Credible Decentralized Exchange Design via Verifiable Sequencing Rules

Matheus Venturyne Xavier Ferreira
matheus@seas.harvard.edu
Harvard University
Boston, MA, USA

David C. Parkes
parkes@eecs.harvard.edu
Harvard University
Boston, MA, USA

ABSTRACT
Trading on decentralized exchanges has been one of the primary use cases for permissionless blockchains with daily trading volume exceeding billions of U.S. dollars. In the status quo, users broadcast transactions they wish to execute in the exchange and miners are responsible for composing a block of transactions and picking an execution ordering—the order in which transactions execute in the exchange. Due to the lack of a regulatory framework, it is common to observe miners exploiting their privileged position by front-running transactions and obtaining risk-free profits. Indeed, the Flashbots service institutionalizes this exploit, with miners auctioning the right to front-run transactions. In this work, we propose to modify the interaction between miners and users and initiate the study of verifiable sequencing rules. As in the status quo, miners can determine the content of a block; however, they commit to respecting a sequencing rule that constrains the execution ordering and is verifiable (there is a polynomial time algorithm that can verify if the execution ordering satisfies such constraints). Thus in the event a miner deviates from the sequencing rule, anyone can generate a proof of non-compliance.

We ask if there are sequencing rules that limit price manipulation from miners in a two-token liquidity pool exchange. Our first result is an impossibility theorem: for any sequencing rule, there is an instance of user transactions where the miner can obtain non-zero risk-free profits. In light of this impossibility result, our main result is a verifiable sequencing rule that provides execution price guarantees for users. In particular, for any user transaction $A$, it ensures that either (1) the execution price of $A$ is at least as good as if $A$ was the only transaction in the block, or (2) the execution price of $A$ is worse than this “standalone” price and the miner does not gain when including $A$ in the block. Our framework does not require users to use countermeasures against predatory trading strategies, for example, set limit prices or split large transactions into smaller ones. This is likely to improve user experience relative to the status quo.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

STOC ’23, June 20–23, 2023, Orlando, FL, USA
ACM Reference Format:
Matheus Venturyne Xavier Ferreira and David C. Parkes. 2023. Credible Decentralized Exchange Design via Verifiable Sequencing Rules. In Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC ’23), June 20–23, 2023, Orlando, FL, USA. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564246.3585233

1 INTRODUCTION
Decentralized finance, also referred to as DeFi, has been one of the main applications of permissionless blockchains. DeFi protocols allow liquidity providers to lock capital into smart contracts. The locked money enables the liquidity provider to obtain revenue from transaction fees by providing liquidity for financial services such as lending and trading. As of 2022, the locked capital in DeFi protocols exceeds $40 billion U.S. dollars [1], with the most prominent decentralized exchange, Uniswap [7], having over $7 billion U.S. dollars in reserves and trading volume that often exceeds billions of U.S. dollars per day [4].

In theory, a decentralized exchange allows traders to submit buy or sell orders to a smart contract without needing any intermediary. Uniswap implements a liquidity pool decentralized exchange as follows. Liquidity providers lock capital into a liquidity pool of two token types. Let $X_i$ be the amount of token $i \in \{1, 2\}$ locked on a liquidity pool. The product $X_1 \cdot X_2$ defines the potential corresponding to the current state of the exchange. A user can submit a trade on Uniswap against the liquidity pool and, for example, withdraw $q > 0$ units of token 1 as long as they deposit a quantity $p > 0$ of token 2 that preserves the potential:

\[(X_1 - q) \cdot (X_2 + p) = X_1 \cdot X_2.\]

Many alternatives exist to the product potential function of Uniswap [20, 36], and Uniswap itself has been generalized in recent years to Uniswap v3, which allows for the amount of locked liquidity to vary by price. Still, a common feature of liquidity pool exchanges is the existence of a reserve $X \in \mathbb{R}^2$ (or, in general, $X \in \mathbb{R}^n$ for $n$ tokens) representing the locked capital (and the exchange state). Users submit orders to buy or sell a particular token, modifying the exchange state after execution. In the example (1), the exchange state changes from $(X_1, X_2)$ to $(X_1 - q, X_2 + p)$ when the user trades $p$ units of token 2 for $q$ units of token 1.
In the blockchain setting, where computation and storage resources are highly scarce, liquidity pool exchanges provide benefits compared with traditional order books. First, the memory storage is constant while an order book’s memory requirement grows with the number of pending transactions. In terms of computation, executing an order requires only a constant number of operations, while an order book requires matching buys with sells and updating underlying data structures. Regarding latency, orders execute instantaneously when processed, while order books need each buy/sell order to wait to match with a corresponding sell/buy order.

