Adaptative Exponential Smoothing for Improved Security on DeFi Protocols

Geminon protocol

info@geminon.fi

January 31, 2023

***Abstract* — Price manipulation attacks are the most common type of attack against blockchain-based decentralized finance protocols. These attacks are almost always executed using a flash loan, which is why they are also known by that name. Despite the fact that this type of attack has been carried out dozens (if not hundreds) of times and its mechanism is widely known, projects with the same vulnerabilities that allow these attacks continue to be put into production.**

In this paper, we propose the use of different variants of adaptive exponential smoothing for building price oracles and other security applications in DeFi protocols. We also show examples of its practical application in the Geminon protocol.

I. INTRODUCTION

Flash lending is a feature that was initially introduced by the Aave lending protocol in 2020, and soon after by Uniswap v2. As most decentralized exchange and lending protocols released in the following months were inspired by or forked from Aave and Uniswap, flash lending functionality ended up being available everywhere.

A flash loan allows users to borrow any amount of tokens available in a smart contract (usually a liquidity pool on a DEX or a loan pool) without posting any collateral as long as it is returned in the same transaction. By definition, this is only possible if another smart contract is used to execute the transaction, so the use of flash loans is reserved for developers specialized in smart contracts technology.

Thanks to flash loans, anyone with the necessary knowledge can have an almost unlimited amount of liquidity for their operations. The flash loan functionality was created with the aim of providing decentralized protocols with high liquidity and capital efficiency, allowing anyone to carry out large arbitrage operations without the need to have the required capital in advance, also generating extra income from fees to the protocols. However, this also opened the door to the possibility that anyone could carry out price gouging attacks on the blockchain.

II. TIME WEIGHTED AVERAGE PRICE (TWAP)

In order to avoid the aforementioned attacks, it is necessary to have an oracle that provides prices of the assets in the pool that is resistant to manipulation attempts. One of the best-known methods is the one proposed by Uniswap v2 (Adams et al., 2020), which consists of calculating a time-weighted average from an accumulator of the prices at each second of the history of the pool:

$$a_t = \sum_{i=1}^{t} p_i$$

Since it is not feasible to update the accumulator every second, the amount that is calculated in practice is the product of the price times the time interval elapsed since the last update of the accumulator:

$$a_{t_n} = \sum_{n=1}^{m} p_n \Delta t_n$$

The accumulator is automatically updated by the liquidity pool each time an operation is performed on it. Using such an accumulator, it is possible to build a price oracle by reading the initial and final values of the accumulator at a time interval and calculating the following expression:

$$\bar{p}_{t_1,t_2} = \frac{a_{t_2} - a_{t_1}}{t_2 - t_1}$$

This method, however, has several drawbacks:

- Since the liquidity pool only stores the last value of the accumulator, to build the oracle it is necessary to send periodic transactions to read and store the accumulator checkpoints required for the calculation.
- As it is an average calculated in the time domain, it has the same limitations as any moving average: the calculated price has a lag equal to the half-period of the average and the ability to filter attacks is directly proportional to this lag, so the greater security required by the oracle, the less accuracy is obtained in the calculated prices.
- The prices are inconsistent: for a pair of assets X and Y in a liquidity pool, the oracle price of X with respect to Y and the reciprocal of the oracle price of Y with respect to X are different. This implies that it is necessary to calculate and store both prices separately, so the cost of gas is double.

III. TIME WEIGHTED GEOMETRIC MEAN PRICE (TWGMP)

In order to overcome some of these drawbacks, Uniswap v3 (Adams et al., 2021) introduced some changes to the accumulator calculation method:

- Instead of only storing the last accumulator value, it is possible to optionally store a list with up to 65536 accumulator values in the liquidity pool contract.
- The accumulator stores the base 1.0001 logarithm of the price instead of the price. This allows the oracle to be calculated using the geometric mean of the prices, instead of the arithmetic mean, which solves the problem of price mismatch between the assets of the pair, allowing a single accumulator to be stored.

More formally, the accumulator used by the Uniswap v3 protocol has the form:

$$a_t = \sum_{i=1}^{t} \log_{1.0001}(p_i)$$

And the oracle price can be obtained by calculating the geometric mean of the previous accumulator, using the expression:

$$\bar{p}_{t_1,t_2} = \left(\prod_{i=t_1}^{t_2} p_i \right)^{\frac{1}{t_2 - t_1}} = 1.0001^{\frac{a_{t_2} - a_{t_1}}{t_2 - t_1}}$$

Although this model solves the problems of price mismatch between the pair's assets and the need to make a transaction to create the accumulator checkpoints, it still suffers from the main drawback of TWAP: the need to choose between oracle price precision (lag) and its ability to filter attacks (smoothing level).

