# SecDeLP : Secure Decentralized Lending Platforms against Oracle Manipulation Attacks

Sanidhay Arora
sanidhay@uoregon.edu
University of Oregon
Eugene, USA

Yingjiu Li
yingjiul@uoregon.edu
University of Oregon
Eugene, USA

Yebo Feng
yebof@uoregon.edu
University of Oregon
Eugene, USA

Jiahua Xu
jiahua.xu@ucl.ac.uk
University College London
London, UK

## ABSTRACT

As an integral part of the decentralized finance (DeFi) ecosystem, decentralized lending platforms (DLPs) have gained massive traction with the recently revived interest in blockchain technology. However, with the traction and the novel services that are being emerged in the DeFi space, several interesting security vulnerabilities and attacks have been observed in the last few years. *Oracle manipulation attacks* have been witnessed numerous times on DLPs, and in this paper, we aim to secure DLPs from these attacks. We propose an algorithmic solution called SecDeLP , that secures a general DLP against oracle manipulation attacks that are performed using *flash loans*. We provide a theorem to prove that if certain conditions are satisfied on some specific system and input parameters then oracle manipulation attacks using flash loans must be reverted, hence preventing the attack. Furthermore, we present a practical analysis using empirical data and show that the constraints used in our solution are reasonable. Specifically, we introduce a new input parameter in the SecDeLP algorithm that is required for each crypto-asset available on the DLP. Next, using the past three years of market price data of several crypto-assets, we illustrate safe-to-use values for this input parameter. We show that our requirements on this parameter are satisfied with a high degree of confidence. We also show that the cost overhead for implementing SecDeLP is minimal and practical.

## CCS CONCEPTS

• **Blockchain**; • **Decentralized Finanace Security**;

## KEYWORDS

Blockchain, Flash Loan, Oracle Manipulation Attack, DeFi


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

**ACM Reference Format:**
Sanidhay Arora, Yingjiu Li, Yebo Feng, and Jiahua Xu. 2023. SecDeLP : Secure Decentralized Lending Platforms against Oracle Manipulation Attacks. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Recently, there have been in-depth conversations regarding the potential applications of blockchain technology, and the rise of decentralized finance (DeFi) has emerged as a significant force driving blockchain adoption. In contrast to traditional finance, DeFi utilizes transparent and unalterable on-chain smart contracts to execute trading activity, eliminating the need for centralized intermediaries such as banks, brokers, and custodians. As smart contracts are open-source and self-governing, recent advancements in DeFi have led to the identification of notable security vulnerabilities and attack mechanisms. In total, all DeFi attacks have amassed over $1B [6] to date with over 1M% ROI. In this paper, we focus on a specific security vulnerability in *Decentralized Lending Platforms (DLPs)*, which are an integral part of the DeFi world and have incurred over $100M [6] in losses due to certain attacks and vulnerabilities.

One of the most notable security vulnerabilities in Decentralized Lending Platforms (DLPs) arises from the usage of price oracles. Price oracles are sources of market price data provided by a third-party service. Generally, this third-party service is provided by a set of smart contracts. The DLPs use these price oracles to determine the spot price of the cryptocurrency assets available on their platform. However, these oracles can be manipulated and hence, the DLP can be exploited. In this paper, we focus on securing DLPs against *Oracle manipulation attacks*.

Specifically, this work focuses on Oracle manipulation attacks on DLPs that are performed using *Flash loans*. In the last few years, Flash loans have emerged as a novel service in the DeFi world. Flash loans provide a user with the ability to borrow vast amounts of capital by only risking minimal gas fees. This ability proves to be extremely powerful in its capabilities to manipulate the cryptocurrency market. Flash loan providers, price oracles, and DLPs, all operate on smart contracts and hence are publicly accessible to anyone. A malicious user can easily write programs to leverage flash loans and use the borrowed capital to manipulate a price oracle. Any DLP using this manipulated price oracle can now be easily exploited. In the recent past, we have witnessed several interesting

attacks being performed on DLPs with over $\$100M$ of value stolen in crypto-assets [6].

Oracle manipulation attacks can be extremely difficult to prevent. A notable reason for this difficulty is that these attacks happen in a single transaction. Moreover, since multiple smart contracts interact with each other, the complexity of financial transactions across multiple DeFi platforms makes it even more challenging to find security vulnerabilities. In our work, we provide an algorithm that can be used by a DLP to prevent these attacks with a high degree of confidence.

The key idea of our solution is to keep track of a Price-state for each crypto-asset. This state stores *(i)* the last used price of a crypto-asset and *(ii)* the timestamp of the block in which this price state was last updated. We also introduce an input parameter for each available crypto-asset in our algorithm. We then impose certain constraints on these parameters. These constraints ensure that the DLP only uses a price oracle if the last stored price of the asset is within a certain threshold. We equate this threshold value to the minimum price distortion required by an adversary to gain a profit in the attack transaction. Since the attacker is unable to profit, she will not be able to pay back the flash loan, and hence the attack is prevented.

Next, we show that SecDeLP is easy to implement and the algorithm requirements are practically reasonable. Specifically, we require that a stored price state of an asset must be updated every $T$ blocks on the blockchain. Further, we require an input variable $\alpha$ for each crypto-asset. $\alpha$ represents an acceptable fractional change in the price of an asset over a fixed period of time. The value of $\alpha$ and $T$ can be set by the DLP based on empirical data. We collect the last three years of market data and provide an empirical analysis of SecDeLP . We illustrate practical values for these parameters that can be used by a general SecDeLP (Section 7). In the same section, we show that SecDeLP is a resource and cost-efficient solution.

## 1.1 Our Contributions

This paper presents SecDeLP , a novel and practical solution for DLPs to prevent oracle manipulation attacks that are performed using flash loans. Specifically, our contributions are as follows:

- **Easy-to-implement solution to oracle manipulation attack.** We propose a novel security solution for a general DLP called SecDeLP . Specifically, we present an algorithm that can easily be implemented in a general DLP, which prevents Oracle Manipulation attacks that are performed using flash loans (Section 6). The key idea of this solution is to prevent the adversary from paying back the flash loan. This idea is implemented by making sure that certain conditions and constraints on system parameters are satisfied in the DLP's state (Proposition 6.4).
- **Oracle Independent and Agnostic.** Existing solutions focused on preventing Oracle manipulation attacks are applied on Oracle's source and not the platform using a price oracle The use of multiple oracles is a proven technique to improve DLP's security. SecDeLP is an algorithm that can be applied to any general DLP as an inherent security measure in its architecture. Hence, SecDeLP allows the DLP to use multiple oracles without the need to consider security issues on the oracle's side, which is a major benefit of using SecDeLP . SecDeLP is a theoretical solution that reverts all attack transactions using simple and proven logic (Theorem 6.9), and hence is reliable.
- **Adaptable and steerable solution.** We consider two system parameters $\alpha$ and $T$ in SecDeLP . Specifically, SecDeLP allows a DLP to be able to quantify its security in terms of risk ($\alpha$) and availability ($T$) based on these parameters (Section 7). These values may vary for different available assets providing developers with modularity and more control over their platform. We provide an empirical analysis and demonstrate how to calculate these values for different assets in real-time to ensure a high degree of security.
- **Resource and cost efficient.** Finally (Section 7), we collect the data on gas prices on the Ethereum blockchain over the last three years and consider the worst-case scenario in terms of operational cost. At the current price of Ethereum ($1500), this cost would be approximately $20 per asset per minute. This cost comes down to an operational cost of less than $1 per asset per minute in the best-case scenario. Considering the aforementioned cases, we found that the additional operational costs incurred by using SecDeLP are minimal on average and negligible when compared with the overall operational cost of a DLP.

**Organization.** The rest of the paper is organized as follows. In Section 2 we discuss the background and compare our solution with the related literature. Then, in Section 3 we explain some important preliminary concepts of the DeFi world which are required for understanding our work. In Section 4, we introduce a formal model of a general DLP, including the current security model and security control measures used by a standard DLP. In Section 5 we provide the adversary model and model a basic Oracle manipulation attack on a standard DLP. This model contains the necessary procedures to capture this type of attack. Then, in Section 6 we present our solution SecDeLP . We provide a Proof-of-concept algorithmic solution for DLPs that prevents this attack. We state a proposition, define four lemmas to prove the proposition, and finally provide a theorem stating that SecDeLP is secure under reasonable practical requirements on system parameters. In Section 7, we do a practicality analysis of SecDeLP based on empirical data collected over the past 3 years. We provide empirical analysis to show that the cost of implementing this design is minimal compared to the overall cost required to run a DLP. Finally, in Section 8 we discuss the state-of-the-art security measures and techniques that could be implemented and used by DLPs.

# 2 BACKGROUND AND RELATED WORK

Decentralized Finance security has been a hot research topic these days due to the prosperous applications based on smart contracts. Fabian in [7] highlights opportunities and potential risks of the DeFi ecosystem. The paper provides a multi-layered framework to analyze the implicit architecture and the various DeFi building blocks, including token standards, decentralized exchanges, decentralized debt markets, blockchain derivatives, and on-chain asset management protocols. However, the paper does not fully address the scalability challenges that blockchain-based financial markets face, which can limit their ability to compete with traditional financial markets and support high transaction volumes.