In an ideal setting, users would privately submit orders to the liquidity pool exchange; in this sense, there would be no intermediary. In practice, miners act as intermediaries between users and the exchange. Miners choose which pending orders to include in a block and, in the status quo, are also responsible for specifying the order with which transactions execute, which we refer to as the execution ordering.

As one can observe from (1), the state at which an order executes highly influences the trade outcome. Not surprisingly, miners can take advantage of their role to manipulate the content of a block and its execution ordering [40]. This is known as Miner Extractable Value (MEV), also referred to as maximum/maximal extractable value. A well-documented, front-running attack from miners is a sandwich attack [18, 45]. Here, a malicious miner purchases $x > 0$ units of a token ahead of $q > 0$ units purchased by a user, and then immediately sells the $x$ units. The effect is that the miner achieves risk-free profits at the cost of a higher price for the user. This attack has been institutionalized through the front-running-as-a-service business model provided by Flashbots [2, 44], where miners auction the right to allow another party to insert orders and manipulate the order execution.

Although front-running schemes are familiar to traditional finance, a history of regulation in the financial system is designed to protect traders from this kind of market manipulation [12]. However, these regulations have not been enforced on decentralized exchanges where anonymous entities can become a miner and create blocks. Given the absence of regulatory enforcement, there is a great interest in developing algorithmic techniques to mitigate market manipulation by miners (or other entities). Batch auctions are one such approach [5, 13, 37], where transactions are batched and all clear at the same market clearing price. However, blockchains execute transactions sequentially rather than in batches. Thus smart-contract-based batch auctions introduce latency and computational overhead. Instead, we focus on the well-established (sequential) liquidity pool exchange model, where transactions execute sequentially at potentially different prices, due to their computational efficiency and low latency, and we seek to design sequencing rules that can provably mitigate opportunities for miner manipulation.

A trusted relay service [2, 14] like Flashbots Protect (developed by the same organization that operationalizes MEV auctions) is another approach to mitigating manipulation and one that has gained relatively rapid adoption. A user privately communicates their transaction to a trusted service under the promise that the service will not act on privileged information obtained from that user (by injecting transactions and manipulating execution ordering). The relay service recruits a trusted set of miners to include that transaction in a block. Such approaches, however, lack credibility because existing solutions provide no mechanism for which a user can verify that the service manager (or the trusted miner) does not manipulate the execution ordering for profit.

Our approach is similar to a relay service, where users communicate their transactions to the service, and the service sequences these transactions into blocks. However, our approach makes such a relay service credible: we give a concrete procedure by which any user can verify a set of formal guarantees. To be concrete, we model a miner as the entity that is responsible for picking which transactions to include in a block $B$, and that is free to manipulate the block’s content and include its own transactions. However, the miner can commit to picking an execution ordering from a verifiable sequencing rule. Formally, a sequencing rule is a function $S$ that takes the initial state $X_0$ (of liquidity reserves), before any transaction executes, the block $B$ (the transactions to include), and outputs a non-empty set $S(X_0, B)$, which is a set of permutations of $B$. We refer to the elements of $S(X_0, B)$ as valid execution orderings.

We ask that a sequencing rule is efficient, in the sense that there is a polynomial time algorithm that can compute some $T \in S(X_0, B)$ for any $(X_0, B)$. Moreover, we ask that a sequencing rule is verifiable, in the sense that anyone can efficiently check if $T \in S(X_0, B)$ for any $(T, X_0, B)$. Hence, any detectable deviation by a miner from $S$ (picking $T \notin S(X_0, B)$) can be punished, either financially or via reputation loss.