IV. Volume Weighted Exponentially Smoothed Price (VWESP)

Given the previously exposed limitations of models based on time-weighted averages, the Geminon protocol has used a completely new system based on exponential smoothing for the construction of its oracles.

Exponential smoothing methods are widely used in industry due to several practical considerations in short-range forecasting. Model formulations are relatively simple (Gardner, 1985) and were developed in the 1950s by Robert G. Brown, having their first applications in demand forecasting for inventory management, planning and production control (Gardner, 2005).

The equation of Brown's simple exponential smoothing model in its recursive form can be written as:

$$S_t = \alpha X_t + (1 - \alpha) S_{t-1}$$

The model proposed by Geminon for calculating a price oracle has two differences with respect to simple exponential smoothing:

- It is an adaptive model, that is, the smoothing parameter is a function instead of a constant.
- It works in the volume (amount) domain instead of the time domain.

The smoothing parameter α for the VWESP is obtained from the relationship between the volume of the current trade v_n and the average volume of trades in the pool $\bar{V}_n$:

$$\alpha_n = \frac{\bar{V}_n}{v_n}$$

The parameter α takes its maximum value, limited to 1, when the amount of the current trade is less than the mean volume of the pool. If the volume of the last trade is higher than the usual volume, then $\alpha < 1$ and the price is filtered proportionally to the volume anomaly using a simple exponential smoothing:

$$\bar{p}_n = \alpha_n p_n + (1 - \alpha_n) \bar{p}_{n-1}$$

To keep track of the average volume of operations, simple exponential smoothing can also be used:

$$\bar{V}_n = \beta v_n + (1 - \beta) \bar{V}_{n-1}$$

V. METHOD COMPARISON

Next we analyze the behavior of the different oracles in different scenarios. For this we use the classical approach of digital signal processing, introducing the basic functions of ramp, step and impulse in the different algorithms to compare their response.

Five oracle algorithms have been compared:

- TWAP: uses the moving average approach, the oracle price value is calculated from the last checkpoint to the instant the oracle is queried.
- TWAP CP: uses the fixed window approach, the oracle price is the value of the TWAP calculated in the last checkpoint (it remains constant over all the period).
- TWGMP: uses the moving average approach as described before.
- TWGMP CP: uses the fixed window approach as described before.
- VWESP: uses the adaptive exponential smoothing method proposed in this paper.

A. Response to ramp

For this test a linear price increase function ($y = x$) is used. For oracles based on time-weighted averages, an interval of 10 periods between checkpoints is used. For VWESP the volume of operations is assumed to be constant.

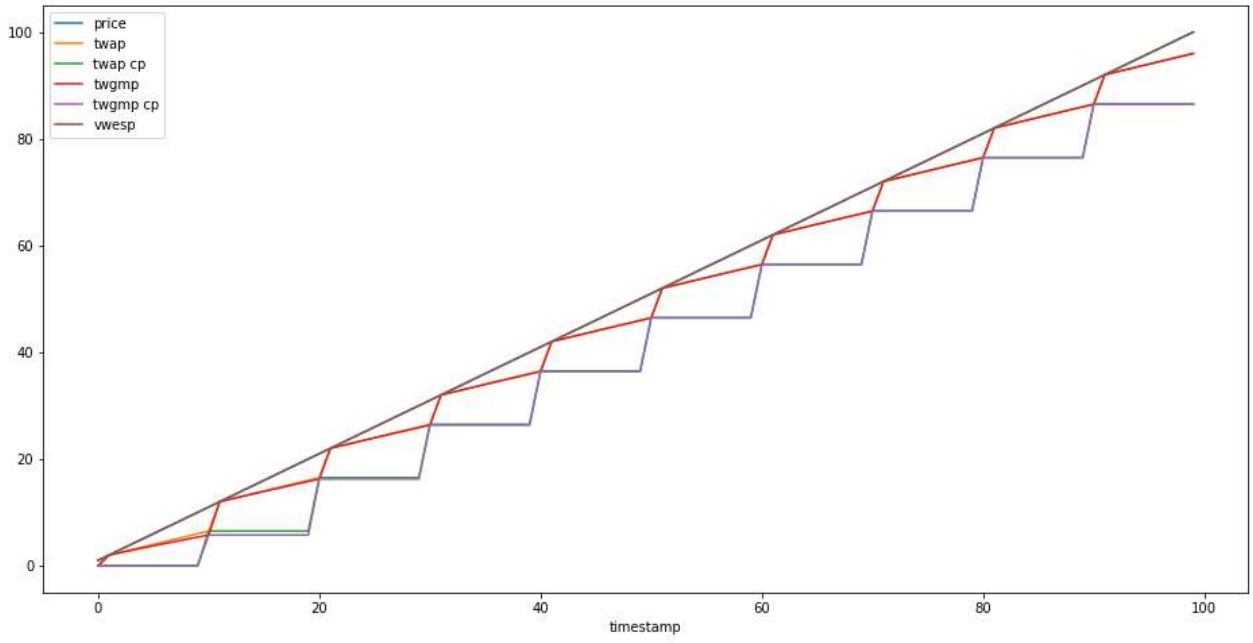

Figure 1: Oracles response to the price ramp function.