Grish et al. in [3] provide a novel and promising approach to the security analysis of Ethereum smart contracts, which could help developers identify and prevent security vulnerabilities in their smart contracts more effectively. The authors evaluate their approach on a set of real-world smart contracts and demonstrate that their framework is able to detect a wide range of security vulnerabilities, including reentrancy attacks, integer overflows, and timestamp dependence vulnerabilities. However, the framework presented in the paper is limited in its ability to handle complex data structures, such as arrays and mappings, which are commonly used in smart contracts. This limitation may result in false positives or false negatives in the analysis. While the framework is able to detect a wide range of security vulnerabilities, it may not cover all potential attack vectors or zero-day exploits. This limitation highlights the need for ongoing research and development of new techniques and tools for the security analysis of smart contracts.

"The oracle problem", had recently drawn attention to DeFi, given the crescent number of related hacks that caused the loss of millions of dollars held in DeFi projects. Giulio and Joshua in [2] shed light on the pattern that identifies the oracle problem in DeFi and outline the most promising ways to overcome the related weaknesses. They discuss the current risks connected with the implementation of oracles in DeFi, and how they are addressed. The two most common solutions are decentralized price oracles and DeFi security platforms. However, these solutions are not yet completely adopted by the developer community and are still in the development stages. It is interesting to note that the above-mentioned solutions involve a third party, i.e. either the Oracle source or a security audit platform. We develop a novel solution to prevent oracle manipulation attacks that is theoretically sound and does not involve other parties.

Recently, oracle manipulation attacks on DLPs have become quite popular amassing over $100M in damages in the last 2 years. Massimo et al. in [1] systematize the existing knowledge on DLPs. They discuss the security and risk considerations associated with DLPs. These include the risk of smart contract vulnerabilities, the risk of liquidity crises, and the risk of oracle manipulation attacks using flash loans. Although they discuss the potential vulnerabilities and attack possibilities, they do not provide any security measures to prevent such attacks. In our work, we present a novel solution to prevent Oracle manipulation attacks on DLPs.

Flash loans and their ability to manipulate the market is another important security consideration attracting many researchers' attention. We have recently witnessed many papers being published focusing on preventing these attacks. Yixin et al. in [5] are the first to explore the implication of transaction atomicity and flash loans for the nascent decentralized finance (DeFi) ecosystem. The authors also show that an adversary can maximize the attack profit efficiently (in less than 13ms) due to the atomic transaction property. They achieve this by providing an optimization framework to quantify the parameters that yield the maximum revenue an adversary can gain. They also propose "Flashshot", a prototype that is able to transparently illustrate the precise asset flows intertwined with smart contracts in a standardized diagram for each Flash Loan event. While the paper provides a good overview of flash loan attacks, it does not provide detailed information on the technical methods used to execute the attacks. In our work, we detail the specific methods and techniques required for an adversary to perform a successful oracle manipulation attack. This information could be useful for developers and researchers to better understand how to prevent and mitigate these types of attacks. We also use a Flashot prototype to illustrate an example of an oracle manipulation attack.

## 3 PRELIMINARIES

In the following, we outline the required background of blockchains and DeFi applications for our proposed solution to prevent Oracle Manipulation Attacks.

**Oracle.** An oracle is a bridge between the blockchain and the real world. Most blockchain applications, especially in DeFi projects, rely on certain information to trigger the execution of smart contracts, but only on-chain information can be reached by smart contracts because blockchain systems are isolated from the outside world to guarantee safety. They act as on-chain APIs you can query to get information into your smart contracts. For example, an oracle might report the price of ETH/USD on Binance or the 2021 NBA Champions. Oracles can also be bi-directional, used to "send" data out to the real world.

**Flash Loan.** Flash loans are a type of under-collateralized loan that has become popular in decentralized finance (DeFi), especially on Ethereum. In the last three years, flash loan exploits have been used to attack vulnerable DeFi protocols and steal hundreds of millions of dollars. Some of the important use cases of flash loans are Arbitrage, Wash Trading, Collateral Swapping, and Self-Liquidation.

**Decentralized Lending Platform.** Lending platforms are smart contracts that allow users to borrow and lend crypto-assets at an interest rate. The borrowed assets come from a pool in which the creditors have deposited their investments. If this pool suffers a loss due to bad debt, the loss is distributed among these creditors. Due to the inability of these platforms to take action against loan defaulters (borrowers are just public keys on a blockchain), they use over-collateralization to keep the platform liquid. These Decentralized Finance (DeFi) platforms allow people to lend or borrow funds from others, speculate on price movements on assets using derivatives, trade cryptocurrencies, insure against risks, and earn interest in savings-like accounts.

**Automated Market Maker based Decentralized Exchanges (AMM-DEX).** Most liquidity pool-based DEXs use AMMs which pre-defines asset prices algorithmically. AMM is one of the most innovative inventions of DeFi as it achieves high efficiency. The majority of DEXs are AMM-based like Uniswap, Curve, Balancer, Bancor, TerraSwap and Raydium. The functionality of an AMM depends upon a *conservation function* which encodes a desired invariant property of the system. Generally, these AMM-DEXs provide a price-oracle service to any other third party, like DLPs.

## 4 STANDARD DLP MODEL

In this section, we introduce a formal model of a general DLP. We start by introducing the current security notions and security control measures used by a general DLP. Then we discuss the model used by DLPs focusing on the current best security practices. We then explain the liquidation threshold model and provide two equations used by a DLP for security threshold calculations. These

equations help us calculate a crucial system parameter $\epsilon'$, i.e. Safe collateralization ratio, which we use in SecDeLP in Section 6.

## 4.1 Security Model of a standard DLP

There are two important security concepts used when modeling a general secure DLP. They are explained below.

**Collaterilzation Bound and Liquidation.** A DLP design assumes that loans are secured by collateral: liquidations thereof are incentivized in order to recover loans should the borrowing users fail to repay. However, collateral liquidation is exposed to risks. Firstly, the incentive to liquidate is only effective, if the liquidated value of seized collateral is higher than the value of the repaid loan amount, implying a profit. Secondly, large fluctuations in asset prices may reduce the relative value of the collateral such that the loan becomes partially unrecoverable. Furthermore, an attacker with the ability to update token prices can force users to become undercollateralized and then seize the collateral of victims.

**Safe Collaterlization.** A DLP is $\epsilon$-collateralization safe when the ratio of the loan value of under-collateralized loans to the total loan value of the lending pool is below the threshold $\epsilon$. If the liquidation incentive is effective, a value below $\epsilon$ should not persist, as users are quick to execute liquidations. The efficiency of lending pool liquidations has been studied in [4]. Strong price volatility is a risk to $\epsilon$-collateralization safety, as a sharp drop in price can immediately reduce a previously sufficiently collateralized user to become undercollateralized below a defined threshold. Such an immediate drop leaves the user with no opportunity to maintain its collateral with repayments. This model prevents an adversary from liquidating DLP under extreme market conditions and other exploits. The degree to which this model is secure can be quantified by considering several factors that are explained below.

## 4.2 General DLP Security Factors

**Overview.** The security model for liquidation thresholds in decentralized lending pools typically involves multiple layers of protection to minimize the risk of manipulation or exploitation. Smart contracts are used to automate the liquidation process and ensure that it occurs according to predetermined rules and without the need for human intervention, which reduces the risk of fraud or manipulation. Another key security measure is the use of oracles, which are used to calculate the collateralization ratio and the liquidation threshold, and they ensure that the liquidation process is triggered only when the collateral value falls below a certain threshold.

**Liquidation calculation factors.** The liquidation threshold for a decentralized lending pool that uses over-collateralization on loans is typically determined by several factors, including:

- Collateralization Ratio: The collateralization ratio is the ratio of the value of the collateral to the value of the loan. The higher the collateralization ratio, the lower the risk of default, and therefore the lower the liquidation threshold.
- Volatility of the Collateral Asset: The more volatile the collateral asset, the higher the risk of its value falling below the loan value, and therefore the lower the liquidation threshold.
- Loan Duration: The longer the loan duration, the higher the risk of the collateral asset's value changing, and therefore the lower the liquidation threshold.
- Pool Reserves: The amount of reserves held in the lending pool can also affect the liquidation threshold. If the pool has a higher reserve ratio, it can absorb more losses before needing to liquidate loans, and therefore the liquidation threshold can be set higher.
- Market Conditions: Market conditions, such as changes in interest rates, liquidity, or demand for the collateral asset, can also affect the liquidation threshold.

Intuitively, the liquidation threshold is a balance between minimizing the risk of default and maximizing the pool's profitability. Decentralized lending platforms often use complex algorithms to calculate the liquidation threshold based on these factors, as well as other relevant data points.

## 4.3 Liquidation Threshold Calculation

The algorithm used for calculating the liquidation threshold in decentralized lending pools varies depending on the platform and its specific features, but there are some common approaches.

One common algorithm used for calculating the liquidation threshold is the "safe" or "minimum collateralization ratio" approach. In this approach, a safe collateralization ratio is established based on the volatility of the collateral asset and the desired level of risk for the lending pool. The liquidation threshold is then set at a level that ensures that the collateralization ratio does not fall below the safe level. For example, if the safe collateralization ratio is 150% and the loan value is $\$1,000$, the liquidation threshold would be set at $\$666.67$ (i.e., $1/150\%$ of $\$1,000$).

Another common algorithm used for calculating the liquidation threshold is the "dynamic" approach, which adjusts the liquidation threshold based on changes in the value of the collateral asset. In this approach, the liquidation threshold is initially set at a safe level, but it is then adjusted downward as the value of the collateral asset declines. For example, if the safe collateralization ratio is 150% and the value of the collateral asset falls by 20%, the liquidation threshold would be adjusted downward by 20% to maintain the safe collateralization ratio.