We ask if there are sequencing rules where the miner cannot profit from manipulating the block content while respecting the sequencing rule. We summarize our findings as follows:

(1) **Theorem 4.1 (Folklore).** For a large class of liquidity pool exchanges, including the product potential design of Uniswap, and where miners can pick the content of a block (including adding their own transactions) and sequence...
transactions as they like, a user might receive an arbitrarily bad execution price. That is, a user who buys \( q \) units of a token might make an arbitrarily large payment. Equivalently, a user who sells \( q \) units of a token might receive an arbitrarily small payment.

(2) **Theorem 4.2.** For a class of liquidity pool exchanges (that includes Uniswap), for any sequencing rule, there are instances where the miner has a profitable risk-free undetectable deviation.

(3) **Theorem 5.2.** We specify a sequencing rule (the Greedy Sequencing Rule) such that, for any valid execution ordering, then for any user transaction \( A \) that the miner includes in the block, it must be that either (1) the user efficiently detects the miner did not respect the sequencing rule, or (2) the execution price of \( A \) for the user is at least as good as if \( A \) was the only transaction in the block, or (3) the execution price of \( A \) is worse than this standalone price but the miner does not gain when including \( A \) in the block.

Theorem 4.1 shows a miner can force an arbitrarily bad execution (and profit as a result) if they can pick an execution ordering (in the absence of trading costs for the miner). This result can be weakened by introducing transaction fees (paid by each transaction in the block) or trading fees (proportional to the trading volume). However, sufficiently large transactions can still be the victim of a sandwich attack. While prior work has explored (often complex) trade-offs between fees and order size [29] to mitigate market manipulation, our positive result holds even in the absence of trading costs (blockchain transaction fees or exchange fees).

One can also weaken Theorem 4.1 by allowing users to make use of a *limit price* to specify the maximum they want to pay for a buy order (or the minimum they want to receive for a sell order). By setting a limit price, a transaction fails to execute unless their execution price is as good as this limit. If a transaction fails, no trade takes place, but users must still pay blockchain transaction fees. Requiring all transactions to pay fees, even if they fail, is a security mechanism from existing blockchains to increase the cost for a Denial-of-Service-Attack (where a malicious user nationally changes the state of the exchanges to prevent panic attacks). That is, a miner can profitably insert their own transaction. That happens because the miner never profits from including the user’s transaction. For example, if two users each wish to buy \( q \) units of the same token, then in the absence of any other transactions, it is inevitable that the transaction that executes in second place pays a higher price. This is due to competition for the same token and not due to miner manipulation.

### 1.1 Technical Overview

During a sandwich attack, a miner manipulates the state of the exchange in a way that causes one or more user transactions to achieve a worse execution price. We formalize the properties achieved by our sequencing rule by taking the price at the most recent state of the liquidity reserves, \( X_0 \in \mathbb{R}^2 \), when a user submits a buy or sell order, as a benchmark. This is a relevant benchmark because the blockchain consensus ensures that \( X_0 \) is not manipulable by the miner.4

Crucial for our sequencing rule is the observation that any liquidity pool exchange with two tokens satisfies the following duality property: at any state \( X \in \mathbb{R}^2 \), it is either the case that (1) any buy order receives a better execution at \( X \) than at \( X_0 \), or (2) any sell order receives a better execution at \( X \) than at \( X_0 \). Thus, at any point during the execution of the orders in a block, as long as the transactions yet to execute are not all of the same type (i.e., not all buy orders or all sell orders), there is at least one order that would be happier to be the next order to execute compared with executing at the beginning of the block.

To be concrete, in defining our *Greedy Sequencing Rule*, let \( T_1, T_2, \ldots, T_l \) be the execution ordering up to step \( t \) of the current block; these are the transactions already added to the execution ordering. To define which transaction \( T_{t+1} \) executes at step \( t+1 \), we simulate what the state \( X_0 \) would be after \( T_1, T_2, \ldots, T_t \) executes and add the constraint that \( T_{t+1} \) must be a buy order (any buy order) if buy orders receive a better execution at \( X_t \) than at \( X_0 \) and a sell order (any sell order) if sell orders receive a better execution at \( X_t \) than at \( X_0 \). It is possible that at step \( t+1 \), only buy or sell orders are left to execute. In this case, we let the miner sequence the remaining orders as they wish. Importantly, this rule operates on any set of transactions included in a block and does not need knowledge as to whether a transaction comes from a user or the miner. This makes the rule verifiable based purely on information written on the blockchain.