The results show that TWAP and TWGMP give identical results for trending prices, with the fixed window variants TWAP CP and TWGMP CP having a much bigger lag than the moving average versions, as expected. This lag is increased if a larger interval between checkpoints is used. VWESP is the only one capable of perfectly following price with zero lag, so its curve overlaps that of price.

B. Response to step

In this test an instantaneous level change in the series is simulated. For oracles based on time-weighted averages, an interval of 10 periods between checkpoints is used. For VWESP the volume of operations is assumed to be constant.

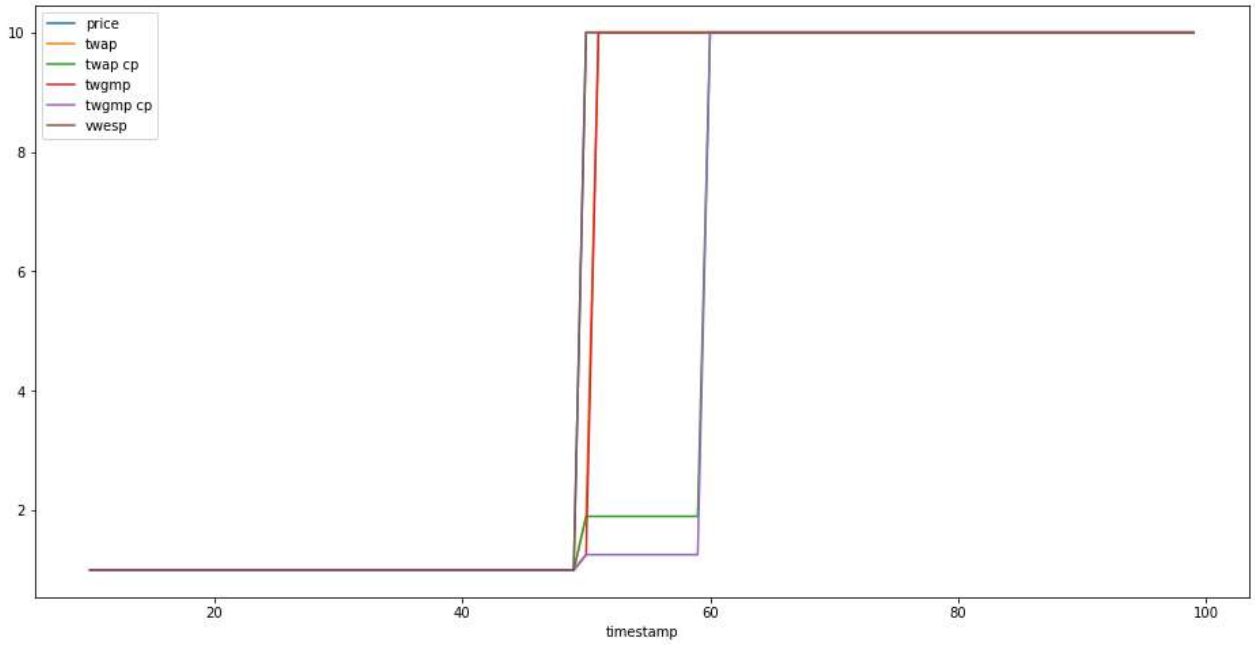

Figure 2: Oracles response to the price step function.

TWAP has a slightly faster initial step response than TWGMP, although both take the same time to converge to the new series level. TWAP CP and TWGMP CP show a much longer delay to adapt to the level change, being TWGMP CP the one with the slowest initial response, the same as in the previous case. VWESP is the only algorithm capable of instantly reflecting the change in level, so its curve overlaps that of the price.

C. Response to impulse (flash loan attack)

In this test, the price changes level for a period to a value 1000 times higher, and then returns to the previous level. The parameters of the oracles based on time-weighted averages are the same as in the rest of the experiments. For VWESP, on the other hand, a volume in the price impulse is assumed 1000 times greater than normal, simulating a flash loan attack.

TWAP and TWAP CP fail the test, showing an instant oracle price increase of 100 times. Also, TWAP CP still shows the price altered by the attack several periods later. Although it would be possible to reduce this impact by using a longer averaging period, this would cause the delay to the ramp and step functions to be greater, and would impair the performance of TWAP CP since it would maintain a (somewhat less) altered price for longer. TWGMP, TWGMP CP and VWESP are capable of adequately filtering the price anomaly, remaining practically unchanged.