In addition to these approaches, some platforms may use more complex algorithms that take into account factors such as the loan duration, the market conditions, and the pool reserves to calculate the liquidation threshold. It's worth noting that the specific algorithm used for calculating the liquidation threshold can have a significant impact on the risk profile of the lending pool, so it's important to carefully evaluate the algorithm and the underlying assumptions before participating in the pool.

## 4.4 $\epsilon$-collateralization-safe DLP model

The state of a DLP can be described mathematically using a set of variables and equations that represent the various parameters and interactions within the platform. Table 1 provides a description of the state parameters required to calculate the liquidity threshold.

We define two equations to model the state of the DLP which can be used to determine the security thresholds. Table 2 provides the summarized equations. The purpose of each equation is stated below.

**Table 1: Liquidation Threshold Parameters**

| Notation | Meaning |
|---|---|
| $S$ | Total value of the pool's reserves (in some currency, e.g., USD) |
| $T$ | Total value of outstanding loans |
| $\epsilon'$ | Total collateralization ratio (as a decimal, e.g., 1.5 for 150%) |
| $C$ | Total value of collateral assets held by borrowers |

| Name | Equation |
|---|---|
| Total collateralization ratio | $C/S \leq 1/\epsilon'$ |
| Loan-to-value ratio | $T/C \leq 1/\epsilon'$ |

**Table 2: Security threshold equations**

- **Total collateralization ratio.** It is a measure of the total value of the collateral held compared to the total value of the pool reserves. This equation ensures that the total value of collateral assets held by borrowers is greater than or equal to the safe collateralization ratio.
- **Loan-to-value ratio.** It is a measure of the total value of outstanding loans ($T$) to the total value of collateral assets held by borrowers. This equation ensures that the total value of outstanding loans is less than or equal to the maximum loan-to-value ratio allowed by the safe collateralization ratio.

These equations can be used to monitor the state of the platform and ensure that it remains solvent and secure. If any of these equations is violated, it may indicate that the platform is at risk of default or insolvency, and corrective measures may need to be taken. With this, we now define a $\epsilon$-collateralization-safe DLP below.

*Definition 4.1 ($\epsilon$-collateralization-safe DLP).* A DLP $\mathbb{D}$ is $\epsilon$ collateralization safe if the total collateralization ratio and loan-to-value ratio are less than or equal to the inverse of collateralization ratio $\epsilon$, i.e.

$$C/S \leq 1/\epsilon$$

and,

$$T/C \leq 1/\epsilon.$$

## 5 ADVERSARY MODEL

In this section, we define a basic model of blockchains and the adversary. We then model a general oracle manipulation attack and provide the formal attack steps.

### 5.1 Basic Blockchain and Adversary Model

We model the interaction between users and the blockchain as a state transition system. We assume one computationally bounded and economically rational adversary $\mathbb{A}$. The adversary is not required to provide its own collateral to perform the attacks described below. We assume that the adversary is financially capable to pay any transaction fees associated with the network. Now, consider $\mathbb{A}$ (or equivalent) creates a set $M$ of $m$ malicious smart contracts. It is reasonable to assume that the adversary $\mathbb{A}$ and all $m$ malicious smart contracts can interact with each other according to the adversary's wish. Since smart contracts can be programmed to create other smart contracts from them, the set $M$ also contains all the smart contracts created by other malicious smart contracts at any point in time as well.

| Notation | Meaning |
|---|---|
| $\mathrm{S}_i$ | $i^{th}$ state of the blockchain |
| $\mathcal{T}_i$ | $i^{th}$ sub-transaction; $\mathcal{T}_i : \mathrm{S}_{i-1} \rightarrow \mathrm{S}_i$ |
| $\mathcal{T}$ | Attack transaction: $(\mathcal{T}_1, \cdots, \mathcal{T}_n)$ |
| $n$ | Number of sub-transactions in $\mathcal{T}$ |
| $\tau_i$ | $i^{th}$ transaction-step |

**Table 3: Adversary Model Notations**

| Notation | Meaning |
|---|---|
| $\mathcal{F}(X, \mathbf{A})$ | Take \$X worth of flash loan in asset **A** |
| $\mathcal{S}(\mathbf{A}, \mathbf{B})$ | Swap: Send asset **A** to get **B** |
| $\mathcal{D}(\mathbf{A}, \mathbf{cA})$ | Deposit **A** as collateral to get **cA** |
| $\mathcal{B}(\mathbf{cA}, \mathbf{C})$ | Borrow **C** using **cA** |
| $\mathcal{P}(X, \mathbf{A})$ | Repay \$X flash loan in **A** |

**Table 4: Transaction-steps ($\tau_i$)**

**Notations.** Let the initial state of the blockchain be denoted by $\mathrm{S}_0$. To execute an attack, $\mathbb{A}$ initiates a transaction $\mathcal{T}$ by calling a malicious smart contract belonging to $M$. A *transaction* is defined as an indexed-sequence of $n$ atomic state-transition-functions, i.e. $\mathcal{T} = (\mathcal{T}_1, \cdots, \mathcal{T}_n)$. *Atomic state transitions* are indivisible and irreducible series of state-change operations on the blockchain such that either all occurs, or nothing occurs. Here, $\mathcal{T}_i : \mathrm{S}_{i-1} \rightarrow \mathrm{S}_i$ represents an atomic state transition function, i.e. it is a definite series of operations on the blockchain state. Let $\mathcal{T}_i$ be referred to as $i^{th}$ *sub-transaction*. The state of the blockchain after transaction $\mathcal{T}$ is $\mathrm{S}_n$, i.e. $\mathcal{T} : \mathrm{S}_0 \rightarrow \mathrm{S}_n$. Now consider an equivalent k-indexed-sequence $\tau = (\tau_1, \tau_2, \cdots, \tau_k)$ of the transaction $\mathcal{T}$, i.e. $\tau : \mathrm{S}_0 \rightarrow \mathrm{S}_n$. Here, $\tau_i$ is an indexed sequence of sub-transactions (or atomic state transition functions), i.e. $\tau_i = (\mathcal{T}_x, \cdots, \mathcal{T}_y)$ where $0 < x < y \leq n$ and $x, y \in \mathbb{Z}^+$. Trivially, $\tau$ must be mutually exclusive, topologically sorted, and completely exhaustive for a given transaction $\mathcal{T}$. $\mathrm{S}_y$ denotes the state of the blockchain after the transaction step $\tau_i$, i.e. $\mathrm{S}_y = \tau_i(\mathrm{S}_{x-1})$ where $i < x < y \leq n$. Notations used in the basic adversary model are summarized in Table 3.

**Transaction-steps.** Table 4 summarizes a short description of the crucial transaction steps that are necessary for the oracle manipulation attack. $\mathcal{D}(\mathbf{A}, \mathbf{cA})$ is the step in which a borrower deposits an asset **A** as collateral on a DLP, $\mathbb{D}$. The depositor gets **cA**, which is an asset (usually created by the DLP) that represents the deposited amount of $A$ as collateral on $\mathbb{D}$. Here '**c**' in **cA** denotes a

