quad (\text{A4})$$

From here, we can express the Δx —how many tokens x are needed to get the target p_j :

$$\Delta x = \sqrt{p_j \times y \times x} - x \quad (\text{A5})$$

Now, when we know both how many tokens x will be needed to make a swap and how many tokens y we receive in exchange, we can easily calculate the total attack cost by subtracting $\Delta x - \Delta y$:

$$AC = (\sqrt{p_j \times y \times x} - x) - p_j \times y \frac{\sqrt{p_j \times y \times x} - x}{x + (\sqrt{p_j \times y \times x} - x)} \quad (\text{A6})$$

Appendix B. Attack Cost Calculation for Stableswap Market

The stableswap formula first introduced by Curve protocol is as follows:

$$2^2 A(x + y) + D = 2^2 AD + \frac{D^3}{2^2 xy} \quad (\text{A7})$$

To be able to calculate the cost of an attack, we first need to express the y —how it is valued in terms of token x . For this, we rearrange Equation (8) in the form of quadratic equation which allows us to easily obtain the y formula:

$$y = \frac{(1 - \frac{1}{A}) \times (\frac{D}{4} - x) + \sqrt{[(1 - \frac{1}{A}) \times \frac{D}{4} - x]^2 + \frac{4D^3}{16Ax}}}{2} \quad (\text{A8})$$

The derivative of the function in Equation (A8) would stand for the price:

$$y' = 0.5 \frac{[-1 + \frac{1}{2}(-\frac{D^3}{Ax^2} - 2(1 - \frac{1}{A})D - x)]}{\sqrt{\frac{D^3}{Ax} + ((1 - \frac{1}{A})D - x)^2}} \quad (\text{A9})$$

To calculate the number of tokens Δx an attacker needs to swap to move the price to a specific target, we numerically solve for x in the price formula obtained in the previous step.

After we find how many tokens Δx are needed to have the price y' (or p_j to be consistent with Appendix A), we use that number to figure out how many tokens Δy would be in the pool after the swap by using Equation (A8) with the known x .

Finally, we can calculate the attack cost by subtracting the difference in token numbers before and after the swap: $AC = \Delta x - \Delta y$.

References

1. Caldarelli, G. Who Is Contributing to Academic Research on Blockchain Oracles? A Bibliometric Analysis. *Preprints* **2021**. Available online: <https://www.preprints.org/manuscript/202109.0135/v1> (accessed on 5 November 2022).
2. Zhou, L.; Xiong, X.; Ernstberger, J.; Chaliasos, S.; Wang, Z.; Wang, Y.; Qin, K.; Wattenhofer, R.; Song, D.X.; Gervais, A. SoK: Decentralized Finance (DeFi) Incidents. *arXiv* **2022**, arXiv:abs/2208.13035.
3. Wu, S.; Wang, D.; He, J.; Zhou, Y.; Wu, L.; Yuan, X.; He, Q.; Ren, K. DeFiRanger: Detecting Price Manipulation Attacks on DeFi Applications. *arXiv* **2021**, arXiv:abs/2104.15068.
4. Zhou, L.; Qin, K.; Cully, A.; Livshits, B.; Gervais, A. On the Just-In-Time Discovery of Profit-Generating Transactions in DeFi Protocols. In Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 24–27 May 2021; pp. 919–936.
5. Werner, S.M.; Perez, D.; Gudgeon, L.; Klages-Mundt, A.; Harz, D.; Knottenbelt, W.J. SoK: Decentralized Finance (DeFi). *arXiv* **2021**, arXiv:abs/2101.08778.
6. Eskandari, S.; Salehi, M.; Gu, W.C.; Clark, J. SoK: oracles from the ground truth to market manipulation. In Proceedings of the 3rd ACM Conference on Advances in Financial Technologies, Arlington Virginia, VA, USA, 26–28 September 2021.
7. Al-Breiki, H.; ur Rehman, M.H.; Salah, K.; Svetinovic, D. Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges. *IEEE Access* **2020**, *8*, 85675–85685. [[CrossRef](#)]
8. Lo, S.K.; Xu, X.; Staples, M.; Yao, L. Reliability analysis for blockchain oracles. *Comput. Electr. Eng.* **2020**, *83*, 106582. [[CrossRef](#)]