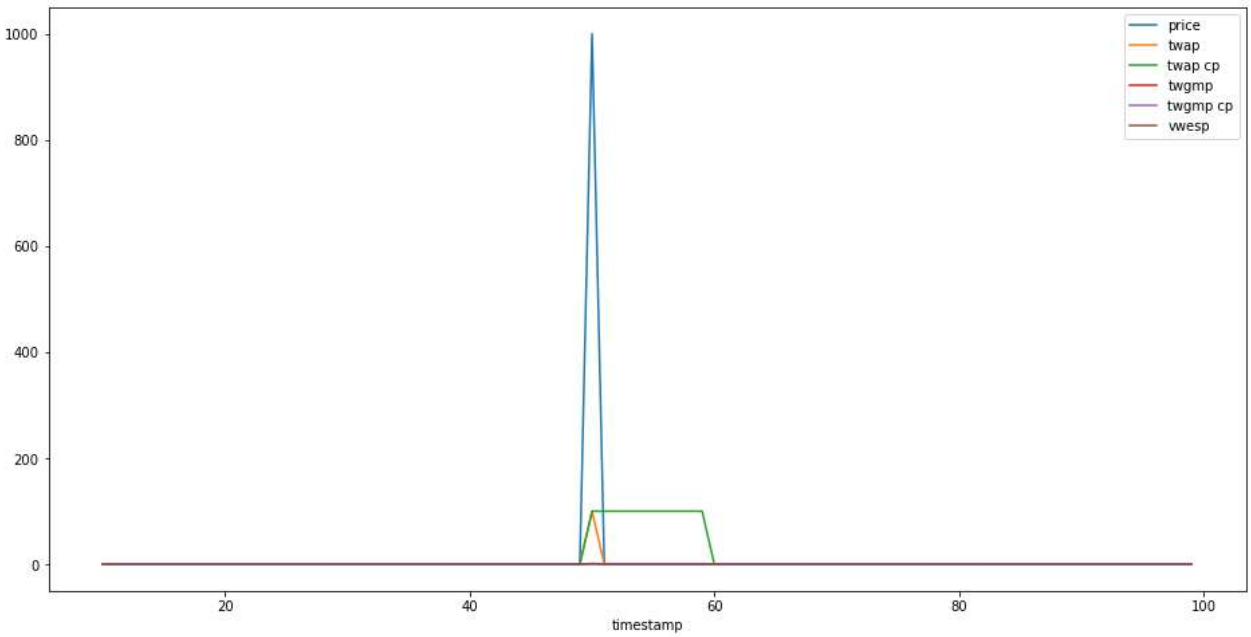

Figure 3: Oracles response to the price impulse function.

D. Response to negative impulse (flash loan attack)

This test maintains the same parameters as the previous one for the oracles, but a price drop to a level 1000 times lower is simulated to later recover the previous level.