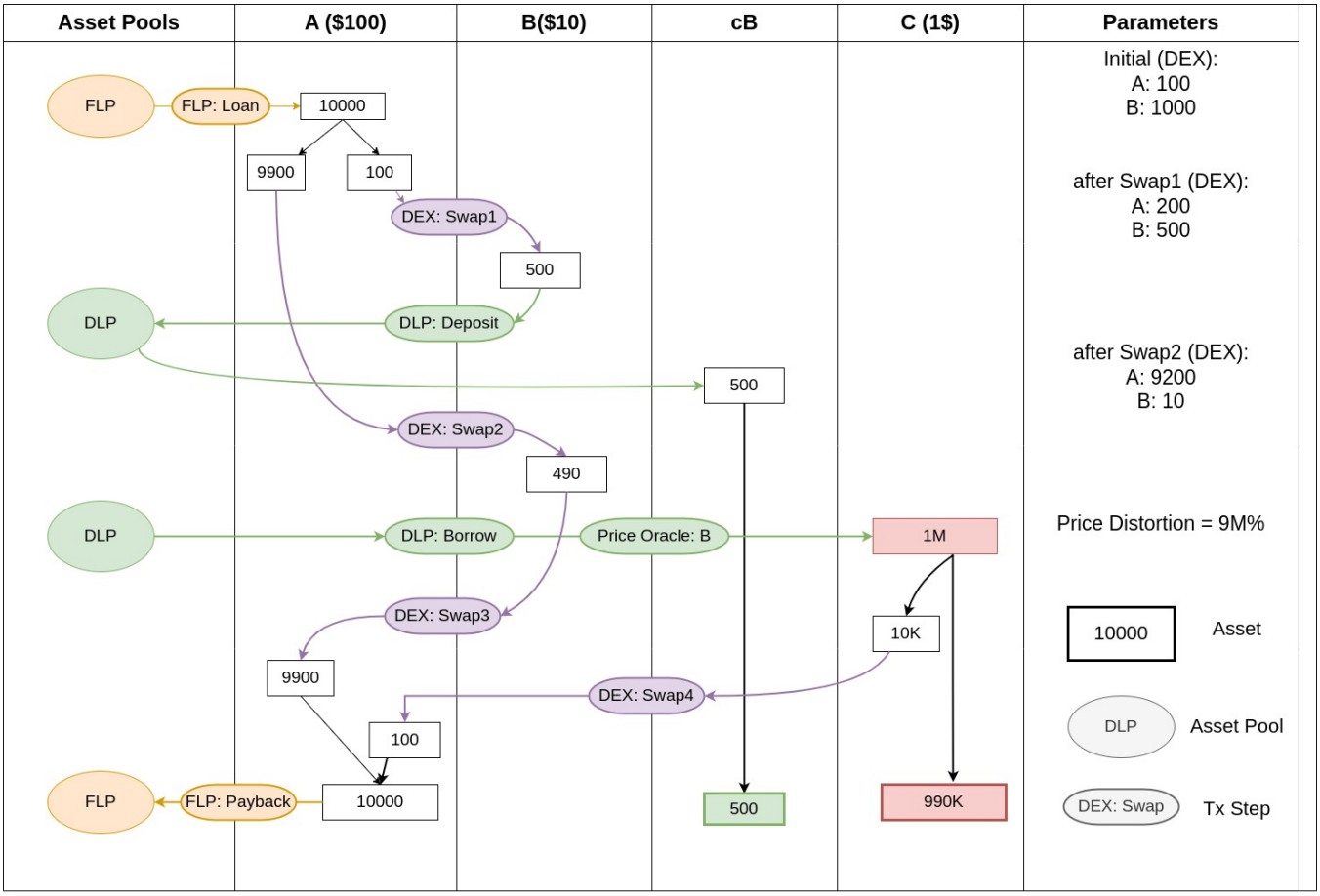


**Figure 1: Oracle Manipulation Attack Example**

*collateralized-asset*. The next required transaction step in topological order is $\mathcal{B}(\mathbf{cA}, \mathbf{C})$. In this transaction step, the adversary (as the borrower) takes out a loan in asset $\mathbf{C}$. This asset could potentially be any asset available on the DLP $\mathbb{D}$ for users to borrow. In this step, $\mathbb{A}$ must use a collateralized asset provided by the DLP, in this case, $\mathbf{cA}$. The adversary can then borrow an asset $\mathbf{C}$ within the borrowing limit available to her. This borrow limit is determined by two factors, which are *(i)* the value of the deposited collateral $\mathbf{cA}$, and *(ii)* $\epsilon$, the collateralization ratio set by the DLP $\mathbb{D}$. Typically $\epsilon \in [1.25, 5]$ for a general DLP.

## 5.2 Understanding Oracle Manipulation Attack

Oracle manipulation attacks using flash loans can be performed out on lending platforms that use oracles to determine the value of the collateralized assets used to secure loans. In this type of attack, the attacker can manipulate the price oracle of a collateralized asset by using flash loans. This manipulation is done by executing a large trade on a DEX that provides the oracle used by the DLP. The attacker can use the flash loan to borrow a large amount of cryptocurrency that is used as collateral for loans on the lending platform. The attacker then uses the borrowed cryptocurrency to execute trades on a DEX that relies on the same oracle used by the lending platform. By executing a large number of trades in a short period of time, the attacker can manipulate the price of the collateral, causing it to appear more or less valuable than it actually is. Once the price of the collateral has been manipulated, the attacker can use it to secure a loan on the lending platform, borrowing a larger amount of funds than would have been possible without the price manipulation. The attacker can then use these funds to execute profitable trades on the DEX, profiting from the price difference.

**Formal attack steps.** Consider a DLP $\mathbb{D}$ using a price oracle $\mathbb{P}_{\mathbf{A}}$ provided by a Decentralized Exchange $\mathbb{DEX}$ to determine the price of an asset $\mathbf{A}$. The adversary $\mathbb{A}$ initiates a transaction $\mathcal{T}$, with a sequence of $k$ transaction steps. The reasoning and the critical steps that are necessary for the attack are described below in topological order:

- $\mathcal{F}$ $(X, \mathbf{A})$; Take $\$X$ worth of flash loan in asset $\mathbf{A}$ as $\tau_1$.
- $\mathcal{S}$ $(\mathbf{A}, \mathbf{B})$; This transaction-step must use a small fraction of the total $\$X$ worth of flash loan.
- $\mathcal{D}$ $(\mathbf{B}, \mathbf{cB})$; $\mathbb{A}$ plans to manipulate the price of $\mathbf{B}$ by distorting the price of the pair $\mathbf{A}/\mathbf{B}$ on $\mathbb{DEX}$, which is used in the price oracle $\mathbb{P}_{\mathbf{B}}$.

- $\mathcal{S}$ $(\mathbf{A}, \mathbf{B})$; $\mathbb{A}$ manipulates/distorts the price of the asset pair $\mathbf{A}/\mathbf{B}$. This price is then instantly fetched by $\mathbb{D}$ as the price oracle $\mathbb{P}_{\mathbf{B}}$ which then directly determines the price of **cB** (linearly proportional). Note that the majority of flash loan amount goes in this swap.
- $\mathcal{B}$ $(\mathbf{cB}, \mathbf{C})$; $\mathbb{A}$ redeems the collateral **cB** (and other equivalents) at the liquidated/distorted price to borrow an asset **C** with $\epsilon$ as collateralization ratio. Note that $\mathbb{A}$ can only gain a maximum profit of $G = \$X \cdot ((1+\theta) \cdot \epsilon - 1)$, where $\theta$ is the fractional price distortion.
- $\mathcal{S}$ $(\mathbf{B}, \mathbf{A})$; This step is basically reverting the swap in which adversary manipulated the price, getting back the flash loan amount $X$ to payback the loan.
- $\mathcal{P}$ $(X, \mathbf{A})$; Payback the flash loan in asset **A**.

**Attack Example.** Figure 1 shows an illustration of a simple Oracle Manipulation Attack on a DLP. Considering the price of assets **A**, **B**, and **C** to be \$100, \$10, and \$1, respectively. Here, the timeline of transaction steps are shown from top to bottom. The parameters shown on the right are the amount of liquidity reserves of asset pool $\mathbf{A}/\mathbf{B}$ on the Decentralized Exchange (DEX). Here, the adversary must use a small fraction of the flash loan amount in 'Swap1', to ensure that it has enough funds remaining to distort the oracle price. The total liquidity available for the asset **B** must be close to this value to maximize returns on a successful attack. In the end, the adversary gains $990K$ amount of asset C shown in red. Here the price of B relative to A is provided by the DEX as an oracle. This price is determined by the amount of liquidity present in the DEX's pool. Here, price of B is directly proportional to reserve of B over reserve of A. Notice that, in this example, the adversary distorts the price by 9.2 Million percentage or $9.2 \times 10^4$ times. The collateral provided to the DLP, 500**cB**, remains there as shown in green. This amount is negligible compared to the profit of the adversary. Hence, it can be ignored and be left as a locked collateral in the DLP.

It is worth to note that flash loan providers might require a small fraction of fee to use their service. Typically this fee is less than 0.1% of the total value of flash loan borrowed. Considering that this fraction is negligible to the price-distortion and adversaries' Return-On-Investment (ROI), this fee is neglected for simplicity.

# 6 SECDELP : SECURE DLP SOLUTION

In this section, we present our security solution, SecDeLP . We first provide some important prerequisite concepts in DLP security. Then we give an overview of SecDeLP with the some practical requirements to enable SecDeLP . We then proceed by presenting the SecDeLP algorithm. First, we present a proposition, then define four lemmas, and then use these lemmas to provide a Theorem. This theorem states that SecDeLP is secure against oracle manipulation attacks.

## 6.1 Prerequisites

Before explaining our solution, we first need to look at two prerequisite concepts to understand SecDeLP . These concepts must be applied to ensure the security of a DLP.

*Definition 6.1 (Availability).* It refers to the degree to which a system or resource is accessible and usable by authorized users when they need it.

Availability means that the system is operational and can be used to perform the functions for which it was designed, and that data and other resources are available for authorized use. It is one of the three key pillars of information security, along with confidentiality and integrity. The most common metric is *uptime percentage*, which represents the portion of time the service is operational during the measurement period.

*Definition 6.2 (Price Conservation).* The mechanism that is employed in DLP which ensures that for any transaction, the value removed in one asset equals the value added in the other asset.

Price conservation is an important property in Automated Market Maker (AMM) based Decentralized Exchanges (DEXs) that helps to ensure the stability and accuracy of the price of assets traded on the exchange. Price conservation is maintained in AMM-based DEXs through the concept of constant product formula, which is a mathematical formula used to calculate the price of assets in the liquidity pool. The formula ensures that the product of the number of tokens in each asset's balance remains constant, which helps to maintain the price stability of the assets.

**Price conservation example.** For example, in a simple AMM-based DEX like Uniswap, if the balance of ETH and USDT in a liquidity pool is 10 ETH and 10000 USDT respectively, then the product of these two balances must always remain constant, i.e., 10 ETH $\times$ 10000 USDT = 100000. If a trader buys 1 ETH, the balance of ETH decreases to 9, and the balance of USDT increases to 11111.11, so that the constant product of the two balances remains 100000. Price conservation ensures that the price of assets is accurate and reflects the actual supply and demand of the market, which is crucial for fair and efficient trading.