Interestingly, when only buys or sells are left to execute, we will argue the miner is indifferent as to whether or not to include those in the block. That happens because the miner never profits from executing a buy (respectively sell) order after another buy

---

4 The miner could manipulate \( X_0 \) over multiple blocks, but we assume a different miner creates each block, or equivalently that miners are myopic. This assumption is well-motivated for a decentralized blockchain where the miner for a block is sampled from a large population and the miner selection mechanism is unpredictable so that the miner for a round is unknown until that round starts. This is a common assumption in the context of transaction fee mechanisms [25, 35, 42]. Note that nothing prevents us from updating the benchmark state \( X_0 \) less frequently to decrease the likelihood of manipulation. That is because manipulating \( X_0 \) over a long period of time increases the inventory risk for an attacker.
that is, there is no risk-free gain to the miner from including these transactions, for example $T_{t+1}$, in the block. That is, there is no risk-free gain to the miner from including these transactions, i.e., no gain in tokens to the miner.

Let us see why the Greedy Sequencing Rule makes sandwich attacks unprofitable on Uniswap when there is a single user who wishes to purchase $q$ units of token 1 at market price (i.e., without reporting a limit on how much they would pay) and the initial state in the block is $(X_1, X_2)$. For the setting where the miner can sequence orders as they wish, the miner can obtain a risk-free profit by first front-running the user and purchasing $w < X_1 - q$ units of token 1. Then they execute the user’s order at state $(X_1 - w, X_2 - X_1 + w - X_2)$. After the user’s order executes, the miner sells the $w$ units of token 1 they purchased in the first step (at a higher price).

On the other hand, if the miner commits to implementing the Greedy Sequencing Rule, once the miner purchases $w$ units of token 1, the miner is forced to execute any outstanding sell order (before executing the user’s buy order). Thus the sequencing rule forces the miner to immediately sell the $w$ units of token 1 they just purchased! One can easily check that no matter how many transactions the miner injects into the block, once constrained by the Greedy Sequencing Rule, the miner cannot obtain a risk-free profit when including only a single user transaction in the block. Interestingly, the miner can obtain risk-free profits if the block contains three or more user transactions (as our impossibility result suggests), but without violating the guarantees of Theorem 5.2; i.e., any gains to the miner do not come at the expense of poorer execution price to users. Our work formalizes this intuition, giving results for any two-token liquidity pool exchange and for any number of user transactions included in a block.\footnote{With a single user transaction in the block, our greedy sequencing rule ensures the miner cannot obtain risk-free profits. Our impossibility constructs an attack on any sequencing rule when blocks contain three or more user transactions. We leave as an open question the case where blocks have two user transactions.}

1.2 Related Work

Blockchain consensus. Decentralized exchanges are one of the most impactful applications of decentralized blockchain technology. Nakamoto [38] introduced the Bitcoin digital currency as the first use case for decentralized blockchains. The bitcoin blockchain uses a PoW (Proof-of-Work) longest-chain blockchain to implement a decentralized distributed computer for payments. No single entity owns the bitcoin system because anyone can volunteer to be a miner. The first miner to solve a computationally hard problem receives the privilege to change the state of the distributed computer and receive cryptocurrency in the form of Bitcoin tokens as a reward. Economic incentives play an important role in the security of decentralized blockchains. A line of work started by Eyal and Sirer [22] introduces selfish mining as a way for miners to improve their profit on longest chain PoW blockchains. Sapirshtein et al. [43] and Kiatias et al. [31] provide guarantees for when honest mining is a Nash equilibrium.