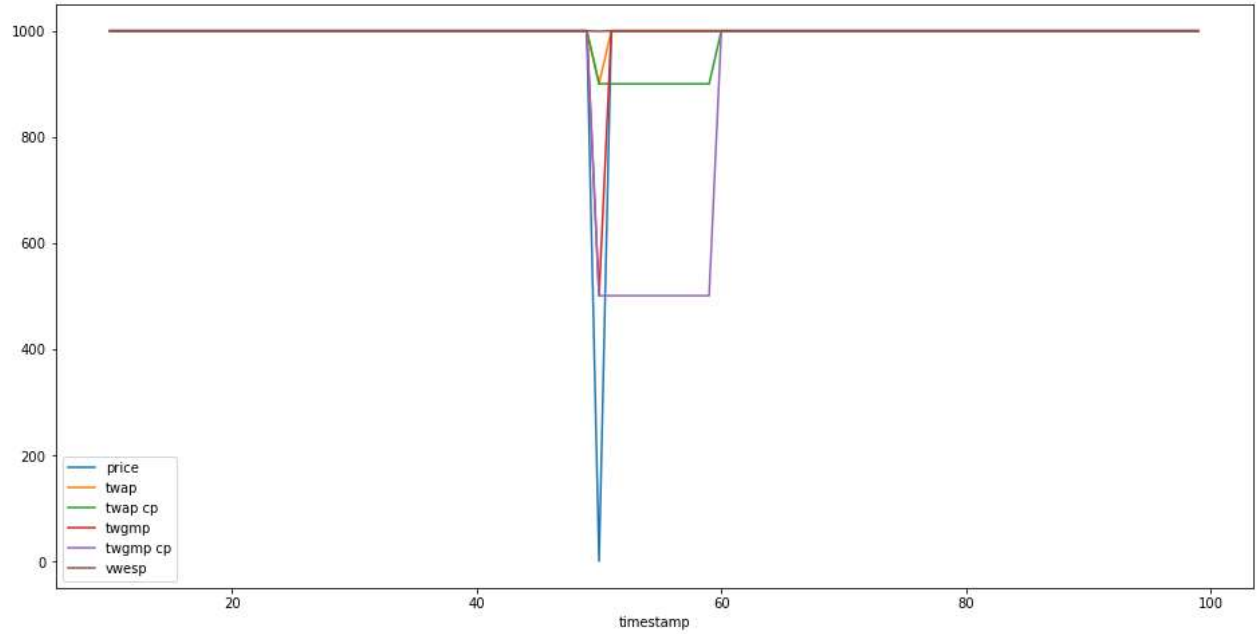

Figure 4: Oracles response to the price negative impulse function.

In this case, it is TWGMP and TWGMP CP that get the worst result, showing an instant decrease in the oracle price of 50%. TWAP and TWAP CP also fail the test, showing a 10% drop in the oracle price similar to the rise shown in the positive impulse test. Analogous to the previous case, TWGMP CP and TWAP CP continue to show the price altered by the attack several periods later. VWESP is the only one able to properly filter the negative price anomaly.

VI. PRACTICAL APPLICATIONS

The Geminon protocol uses different variants of the adaptive exponential smoothing algorithm described here in several of its smart contracts to guarantee their security against different types of attacks. In the following sections we show examples of these uses.

A. Price oracle for Liquidity Pools

Geminon Genesis Liquidity Pools (GLPs) are part of a complete decentralized finance system that includes stablecoin minting and lending. These types of applications are vulnerable to price manipulation attacks, and therefore need reliable sources of prices for their calculations. To protect the protocol against these kind of attacks, an internal oracle has been incorporated into the GLPs that allows secure token pricing.

The GLP internal oracle algorithm is directly based on the adaptive exponential smoothing model described in this paper. To further improve the endurance of the oracle, it calculates two values, an instantaneous value that includes the operations within the current block and another considered the safe value that is only updated when a new block is created.

1) Instant value

The oracle's instantaneous value, l_n, is updated each time an operation occurs on a GLP. The smoothing parameter, β, is obtained from the relationship between the volume of the current trade, v_n, and the average volume of trades in the pool $\bar{V}_n$:

$$\beta_n = \begin{cases} 1 \quad , & \beta_n > 1 \\ \dfrac{\bar{V}_n}{v_n + \varepsilon}, & \beta_n \leq 1 \end{cases}$$

$$\bar{l}_n = \beta_n p_n + (1 - \beta_n)\bar{l}_{n-1}$$

The parameter β takes its maximum value, limited to 1, when the volume of the current trade is less than the usual volume of the pool. In that case, the instantaneous value of the oracle price l_n matches the instantaneous price of the pool p_n. If the volume of the last trade is higher than the usual volume, then $\beta < 1$ and the price is filtered proportionally to the volume anomaly. For example, if the volume of the operation is 1000 times higher than normal, then the new price will have a weight of 0.001 in the average price l_n and its value will not affect the instantaneous price of the oracle, achieving its objective of filtering out abnormal operations.

To calculate the average volume of operations, a simple exponential smoothing is used, with parameter $\gamma = 0.001$. This smoothing is equivalent to using the average of the volume of the last 1000 trades:

$$\bar{V}_{n+1} = \gamma v_n + (1 - \gamma)\bar{V}_n$$

The update of the average volume of operations is done afterwards, so the volume of the current operation is not taken into account in the oracle calculations until the next operation. This detail is important, since it prevents a flash loan from altering the average volume data used to calculate the parameter β, improving the filtering capacity of the algorithm.

Finally, the accumulated volume of the current block is calculated:

$$V_B = \begin{cases} 0 \quad , & \Delta t > 0 \\ \sum_{n \in B} v_n \, , & \Delta t \leq 0 \end{cases}$$

2) Safe value

The oracle safe value is calculated only at the start of a new block. The calculation is similar to that of the instantaneous value, the difference being that the safe value takes as input the values of the oracle's instantaneous value, producing second-order filtering. For the smoothing parameter α, the cumulative volume of the previous block V_{B-1} is used instead of the volume of the last trade:

$$\alpha_n = \begin{cases} 1 & , \quad \alpha_n > 1 \\ \dfrac{\bar{V}_n}{V_{B-1} + \varepsilon}, & \alpha_n \leq 1 \end{cases}$$

Finally, to calculate the safe value of the price m_n, the smoothed instantaneous value of the previous block is used as input, l_{n-1}, instead of the current price p_n:

$$\bar{m}_n = \alpha_n \bar{l}_{n-1} + (1 - \alpha_n)\bar{m}_{n-1}$$

This calculation structure ensures that the safe price m_n cannot be affected by a flash loan attack, since it takes the closing price of the previous block, which has also been previously filtered, ruling out operations with anomalous volume, and which is again filtered through of the parameter α. This double filtering also ensures that even spot attacks are properly discarded until several trades with normal volume are made in the pool at the new price.