## 6.2 Overview and assumptions

**Overview.** The algorithmic solution is to add the following to the DLP architecture: (i) store the last used price of all assets, and (ii) store the block index in which the price was last updated. Then, only update this price *once-per-block* based on certain conditions, ***C1***, ***C2***, and ***C3***. Once updated, use the price oracle as intended only if ***C4*** and ***C5*** conditions are true. ***C1*** ensures that the price is updated once-per-block. ***C2*** ensures that the adversary is unable to distort the price by more than a certain threshold in a single block, or equivalently, in the malicious transaction $\mathcal{T}$. ***C3*** ensures that all legitimate transactions are executed and only the attack is prevented. Finally, the conditions imposed on using the price oracle, i.e. ***C4*** and ***C5*** prevents the attacker from reconfiguring the price-distortion transaction-step into any other equivalent transaction-step. The above-mentioned conditions ensure that SecDeLP $\mathbb{D}$ is secure against oracle manipulation attacks performed using flash loans.

**Practical requirements.** The price of all assets is constantly changing for each transaction on the blockchain. The approximate time to add a new block to the blockchain, or *block-time*, is determined by the specific blockchain protocol. Consider $\alpha_{\mathbf{A}}$ to be the acceptable fractional change in the price of asset **A** per block. We require that the price state of all assets must be updated at least every $T$ blocks on the blockchain, where $T \in \mathbb{Z}^+$. Hence, both $T$ and $\alpha$ must directly

| Notation | Meaning |
|---|---|
| P | State that stores the last updated block index and the price of an asset |
| B.id | Index of the current block for any $\tau_i$ |
| $\epsilon$ | Collateralization ratio set by $\mathbb{D}$ |
| $\alpha_{\mathbf{A}}$ | Acceptable fractional price distortion of asset **A** per block |
| $T$ | Maximum number of blocks after the price of any asset is updated |
| $\theta$ | Maximum allowed price-distortion of an asset in one block |
| $\mathbb{P}_{\mathbf{A}}$ | Price of asset **A** provided by the oracle |

**Table 5: SecDeLP Algorithm Notations**

**Algorithm 1** SecDeLP Algorithm

**Input:** $\alpha_{\mathbf{A}}, \theta, T$ ▷ Acceptable fractional change of asset **A** in one block, maximum allowed fractional change in price, maximum number of blocks before P is updated

**Output:** $Use(\mathbb{P}_{\mathbf{A}})$ ▷ $True$ if price oracle is safe to use else it is $False$

1: **if** $Rec(\mathbb{P}_{\mathbf{A}})$ **then** ▷ $Rec(\mathbb{P}_{\mathbf{A}}) = True$ if price oracle is recieved, else it is $False$
2: **if** P.id == B.id $- t$ & $|1 - \mathbb{P}_{\mathbf{A}}/\text{P.A}| < \theta$ & $\alpha_A \cdot t < \theta$ **then** ▷ $t \in [1..T]$
P $\leftarrow <$ B.id, $\mathbb{P}_{\mathbf{A}} >$
3: **end if**
4: **if** P.id == B.id & $|1 - \mathbb{P}_{\mathbf{A}}/\text{P.A}| < \alpha_{\mathbf{A}}$ **then**
$Use(\mathbb{P}_{\mathbf{A}}) \leftarrow True$
5: **end if**
6: **end if**

depend on the block-time[1], and should be determined accordingly. Later, in Section 7, we illustrate practical values for these parameters based on the past three years of market data. We also show that these requirements are practically feasible and cost-efficient. The notations used in SecDeLP are summarized in Table 5.

## 6.3 SecDeLP Algorithm

The algorithmic solution for $\mathbb{D}$ to prevent this attack is given in Algorithm 1. It is summarized as follows:

(1) Store a state P $=<$ id, A $>$ for each asset **A**. P.A stores the price of **A**, and P.id stores the index of the block in which P was last updated.
(2) Input the values of $\alpha$ for each asset available on the DLP. Here $\alpha_{\mathbf{A}}$denotes the acceptable price distortion of asset **A** per block.
(3) Input the value of $T$, which denotes the maximum number of blocks before a price state P must be updated.
(4) Let $\mathbb{P}_{\mathbf{A}}$ denote the oracle's price of asset **A**. Let B.id denote the index of the current block. And, let $\theta = \epsilon - 1$ where $\epsilon$ is the collateralization ratio set by $\mathbb{D}$. Now, consider an integer variable $t \in [1..T]$.
(5) When $\mathbb{P}_{\mathbf{A}}$ is received, update the state P if and only if:
***C1*** P.id = B.id $- t$,
***C2*** $|1 - \mathbb{P}_{\mathbf{A}}/\text{P.A}| < \theta$, and
***C3*** $\alpha_{\mathbf{A}} \cdot t < \theta$.
(6) Once P is updated, use $\mathbb{P}_{\mathbf{A}}$ as intended if and only if
***C4*** P.id = B.id, and
***C5*** $|1 - \mathbb{P}_{\mathbf{A}}/\text{P.A}| < \alpha_{\mathbf{A}}$.

**SecDeLP additional benefits.** SecDeLP takes into account the system parameter $\epsilon$ which can be chosen accordingly for each different collateralized asset. This ability provides the platform developers greater control over $\theta$ for different assets. Specifically, users will be able to quantify the security, i.e. risk ($\theta$) and availability ($T$ and $\alpha$), associated with each asset separately. There is also an additional benefit in creating a design for the lending platform in contrast to any applying any solutions on the decentralized exchange or the oracle source. It gives the ability to securely use multiple sources of oracles without having to worrying about security issues on the oracle source's side. The design makes the lending platform more modular, more inclusive of assets with smaller market capitalization, and better at risk analysis and control over security assumptions regardless of the source of price oracle.

## 6.4 SecDeLP Theorem

In this subsection, we provide a theorem to show that SecDeLP is secure against oracle manipulation attacks. First, we provide a definition of a secure DLP below.

*Definition 6.3 (Secure DLP).* A SecDeLP $\mathbb{D}$ is secure against oracle manipulation attacks performed using flash loans if the profit $G$ gained by an adversary $\mathbb{A}$ from a transaction $\mathcal{T}$ is negative i.e. $G < 0$ or the attack transaction $\mathcal{T}$ is unsuccessful.

Now, we state a proposition to prove that SecDeLP ensures that certain conditions hold true for a DLP. We later show that these conditions are necessary to secure a DLP from oracle manipulation attacks performed using flash loans.

Proposition 6.4. *SecDeLP ensures that: (**C1**) the price state of an asset is updated at most once per block; (**C2**) the adversary can not pay back the flash loan in the transaction $\mathcal{T}$; (**C3**) the price change in all assets on the DLP is conserved over the last $T$ blocks; (**C4** and **C5**) and the adversary is unable to reconfigure the price-distortion transaction-step into any other equivalent transaction-step.*

To prove this proposition, we now define four lemmas and consequently prove them.

Lemma 6.5. *The condition **C1** ensures that the price state* P *of any asset is updated at most once per block in the blockchain.*

Proof. According to the algorithm, the price state P is updated if and only if three conditions are satisfied, that is if ***C1***, ***C2***, and ***C3***

[1]The average time it takes to add one block on the blockchain

are true. All transactions in the same block must share the same index B.id. Consider a block B consisiting of $n$ transactions, i.e B = $(\mathcal{T}_1, \cdots, \mathcal{T}_n)$. Now trivially,

$$\forall i, j \in [1..n], \mathcal{T}_i.\text{id} = \mathcal{T}_j.\text{id} = \text{B.id} \tag{1}$$

where, $\mathcal{T}_i$.id denotes the index of the block stored in the transaction $\mathcal{T}_i$ in that block.

Due to the constraint $t \geq 1$, the price state P will only be updated when the index of the current block is greater than the index of the block where it was last updated, i.e.,

$$\text{P.id} < \text{B.id}. \tag{2}$$

The constraints in equations 1 and 2 ensure that the price state P will not be updated more than once for any subsequent transaction in the same block. Therefore, ***C1*** ensures that the price state P is updated at most once for a single block. Hence proved. □

Lemma 6.6. *The condition **C2** ensures that the adversary $\mathbb{A}$ can not pay back the flash loan.*

Proof. Let the amount of the flash loan be $\$X$. For a price distortion of a fraction $\theta'$, $\mathbb{A}$ can gain a maximum profit $G$, where

$$G = \$((1 + \theta')/\epsilon - 1) \cdot X.$$

Here, $\epsilon$ is the safe collateralization ratio. It is trivial that considering the upper bound on $G$ to be the flash loan amount, i.e. $\$X$, makes it infeasible for the adversary to pay back the loan. Hence, we get the upper-bound on $\theta$, i.e.

$$\theta' <= \epsilon - 1.$$

In SecDeLP , we take $\theta = \epsilon - 1$, where $\theta$ represents the upper bound on the price-distortion in one block. Hence, we get the condition ***C2*** i.e.

$$\left|1 - \frac{\mathbb{P}_{\mathbf{A}}}{\text{P.A}}\right| < \theta,$$

which ensures that price cannot be manipulated more than or equal to $\theta \cdot \text{P}_{\mathbf{A}}$ in a single block. Therefore, the adversary cannot distort the price to gain enough profit to pay back the flash loan in the same transaction $\mathcal{T}$. Hence proved.