The assumption that miners are myopic—they do not manipulate prices across multiple blocks—is rooted on the assumption that a decentralized blockchain uses an unpredictable miner selection. Although longest chain Proof-of-Work (PoW) blockchains satisfy this assumption, they have very high energy consumption [6]. On the other hand, longest chain Proof-of-Stake (PoS) blockchains [16, 19, 32, 33] have a negligible energy cost, but their miner selection are sometimes predictable [11]. Even for non-longest chain PoS blockchains, Ferreira et al. [24] show that an adversary can bias the miner selection making the protocol predictable to a certain degree. Ferreira and Weinberg [27] ask if longest chain PoS blockchains can provide similar miner selection guarantees as longest chain PoW. They show that when the blockchain has access to an external source of randomness (such as the NIST randomness beacon) longest chain PoS blockchains can provide similar (but strictly weaker) fairness guarantees (i.e., unpredictable and unbiased miner selection) than their PoW equivalent.

Incentive analysis in blockchain consensus often assumes a constant reward per block [22, 24, 27, 31, 43]. Carlsten et al. [15], on the other hand, argue that transaction fees, when larger than block rewards, introduce a high variance in the revenue per block and can pose a risk to blockchain security. Qin et al. [41] argue that DeFi applications can also disrupt miner incentives. They measure miner extractable value (MEV) from DeFi applications and quantify their risk to the blockchain security.

Constant product automated market makers. Uniswap is the highest trading volume liquidity pool exchange and uses the product potential [1]. Their exchange is commonly referred as a constant product automated market maker. There is an underlying risk for providing liquidity to these exchanges, but liquidity providers receive trading fees as compensation. Neuder et al. [39] and Heimbach et al. [28] show that complex liquidity provision strategies can improve the liquidity provider’s revenue and Fan et al. [25] studies the tradeoffs between return to liquidity providers and gas fees to traders in the design of Uniswap v3 style schemes for differential price liquidity provision.

Miner extractable value. Our work assumes the miner is profit seeking. Alternatively, front-running on decentralized exchanges have been studied in the context of an honest miner and a self-interested user that attempts to front-run other users [29, 34, 45]. This adversarial model is strictly weaker than ours because a self-interested user has uncertainty over the execution ordering.

Heimbach and Wattenhofer [29] observe that, with trading costs (e.g., trading fees), users can limit their trading volume driving front-running schemes unprofitable. However, their approach is, in effect, limited to a small number of transactions in a block. Otherwise, the adversary can combine multiple transactions by executing them in sequence. For example, if $n > 1$ buy orders each have volume $q > 0$, then the adversary executes all $n$ orders in sequence which, for all purposes, is equivalent to a single order of volume $n \cdot q$. Thus, there is a sufficiently large $n$ where a front-running scheme remains profitable while unprofitable if executed only on individual
transactions. On the other hand, our approach works even if the miner is profit seeking, the number of user transactions per block is unbounded, the trading volume for any particular transaction is arbitrarily large, and there are no trading costs.

**Mechanism design with imperfect commitment.** Traditional mechanism design assumes that the entity running the mechanism can commit to the rules of the game. Front-running schemes would not be a concern if the miner could commit to ordering transactions in the same order as they were observed, i.e., without introducing their own transactions after observing user transactions and interspersing them with suitably ordered user transactions. Unfortunately, one cannot enforce such a sequencing rule because it is not verifiable: in the presence of latency, different miners could observe transactions in different orders [30]. Then the miner has plausible deniability to act on privileged information—and include their own transactions after learning about the user transactions.

Away from the design of mechanisms for decentralized exchanges, this challenge with imperfect commitment is an important constraint in the design of transaction fee mechanisms for blockchains [25, 35, 42]. These are the mechanisms that determine which transactions win the right to be executed on a blockchain and reserves of tokens \(i\).

### Background

#### 2 BACKGROUND

The exchange has a state \(X = (x_1, x_2)\) where \(x_i \geq 0\) is the current reserves of tokens \(i \in \{1, 2\}\). Let \(\{e_1, e_2\}\) be the standard basis of \(\mathbb{R}^2\) which allow us to rewrite \(X = x_1 \cdot e_1 + x_2 \cdot e_2\).

A user submits a transaction that either buys or sells token 1. A buy order \(Buy(q, p)\) purchases \(q\) units of token 1 for at most \(p \cdot q\) units of token 2. A sell order \(Sell(q, p)\) sells \(q\) units of token 1 for at least \(p \cdot q\) units of token 2. We refer to \(p\) as the limit price. An order is a market order if \(p = \infty\) for a buy order and \(p = 0\) for a sell order, and for a market order we omit \(p\) and write \(Buy(q) := Buy(q, \infty)\) and \(Sell(q) := Sell(q, 0)\).