The justification for security against spot attacks that last longer than the block time of the network is found in the hypothesis of market efficiency and the principle of non-arbitration. An attacker who tries to manipulate the price of a pool without using a flash loan is exposed to market reaction (arbitrage) in the next block to restore the price balance, so the price resulting from the following operations will be a fair market price that must be reflected by the oracle.

This ability to immediately reflect price changes when normal trading flow is established while filtering out one-off anomalies is possible because the smoothing is done in the volume domain rather than the time domain, therefore, a great noise filtering capacity is achieved without introducing delay in the smoothed series.

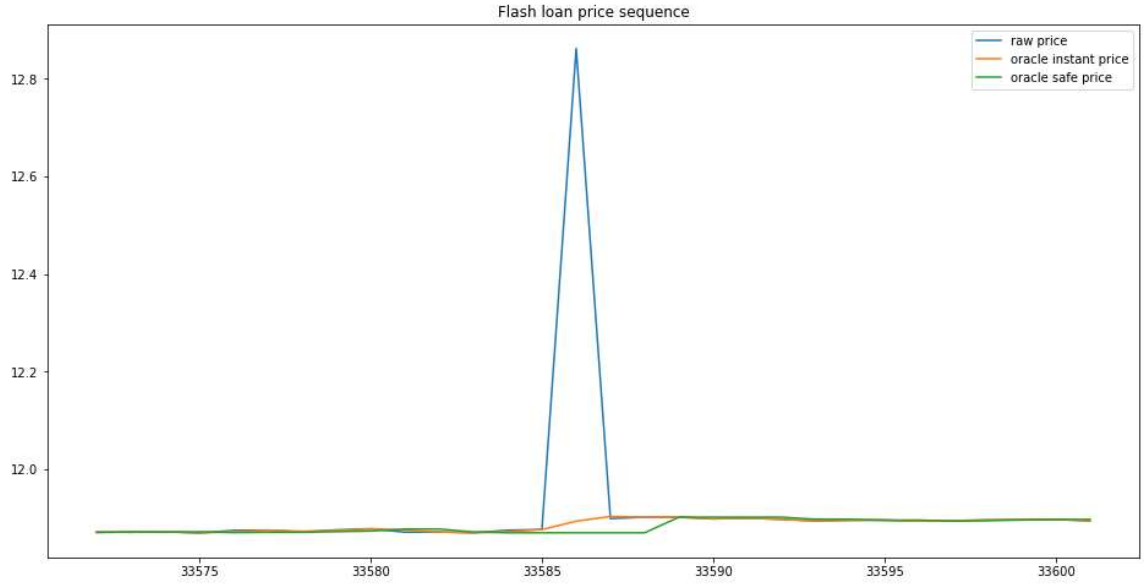

Figure 5: Pool price sequence during a flash loan attack. Raw price and oracle prices.

To test the performance of this algorithm, we carried out simulations of the behavior of the internal oracle of a GLP against different attack patterns using flash lending. No attack pattern managed to manipulate the secure pool price within the same block, implying that no flash loan attack would have been effective against the protocol, even with only one GLP running on the network. The figure shows the pool price sequence during one of the simulated attacks. It can be seen how the attack produces an instantaneous spike in the raw pool price, while the oracle instantaneous price perfectly filters the attack and the safe price remains unchanged until it ends. It can also be seen how during normal operations before and after the attack, the oracle faithfully reflects the exact price of the pool without any lag.

B. Supply limiter for Elastic Supply tokens

The Geminon protocol allows to mint different tokens. As a security measure, the contracts for these tokens incorporate a daily supply limiter that prevents a large number of tokens from being created in a short period of time. The algorithm of said limiter, developed by Geminon, is based on adaptive exponential smoothing whose parameters vary depending on the interval elapsed since the last minting or burning of the token.

$$\delta = \frac{T}{\Delta t + \varepsilon}$$

Where T is the period of time in seconds for which we want to keep track of the supply and Δt the time elapsed since the last supply variation in seconds. Given δ, the weights of the exponential smoothing are calculated as follows:

$$\alpha = \frac{2}{1+\delta}$$

$$w_1 = \begin{cases} 1\ , & \Delta t = 0 \\ \alpha \cdot \delta, & \Delta t > 0 \end{cases}$$

$$w_2 = \begin{cases} 1\ , & \Delta t = 0 \\ 1 - \alpha, & \Delta t > 0 \end{cases}$$

And finally, the value of the limit supply is given by:

$$\lambda_n = w_1 v_n + w_2 \lambda_{n-1}$$

The described algorithm is designed to approximate the value of a 24-hour period (T=86400) moving average calculated on a discrete time series of variable frequency using a minimum calculation.