□

Lemma 6.7. *The condition (**C3**) ensures that the price change over $T$ blocks is conserved for legitimate transactions.*

Proof. The condition ***C3*** illustrates that the average maximum allowed distortion per block over the last $t$ blocks is upper-bounded by the maximum distortion allowed in the current block, i.e.

$$\alpha_{\mathbf{A}} \cdot t < \theta.$$

Since the price state was updated exactly $t$ blocks prior to the current block and $\alpha_{\mathbf{A}}$ is the acceptable distortion per block, the price change over time is conserved for legitimate transactions. Hence, proved. □

Lemma 6.8. *The conditions **C4** and **C5** ensure that the adversary $\mathbb{A}$ can not re-configure the price distortion transaction step into any other equivalent transaction-step.*

| $\epsilon$ | $\theta$ (%) | $\alpha^5$ | $\alpha^{10}$ | $\alpha^{15}$ | $\alpha^{20}$ | $\alpha^{25}$ (%) |
|---|---|---|---|---|---|---|
| 5.00 | 400 | 80.0 | 40.0 | 26.667 | 20.0 | 16.0 |
| 2.50 | 150 | 30.0 | 15.0 | 10.0 | 7.5 | 6.0 |
| 1.67 | 67 | 13.2 | 6.6 | 4.4 | 3.3 | 2.64 |
| 1.42 | 42 | 8.4 | 4.2 | 2.8 | 2.1 | 1.68 |
| 1.25 | 25 | 5.0 | 2.5 | 1.667 | 1.25 | 1.0 |

**Table 6: Illustration of the safe upper-bound values of $\alpha$ for different values of $\epsilon$ and $T$. Calculated using *C3*: $\alpha \times T < \theta$.**

Proof. We know from Lemma 6.5 that ***C1*** ensures that price state P can only be updated once per block. Once updated, the condition ***C4***, i.e.

$$\text{P.id} = \text{B.id},$$

trivially ensures that the price oracle $\mathbb{P}$ is only used if it is received in the current block. Next, the condition ***C5***, i.e.

$$\left|1 - \frac{\mathbb{P}_{\mathbf{A}}}{\text{P.A}}\right| < \alpha_{\mathbf{A}},$$

ensures that the price received by the oracle is within the acceptable price distortion of a single block, that is $\alpha_{\mathbf{A}}$. Trivially, any subsequent sub-transactions can only distort the original updated price to an upper-bound limit of $\alpha_{\mathbf{A}}$ due to ***C5***. Because the adversary $\mathbb{A}$ can not distort the original price by more than the acceptable price distortion of $\alpha_{\mathbb{A}}$ in a single transaction, it makes it impossible to reconfigure the transaction $\mathcal{T}$ and distort the price enough to pay back the loan, similar to Lemma 6.6. Since the loan can not be paid back, the transaction will not execute. Hence, it prevents the adversary from re-configuring the price-distortion step. Hence proved. □

Trivially, proposition 6.4 is true from lemmas 6.5, 6.6, 6.7, and 6.8. We now state the Theorem below.

Theorem 6.9. *A SecDeLP $\mathbb{D}$, using $\epsilon$ as collateralization ratio and $\mathbb{P}$ as a price oracle, assuming that the price for all assets is updated at least every $T$ blocks, and considering $\alpha$ to be the acceptable fractional change to a price in one block, $\mathbb{D}$ is secure according to definition 6.3.*

Proof. We know that ***C2*** ensures that the adversary cannot pay back the flash loan (from 6.6). This inability is accomplished by restraining the price distortion up to the minimum threshold required to gain a profit (same as Lemma 6.6) i.e., $G < 0$. We know from Lemma 6.7 that the price change on $\mathbb{D}$ is conserved, hence there is no value being lost from $\mathbb{D}$. This proves that the adversary cannot gain any profits from the transaction $\mathcal{T}$. Secondly, the transaction $\mathcal{T}$ is reverted in case the adversary is unable to pay back the flash loan. This reasoning shows that SecDeLP $\mathbb{D}$ is secure according to the Definition 6.3. Hence proved. □

## 6.5 Calculating the input parameters

In the SecDeLP algorithm, we require parameter inputs of $\alpha$ and $T$. These input values must be set by the DLP. To calculate these values, we first observe that the value of $T$ would impact the overall operational costs of the DLP. This impact could be in the case where the DLP has to manually update a price state. Now trivially, this value depends on the block time of the specific blockchain the DLP is operating. Later, in Section 7 we illustrate that a reasonable range

| Coin | Market Capitalization | $\bar{x}$ | $\sigma$ | Upper CI of $\theta'$ |
|---|---|---|---|---|
| BTC | $\$505B$ | 0.106 | 0.090 | 0.1069 |
| ETH | $\$210B$ | 0.149 | 0.128 | 0.1504 |
| BNB | $\$39B$ | 0.270 | 0.293 | 0.285 |
| XRP | $\$27B$ | 0.124 | 0.249 | 0.1767 |
| ADA | $\$9.7B$ | 0.166 | 0.175 | 0.1802 |
| SOL | $\$6.1B$ | 0.235 | 0.232 | 0.252 |
| DOT | $\$5.5B$ | 0.175 | 0.208 | 0.198 |
| ICP | $\$2B$ | 0.199 | 0.257 | 0.252 |
| MKR | $\$620M$ | 0.166 | 0.214 | 0.178 |
| ZRX | $\$148M$ | 0.202 | 0.284 | 0.211 |
| PERP | $\$27M$ | 0.166 | 0.194 | 0.188 |

**Table 7: Per-minute Price Change Statistics: $\theta'$**

of time to update a price state is from every one to five minutes. We later discuss the additional costs related to this parameter in Section 7.

Now we calculate the maximum allowed values of $\alpha$. First, note that because of the condition ***C3*** that is $\alpha \times t < \theta$, and the constraint $t \leq T$, $\alpha \times T < \theta$ must also be ensured at all times by the DLP. We consider the Ethereum platform for our calculations as it is the largest platform for DLPs today. On the Ethereum blockchain, the block-time is roughly 12 seconds, which means that every minute 5 new blocks are added on average. As mentioned above, we consider a minimum time to update a state in a range of one to five minutes. From this range, we get the range of $T$ as [5..25]. Now, we can calculate the maximum allowed values of $\alpha$ for this range of values of $T$ using ***C3***. Since $\theta = \epsilon - 1$, we consider a range of values of $\epsilon$ that is typically used by a standard DLP, i.e. $\epsilon \in [1.25, 5.0]$. We now illustrate the safe upper-bound values of *alpha* on the above-mentioned ranges of $T$ and $\epsilon$ in Table 6. These values represent the upper-bound on theoretically safe-to-use values as proven in Thereom 6.9.

In the next section, we show that these upper-bound values for $\alpha$ are well over a two-factor margin than what is required in the real-world setting. We also discuss the worst-case scenario for a DLP in terms of both operational cost and risk. The worst case for operational cost comes when the required $T$ is low (less than 5 blocks on Ethereum). The worst case in terms of security arises when the collateral ratio set by the DLP is low ($\epsilon < 1.25$). We show that our required values are within a secure range of values with a high degree of confidence.

# 7 SECDELP : PRACTICALITY ANALYSIS

In this section, we present the practical analysis of SecDeLP . The aim of this section is to show that the SecDeLP requirements are reasonable in a real-world setting. We first show that SecDeLP is secure with a high confidence value. We do this by providing an empirical analysis of the market data collected over the past three years. Next, we provide an analysis of additional operational costs incurred by using SecDeLP in a standard DLP.

## 7.1 Are SecDeLP requirements practical?

**Data Collection.** We collected the per-minute OHLCV (Open-High-Low-Close) price data of 12 crypto-assets from January 1st, 2020 to June 1st, 2023. This amounts to roughly 1.75 million data points for each crypto-asset. We choose a market-capitalization value ranging from \$25 Million (***PERP***) to \$500 Billion ***BTC***. This range accounts for over 99% of the total cryptocurrency market capitalization value. Table 7 provides some statistics on the market price data of various cryptocurrency assets. These assets are sorted based on their market capitalization value shown in USD. We start by calculating the fractional change in price per minute for each of these assets. We denote this value by $\theta'$. $\theta'$ denotes the fractional price change in the spot price of these assets, where $\theta' = \left|\frac{H-L}{O}\right|$) ($H$ is the high price, $L$ is the low price, and $O$ is the open price). Here, $\bar{x}$ shows the mean value of all $\theta'$, and $\sigma$ shows the standard deviation. We then use this data to calculate a confidence interval of $\theta'$.

**Empricial analysis.** We aim to show that the maximum allowed values of $\alpha$ illustrated in Table 6 are practically reasonable. First, we calculate the Confidence Interval (CI) of $\theta'$, i.e. CI of the per-minute price change of these assets. In frequent statistics, a confidence interval is a range of values that is likely to contain the value of an unknown population parameter. A confidence interval is the mean of your estimate plus and minus the variation in that estimate. The formula is given as follows:

$$CI = \bar{x} \pm z\frac{\sigma}{\sqrt{n}},$$

where $n$ is the total number of samples, and $z$ is the confidence value. Using $z = 1 - 0.10^{10}$, we calculate the Upper bound of the confidence interval on $\theta'$ for all the assets. This provides us with an upper-bound value for an acceptable price change per minute, with a high confidence value. Since $\alpha$ is the acceptable price distortion per block, we can take the average of this upper bound over the number of blocks per minute to get a lower bound on $\alpha$. Let $\alpha'$ denote this lower bound value, where $\alpha' = \theta' / \textit{blocks-per-minute}$.