To define the outcome of a transaction, we endow the exchange with a potential function \(\phi: \mathbb{R}_+^2 \rightarrow \mathbb{R}_+\), which is a real-valued continuous function that takes a state \(X\) and maps to the potential \(\phi(X) \geq 0\). We assume \(\phi\) is strictly increasing and quasiconcave as follows:

**Definition 2.1 (Increasing function).** For a function \(f\) we refer to \(\text{dom}(f)\) as the domain of \(f\). For \(x, y \in \mathbb{R}^n\), we write \(x \geq y\) to denote \(x_i \geq y_i\) for all \(i \in \{n\}\). A real-valued function \(f\) is increasing if for all \(x, y \in \text{dom}(f)\), we have that \(x \geq y\), \(f(x) \geq f(y)\). Moreover, \(f\) is strictly increasing if for all \(x, y \in \text{dom}(f) \subseteq \mathbb{R}^n\) such that \(x \geq y\) and \(x_i > y_i\) for some \(i \in \{n\} = \{1, 2, \ldots, n\}\), we have that \(f(x) > f(y)\).

**Definition 2.2 (Convex Set).** A set \(D \subseteq \mathbb{R}^n\) is convex if for all \(x, y \in D\) and \(\alpha \in [0, 1]\), the linear combination \(\alpha \cdot x + (1 - \alpha) \cdot y \in D\).

**Definition 2.3 (Quasiconcave Function).** A real-valued function \(f\) is quasiconcave if \(\text{dom}(f)\) is a convex set and for all \(x, y \in \text{dom}(f)\), and all \(\alpha \in [0, 1]\), we have that \(f(\alpha \cdot x + (1 - \alpha) \cdot y) \geq \min\{f(x), f(y)\}\).

The execution price of an order is the buying price in the case of a buy order, or the selling price in the case of a sell order. We define the execution price algorithmically using the potential function and the current state. For a buy order \(Buy\) executing at state \(X\), the function \(Y(X, Buy(q))\) denotes the amount of token 2 a user would pay for \(q\) units of token 1, which we define as

\[
Y(X, Buy(q)) := \min\{y \geq 0 : \phi(X - q \cdot e_1 + y \cdot e_2) \geq \phi(X)\}. \tag{2}
\]

For a sell order \(Sell\) executing at state \(X\), the function \(Y(X, Sell(q))\) denotes the amount of token 2 a user would trade for \(q\) units of token 1, which we define as

\[
Y(X, Sell(q)) := \max\{y \leq X_2 : \phi(X + q \cdot e_1 - y \cdot e_2) \geq \phi(X)\}. \tag{3}
\]

Although \(Y(X, Buy(q))\) (or \(Y(X, Sell(q))\)) define the execution price at state \(X\), we introduce some feasibility constraints to determine if an order will successfully execute or fail. First, an order must not turn the liquidity reserves negative. Second, the potential at the next state must be the same as the previous state. Third, the user must pay at most \(q \cdot p \cdot e_2\), in the case of a buy order \(Buy(q, p)\), or receive at least \(q \cdot p \cdot e_2\), in the case of a sell order \(Sell(q, p)\). Formally, a buy order \(Buy(q, p)\) can successfully execute at \(X\) if and only if,

\[
Y(X, Buy(q)) \leq p \cdot q, \\
\phi(X_1 - q, X_2 + Y(X, Buy(q))) = \phi(X), \\
X_1 \geq q.
\]

**1.3 Paper Organization**

Our results are not limited to exchange designs such as Uniswap that make use of the constant product potential, and apply to a large class of liquidity pool decentralized exchanges. We provide the necessary background and introduce a general model for liquidity pool decentralized exchange in Section 2. In Section 3, we introduce the communication model. In Section 4, we show front-running schemes can be profitable for a large class of decentralized exchanges. There we also motivate our impossibility result (Theorem 4.2). In Section 5, we define the Greedy Sequencing Rule and prove our main result, Theorem 5.2. We conclude in Section 6. Appendix A contains the necessary mathematical background. The remaining appendices contain omitted proofs.
This section provides formal definitions for some of the potential functions that are used in practice. However, our results hold for a larger class of potential functions, of which the ones defined here are illustrative.