In a traditional computer application, if we want to monitor the accumulated value of a discrete variable in a certain time interval, what we would do is create a table that stores each value of the variable together with its timestamp, and use said table to calculate an average of all values that are within the desired time window, for example the last 24 hours.

In a blockchain application however such a basic calculation would be cost prohibitive. First, because storing data on the blockchain, especially a congested one like Ethereum, is extremely expensive. And second, because in each transaction it would be necessary to iterate over the lists of values and times stored, with which the complexity of the calculation and with it the cost of the transaction would grow with the number of data stored in the period.

This approach would not only be disproportionately expensive but also dangerous: given a high enough trading volume, the network's gas limit per block could be reached, causing the minting functions of the token in question to be temporarily blocked due to reversing transactions that exceed said gas limit. A malicious actor could exploit this vulnerability to launch denial of service attacks against the protocol.

If we were dealing with a uniform time series, with the values separated at regular intervals in time, the period t moving average could be easily approximated simply by using an equivalent exponential moving average with parameter $\alpha = 2/(1+t)$. In this case, since there is also no possibility of imputing the missing values to transform the irregular series into a regular one, the only option is to introduce a variable that compensates for the irregularity in the frequency domain, adapting the intensity of the smoothing that is applied to each new data depending on how far it is from the basic frequency defined by the period of the moving average that is being approximated (86400 seconds in our case).

The autoregressive algorithm designed by Geminon is capable of optimally approximating the explicitly calculated moving average curve.

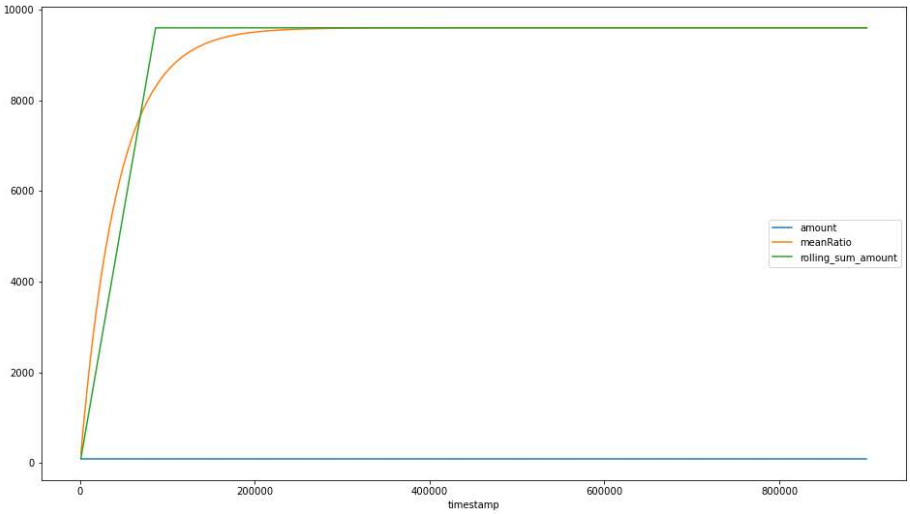

Figure 6: Response of the estimator (orange) to a constant signal (blue). Real value of the series in green.

The estimator is able to perfectly converge to a level shift in the target function, and even when a series of random values is entered at random time intervals it is capable of accurately tracking the real value of the moving average without needing to store the series values:

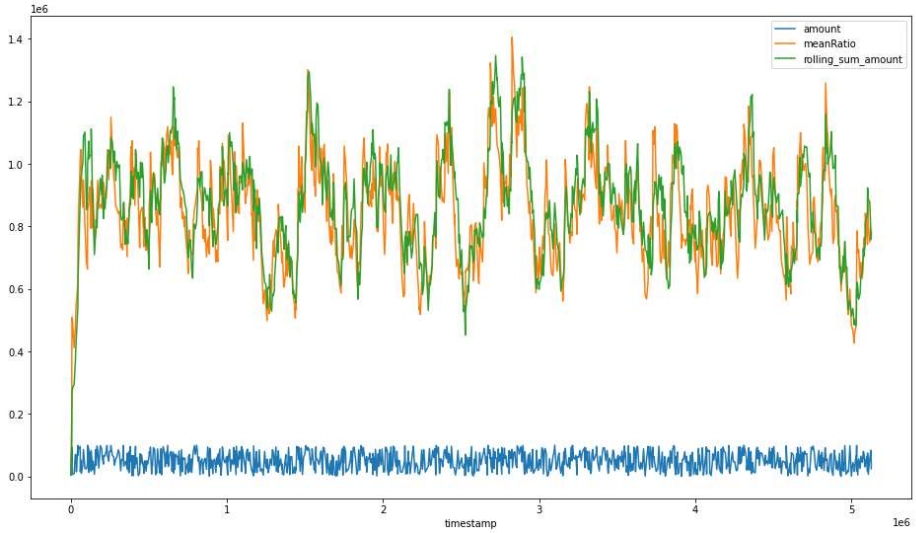

Figure 7: Response of the estimator (orange) to a random signal in amplitude and frequency (blue).

C. StableSwap front-running protection

Geminon stablecoins use Chainlink oracles (Ellis et al., 2017) to obtain real-time prices against the dollar of the fiat currency that they replicate. However, due to the intrinsic characteristics of blockchain technology, there is a delay (lag) between the moment the price is observed in the market and the moment the data is written to the blockchain. This delay is the sum of the time it takes for the Chainlink network to reach consensus on the value of the price data, the time it takes for Chainlink to send the transaction to the destination blockchain with the new value, and the time that the network takes to process the transaction and include it in the next block. The total time that this process lasts can be more than a minute.

Due to the significant delay with which the oracle reflects on-chain prices, it is theoretically possible for an attacker to make a risk-free profit by trading an on-chain asset that is priced using this oracle, since it would only be necessary to observe the price of the asset in the real world to know in advance how its price will change on the blockchain in the immediate future (temporary arbitrage). The attack will only be economically viable if the cost of the operation (gas + commissions + slippage) is less than the expected price movement. To avoid this possibility, a security system has been designed to make this type of attack unfeasible.

The system uses an internal price oracle that tracks global stablecoin positions taken in the last 5 minutes in the smart contract. Every time an operation is carried out, the contract checks if opposite positions have been opened in the last 5 minutes that could produce an abnormal profit. To achieve this, an algorithm similar to those described above is used.

The algorithm is based on an adaptive exponential moving average whose parameter varies depending on the relationship between the volume of the current operation and the total volume accumulated in the last 5 minutes:

$$V_n = \begin{cases} v_n, & \Delta t > 300 \\ V_{n-1} + v_n, & \Delta t \leq 300 \end{cases}$$

$$w_n = \begin{cases} 1, & \Delta t > 300 \\ \dfrac{v_n}{V_n + v_n}, & \Delta t \leq 300 \end{cases}$$

$$m_n = w_n p_n + (1 - w_n) m_{n-1}$$

Analogously to the supply limiter exposed in the previous point, this algorithm is capable of following with great precision the average price of the positions taken by traders in the last 5 minutes in a certain instrument. Knowing said position, it can be estimated if the current trade can generate an unfair profit due to an oracle front-running.

VII. CONCLUSION

In this paper we have presented a new type of price oracle based on adaptive exponential smoothing. The proposed oracle, which works in the domain of the volume of operations, is vastly superior to the oracles used so far that are based on time-weighted averages, both in terms of accuracy and security against manipulation attempts.

Tests carried out using digital signal processing techniques show that oracles based on time-weighted averages on price accumulators, such as those proposed by the Uniswap protocol, present accuracy problems due to the lag they introduce into the price signal and they also fail to completely filter price gouging attacks. Additionally, we show that the use of a geometric mean proposed in Uniswap V3 does not produce an improvement in terms of accuracy or security with respect to the arithmetic mean used in Uniswap V2. Said oracle also presents an asymmetry due to the use of the logarithm function which, although it improves the ability to filter high-price anomalies, significantly worsens the ability to filter low-price anomalies. This asymmetry in the behavior of the Uniswap V3 oracle must be taken into account by developers when using it as a source of prices for their projects.

Finally, we have shown several practical examples of the application of adaptive exponential smoothing in the Geminon protocol, both in the volume domain and in the time domain. These examples show that the technique is very versatile and can be adapted to different applications in DeFi protocols, helping to greatly improve the security of these protocols.

REFERENCES

Aave (2020). Protocol Whitepaper V1.

Aave (2020). Protocol Whitepaper V2.

Adams, H., Zinsmeister, N. & Robinson, D. (2020). Uniswap v2 Core.

Adams, H., Zinsmeister, N., Salem, M., Keefer, R. & Robinson, D. (2021). Uniswap v3 Core.

Ellis, S., Juels, A. & Nazarov, S. (2017). Chainlink: A Decentralized Oracle Network.

Gardner, E.S. (1985). Exponential Smoothing. The State of the Art.

Gardner, E.S. (2005). Exponential Smoothing. The State of the Art - Part II.