Now we show that the lower-bound on $\alpha$, that is $\alpha'$ is much less than the upper-bound on $\alpha$ in the worst-case scenario. From a security and practical perspective, the worst-case scenario for the DLP is when the collateralization ratio is high and a price state update is required every minute. In this case, the upper-bound on

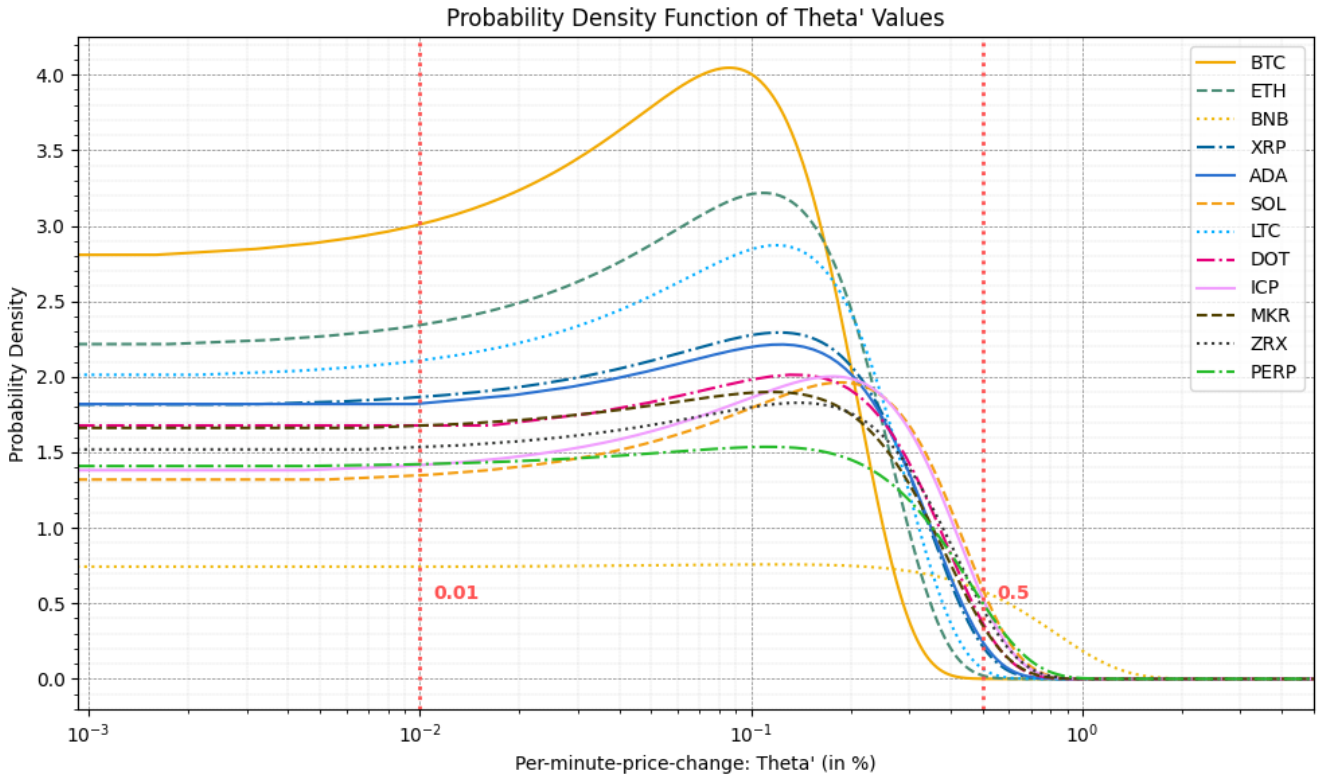


**Figure 2: Probability Density Function of $\theta'$ for 12 crypto-assets**

$\alpha$ would be when $\epsilon = 1.25$ and $T = 5$ (considering on Ethereum blockchain) that is $\alpha < 1\%$ (from Table 6). We can estimate the range of values of $\theta'$ from Figure 3, which illustrates the Cumulative Density Function (CDF) of $\theta'$ and Figure 2, which illustrates the Probability Density Function (PDF) of $\theta'$. We calculated these values using the historical data with a confidence value of $1 - 0.1^{10}$. Here, notice that most of the values of $\theta'$ are well within the range of $[0.01, 0.5]$. Further, the range of values of $\theta$ is a factor of two less than the upper-bound required on $\alpha$ even in the worst-case scenario. Considering Ethereum, if we calculate the value of $\alpha'$ from this range of $\theta'$, we get the range $[0.002, 0.1]$. This range is less than the upper-bound by at least a factor or one. Hence, this illustration shows that SecDeLP requirements on its input parameter of $\alpha$ are practical.

**Discussion.** Figure 4 shows an illustration of the confidence interval upper-bound on $\theta'$ using $z = 1 - 0.1^{10}$ vs the market capitalization of 12 crypto-assets. This figure shows that the market cap is not necessarily related to price volatility. Here we have a market capitalization value range of 4 orders, and yet we observe that $\theta' \in [0.1, 0.3]$ with high confidence. This is an interesting observation as one could consider investigating the factors that affect price volatility for a better understanding of this upper bound value of $\theta'$. Another point worth mentioning here is that the standard deviation and the mean values of $\theta'$ are all within a factor of one. This observation indicates that, even over a period of three years, and even with orders of magnitute of difference in market capitalization of various crypto-assets, the change in price falls within a range of $[0.01, 0.5]$ with high confidence.

## 7.2 SecDeLP additional cost analysis

**Cost of running a standard DLP.** The cost of running a DLP on Ethereum can be significant and can vary depending on a number of factors. Some of these factors are platform's size, complexity, and the technology stack used. It's important for developers and operators to carefully consider these factors before launching a platform and to stay informed about the current state of the Ethereum network to ensure that the platform remains secure, scalable, and cost-effective. Here are some potential costs that a DLP may face:

- **Development and maintenance costs.** This includes the cost of developing and maintaining the smart contracts, user interfaces, and other components of the platform. These costs can include expenses for tools and infrastructure.
- **Gas fees.** DLPs built on the Ethereum blockchain require transactions to be executed on the blockchain, which incurs gas fees. These fees can be substantial during periods of high network congestion and can be a significant ongoing cost for the platform.
- **Security costs**: As decentralized lending platforms handle significant amounts of value, they need to invest in security measures to protect against hacks and other security threats. This can include expenses for security audits, bug bounties, and insurance.

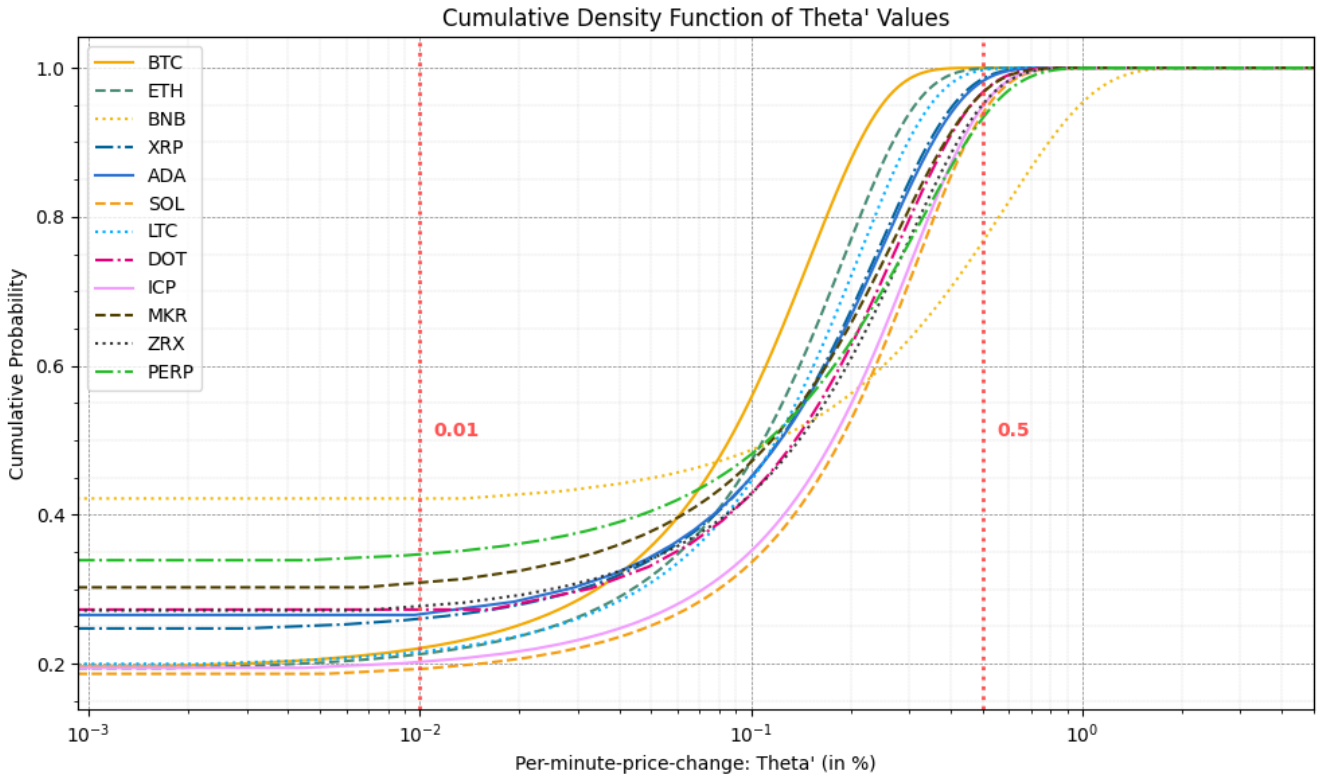


**Figure 3: Cumulative Density Function of $\theta'$ for 12 crypto-assets**

- **Compliance and Operation costs.** This includes regulation and operational expenses such as utilities, administrative staff, etc.