9. Liu, B.; Szalachowski, P. A First Look into DeFi Oracles. In Proceedings of the 2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), Online, 23–26 August 2021, pp. 39–48.
10. Steve Ellis, A.J.; Nazarov, S. Chainlink: A Decentralized Oracle Network. Available online: <https://research.chain.link/whitepaper-v1.pdf> (accessed on 5 November 2022).
11. Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks. Available online: <https://chain.link/whitepaper> (accessed on 5 November 2022).
12. Oraclize. A Scalable Architecture for On-Demand, Untrusted Delivery of Entropy. Available online: https://provable.xyz/papers/random_datasource-rev1.pdf (accessed on 5 November 2022).
13. bZx Hack II Full Disclosure (With Detailed Profit Analysis). Available online: <https://peckshield.medium.com/bzx-hack-ii-full-disclosure-with-detailed-profit-analysis-8126eccc1360> (accessed on 5 November 2022).
14. Understanding the Cream Finance Hack. Available online: <https://medium.com/@AndyPavia/swissblock-post-mortem-cream-finance-hack-7c1caff4335c> (accessed on 5 November 2022).
15. Harvest Finance. Available online: <https://rekt.news/harvest-finance-rekt/> (accessed on 5 November 2022).
16. Mackinga, T.; Nadahalli, T.; Wattenhofer, R. TWAP Oracle Attacks: Easier Done than Said? *Cryptology ePrint Archive*, Paper 2022/445, 2022. Available online: <https://eprint.iacr.org/2022/445> (accessed on 5 November 2022).
17. Othman, A.; Sandholm, T. Automated market-making in the large: the gates hillman prediction market. In Proceedings of the 11th ACM conference on Electronic commerce, Cambridge, MA, USA, 7–11 June 2010.
18. Goel, S.; Pennock, D.M.; Reeves, D.M.; Yu, C. Yoopick: A Combinatorial Sports Prediction Market. In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, Chicago, IL, USA, 13–17 July 2008; pp. 1880–1881.
19. Othman, A. Automated Market Making: Theory and Practice. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2012.
20. Artzner, P.; Delbaen, F.; Eber, J.M.; Heath, D. Coherent Measures of Risk. *Math. Financ.* **1999**, *9*, 203–228. [[CrossRef](#)]
21. Hanson, R.D. Combinatorial Information Market Design. *Inf. Syst. Front.* **2003**, *5*, 107–119. [[CrossRef](#)]
22. Hanson, R.D. Logarithmic Markets Scoring Rules For Modular Combinatorial Information Aggregation. *J. Predict. Mark.* **2012**, *1*, 3–15. [[CrossRef](#)]
23. Othman, A.; Pennock, D.M.; Reeves, D.M.; Sandholm, T. A Practical Liquidity-Sensitive Automated Market Maker. *ACM Trans. Econ. Comput.* **2013**, *1*, 14:1–14:25. [[CrossRef](#)]
24. VC Carvalho, A.; Silveira, D.; Ely, R.A.; Cajueiro, D.O. *A Logarithmic Market Scoring Rule Agent-Based Model to Evaluate Prediction Markets*; SSRN: Amsterdam, The Netherlands, 2022.
25. Peterson, J.; Krug, J.; Zoltu, M.; Williams, A.K.; Alexander, S. Augur: A Decentralized Oracle and Prediction Market Platform. *arXiv* **2022**, arXiv:1501.01042.
26. Gnosis Whitepaper. Available online: <https://github.com/gnosis/research/blob/master/gnosis-whitepaper.pdf> (accessed on 5 November 2022).
27. Berg, J.A.; Fritsch, R.; Heimbach, L.; Wattenhofer, R. An Empirical Study of Market Inefficiencies in Uniswap and SushiSwap. *arXiv* **2022**, arXiv:2203.07774.
28. Hayden Adams, N.Z.; Robinson, D. Uniswap v2 Core. Available online: <https://uniswap.org/whitepaper.pdf> (accessed on 5 November 2022).
29. Adams, H.; Zinsmeister, N.; Salem, M.; Keefer, R.; Robinson, D. Uniswap v3 Core. Available online: <https://uniswap.org/whitepaper-v3.pdf> (accessed on 5 November 2022).
30. Angeris, G.; Chitra, T. Improved Price Oracles: Constant Function Market Makers. In Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, 21–23 October 2020.
31. Angeris, G.; Kao, H.T.; Chiang, R.; Noyes, C.R.; Chitra, T. An Analysis of Uniswap Markets. *Cryptoecon. Syst.* **2021**, *1*. [[CrossRef](#)]
32. Egorov, M. StableSwap-Efficient Mechanism for Stablecoin liquidity. Available online: https://wikibitimg.fx994.com/attach/2020/10/189869321/WBE189869321_21425.pdf (accessed on 5 November 2022).
33. Xu, J.; Paruch, K.; Cousaert, S.; Feng, Y. SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) protocols. *arXiv* **2021**, arXiv:abs/2103.12732.
34. Hoyte, D. Median Oracle. Available online: <https://github.com/euler-xyz/median-oracle> (accessed on 5 November 2022).
35. Gudgeon, L.; Werner, S.M.; Perez, D.; Knottenbelt, W.J. DeFi Protocols for Loanable Funds: Interest Rates, Liquidity and Market Efficiency. In Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, 21–23 October 2020.
36. Aave Protocol Whitepaper. Available online: https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf (accessed on 5 November 2022).
37. R. Leshner, G.H. Compound: The Money Market Protocol. Available online: https://compound.finance/documents/Compound_Whitepaper.pdf (accessed on 5 November 2022).
38. Juliano, A. dYdX: A Standard for Decentralized Margin Trading and Derivatives. Available online: <https://whitepaper.dydx.exchange> (accessed on 5 November 2022).
39. The Maker Protocol: MakerDAO’s Multi-Collateral Dai (MCD) System. Available online: <https://makerdao.com/en/whitepaper> (accessed on 5 November 2022).

40. Krishnamachari, B.; Feng, Q.; Grippo, E. Dynamic Curves for Decentralized Autonomous Cryptocurrency Exchanges. *arXiv* **2021**. [[CrossRef](#)]
41. Egorov, M. Automatic Market-Making with Dynamic Peg. 2021. Available online: <https://classic.curve.fi/files/crypto-pools-paper.pdf> (accessed on 5 November 2022).
42. Port, A.; Tiruviluamala, N. Mixing Constant Sum and Constant Product Market Makers. *arXiv* **2022**. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.