**SecDeLP additional cost.** The cost of operations for the Lending platform using SecDeLP includes minimal additional expenses compared to the costs mentioned above. The two major cost additions are *(i)* the cost of running a program to constantly monitor the prices of each asset on their platform, *(ii)* the cost of updating the state P every time an asset is updated later than $T$ blocks. The cost of running a program is reasonable with little to no extra expense. However, the cost of updating a state on the blockchain can be expensive. To analyze the cost, we first define the worst-case scenario i.e. the costliest possible case to maintain security.

In the worst-case scenario, there are no transactions on the DLP, and hence no price updates. In this case, the DLP would have to make sure to update the state P, i.e. price of every asset stored in the DLP smart contract, every $T$ blocks. This cost depends on the gas fee of the network. On the Ethereum blockchain, gas fees are paid in Ether, and can vary widely depending on network congestion and other factors. During periods of high demand, gas fees can become very expensive, which can increase the cost of running a decentralized lending platform. Hence, empirical analysis of the gas prices is useful in determining the operational costs for this scenario.

**Empirical Analysis.** Gas required to update one uint256 state variable on Ethereum is around $2 \times 10^4$. Total Transaction Gas used to update a state variable is within the range $[30000, 10^5]$. Gas Price range = $[10, 100]$ Gwei, where 1 Gwei is $10^{-9}$ ETH. The maximum cost to update the price for one asset is maximum gas used times the maximum gas price. This amount comes in the order of $10^7$ Gwei, or 0.01 ETH. At the current market price of around $1600, this value is less than $20. These costs should be feasible in any practical scenario.

## 8 DISCUSSION

Lending platform users should also be aware of the potential risks and exercise caution when using the platform. To defend against oracle manipulation attacks, lending platforms use several mechanisms to ensure the accuracy and integrity of their oracle data. However, each of these mechanics have their own limitations. These mechanisms include:

- **Multiple oracle sources.** Lending platforms can use multiple independent oracle sources to obtain pricing data for collateral assets, which reduces the risk of a single point of failure or manipulation. The platform can then use an algorithm to take a weighted average of the prices obtained from each oracle to arrive at a more accurate price. One limitation of using multiple oracle sources is that it can increase the complexity and cost of maintaining the platform. Each additional oracle source requires

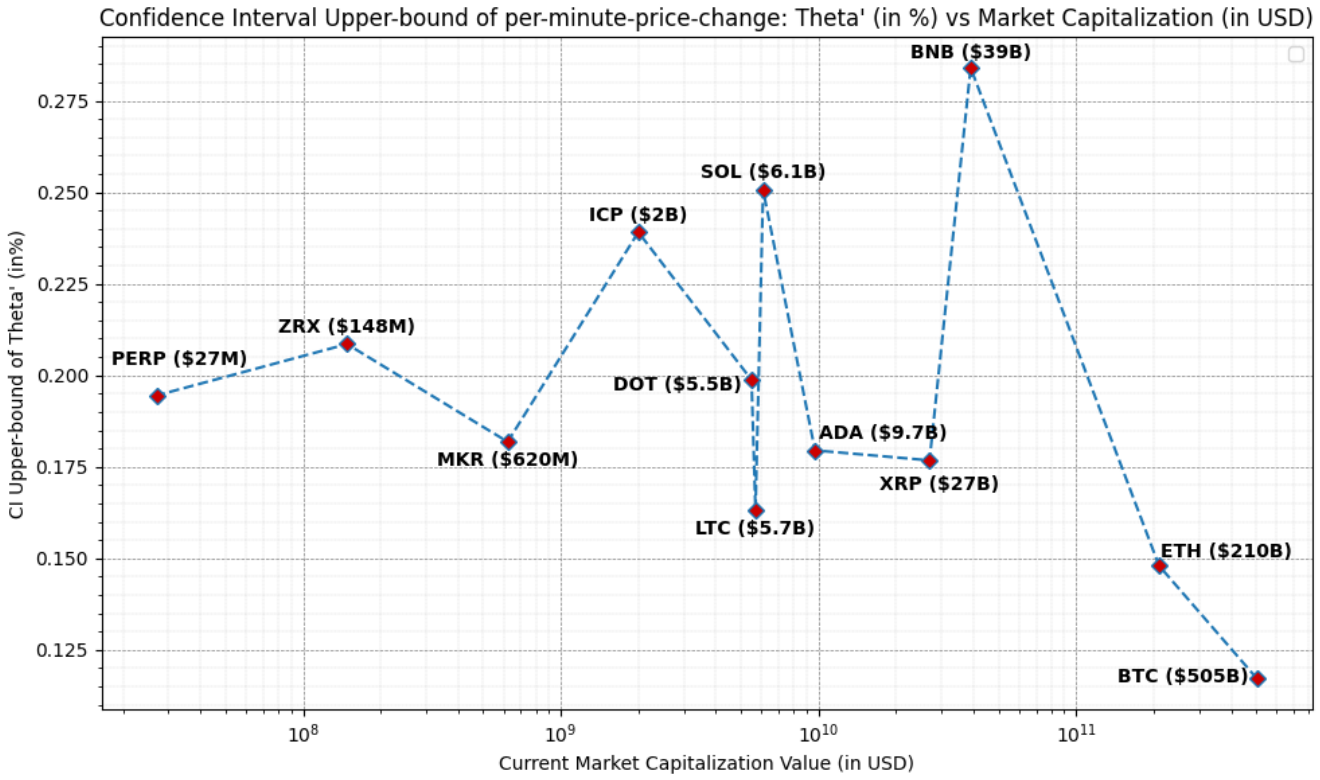


**Figure 4: Confidence Interval Upper bound of $\theta'$ (in %) vs Market Capitalization value (in USD)**

additional infrastructure and maintenance costs, which can be prohibitive for smaller platforms or those with limited resources. Moreover, if all of the oracle sources are compromised or manipulated in the same way, using multiple sources may not provide effective protection against oracle manipulation attacks.

- **Price feeds and averaging.** Lending platforms can also use price feeds and averaging mechanisms to smooth out any outliers or anomalies in the oracle data. For example, the platform can discard any prices that deviate significantly from the median, or use a moving average to smooth out short-term fluctuations in the market. A limitation of using price feeds and averaging mechanisms is that they can be susceptible to algorithmic or statistical attacks. For example, an attacker may be able to manipulate the median or moving average by introducing a large number of fake transactions or orders. Additionally, price feeds and averaging mechanisms may not be effective in cases where the market experiences sudden and significant price changes.
- **Timelock mechanisms.** Some lending platforms use timelock mechanisms to delay the execution of loan liquidations or other transactions until after the oracle data has been verified and confirmed by multiple sources. This can prevent attackers from exploiting temporary price fluctuations or manipulating the oracle data. However, they can also introduce delays and inefficiencies into the platform. For example, if a timelock mechanism delays the execution of loan liquidations, it may leave the platform vulnerable to losses due to price fluctuations during the delay period. Moreover, timelock mechanisms may not be effective against attacks that occur rapidly or in a coordinated manner.
- **Staking and reputation systems.** Lending platforms can also use staking and reputation systems to incentivize oracle providers to provide accurate and reliable data. For example, the platform can require oracle providers to stake a certain amount of tokens or reputation points to participate, and penalize them for providing inaccurate or manipulated data.A limitation of using staking and reputation systems is that they can be susceptible to sybil attacks or other forms of manipulation. For example, an attacker may be able to create multiple fake oracle providers or manipulate the reputation system by colluding with other providers. Moreover, staking and reputation systems may not be effective in cases where the incentive structure is not aligned with the interests of the platform or its users.

Overall, defending against oracle manipulation attacks requires a combination of technical and economic mechanisms to ensure the accuracy and integrity of the oracle data. By implementing these mechanisms, lending platforms can reduce the risk of losses due to oracle manipulation and provide a more secure and reliable platform for borrowers and lenders. At the same time, DLPs should carefully consider these limitations when designing their defense strategies and take a multi-layered approach to ensure the security and integrity of their platform.

# REFERENCES


[1] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. "SoK: Lending Pools in Decentralized Finance". In: *CoRR* abs/2012.13230 (2020). arXiv: 2012.13230. URL: https://arxiv.org/abs/2012.13230.

[2] Giulio Caldarelli and Joshua Ellul. "The Blockchain Oracle Problem in Decentralized Finance—A Multivocal Approach". In: *Applied Sciences* 11.16 (2021). DOI: 10.3390/app11167572.

[3] Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind. "A Semantic Framework for the Security Analysis of Ethereum Smart Contracts". In: *Principles of Security and Trust*. Ed. by Lujo Bauer and Ralf Küsters. Cham: Springer International Publishing, 2018, pp. 243–269. ISBN: 978-3-319-89722-6.

[4] Daniel Perez et al. "Liquidations: DeFi on a Knife-edge". In: *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*. Springer. 2021, pp. 457–476.

[5] Kaihua Qin et al. "Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit". In: *CoRR* abs/2003.03810 (2020). URL: https://arxiv.org/abs/2003.03810.

[6] *Rekt News*. 2021. URL: https://rekt.news/leaderboard/.

[7] Fabian Schär. "Decentralized Finance: On Blockchain- and Smart Contract-Based Financial Markets". In: *Review* 103.2 (Apr. 2021), pp. 153–174. DOI: 10.20955/r.103.153-74. URL: https://ideas.repec.org/a/fip/fedlrv/91428.html.