Uniswap [7] uses a product potential function to implement a liquidity pool exchange with two tokens. Balancer [36] uses a similar design but supports pools with two or more tokens.

A concern is that liquidity pool exchanges based on product potentials can have high price volatility when the reserves are small relative to trade volumes. To address this concern, Curve [20] uses a potential function that aims to provide lower price volatility by assuming token prices are stable. We describe these models next.

**Product potential.** The product potential function $\phi$ maps a state $X$ to the product of the current deposits $\phi(X) = X_1 \cdot X_2$. (4)

An exchange with the product potential is also called constant product automated market maker (CPAMM), referencing the fact the product of the reserves is invariant while liquidity providers neither add nor remove liquidity. In a CPAMM, the prices implied by the current state $X$ remain in rough correspondence with the trading prices in a secondary market such as Coinbase. For example, suppose each unit of token $i$ is worth $p_i$ U.S. dollars in the secondary market. We say $p_1/p_2$ is the relative price of token $i$ with respect to token $j$. Then we say a CPAMM is in equilibrium if $X_1/X_2 = p_1/p_2$; otherwise, arbitrageurs would have an incentive to trade in the exchange and take profits in the secondary market. See Angeris and Chitra [9], Angeris et al. [10] for a discussion on the role of arbitrage in automated market makers.

**Stable potential.** Potential functions that imply small variation in prices for even large trades are popular in liquidity pool exchanges in the case that the underlying tokens are expected to have a stable value. For example, stablecoins such as USDC and USDT aim to have a 1-to-1 parity with the U.S. dollar. The stable potential achieves this by combining the product potential function with the additive potential function, defined as

$$\phi(X) = X_1 + X_2.$$ (5)

With just the additive potential function, the price for token 1 with respect to 2 would always equal 1 and the exchange would be unstable because if the prices in a secondary market are $p_1 \neq p_2$, arbitrageurs would have an incentive to trade the token with the lowest price for the token with the highest price until all liquidity reserves are depleted. The stable potential addresses this by interpolating between the additive and the product potentials:

$$\phi(X) = \frac{X_1 \cdot X_2}{X_1 + X_2} \left( X_1 + X_2 \right) + \left( 1 - \frac{X_1 \cdot X_2}{X_1 + X_2} \right) \frac{X_1}{X_2}. \quad (6)$$

To see that (6) interpolates correctly, it suffices to check that $0 \leq \frac{X_1}{X_2} \leq 1$. The lower bound is clear since $X_i \geq 0$ for $i \in \{1, 2\}$. The upper bound follows from the AM-GM inequality, i.e., $X_1 \cdot X_2 \leq \left( \frac{X_1 + X_2}{2} \right)^2$ (Lemma A.1). Note the inequalities are also tight since the lower bound is attained whenever $X_i = 0$ for some $i \in \{1, 2\}$, and the upper bound is attained whenever $X_1 = X_2$.

In the case that an exchange uses the stable potential and the prices in the secondary market are $p_1 = p_2$, arbitrageurs would have an incentive to trade in the liquidity pool whenever $X_1 \neq X_2$. These coins are successful, with a market cap of over 150 billion U.S. dollars as of 2022. [1] USDT and USDC are known as collateralized stable coins, and are issued by entities that promise that users are always able to redeem 1 unit of USDT or USDC for 1 U.S. dollar. This suggests that these stable coins should always have a 1-to-1 parity with the U.S. dollar. Another class of stablecoins known as algorithmic stablecoins is not collateralized. Instead, they rely on incentive mechanisms, and these have so far failed to hold their 1-to-1 parity during periods of market turbulence. Notably, the algorithmic stablecoin UST had a market cap of over $50 billion U.S. dollars when it collapsed overnight in 2022 due to a bank run.
Hence the only equilibrium has $X_1 = X_2$, and the stable potential behaves closer to the additive potential, as desired.

### 2.2 Model Discussion

This section argues why our assumptions on potential functions are, in essence, without loss. Firstly, the potential must be strictly increasing because this is equivalent to requiring that users should only be able to withdraw tokens from the exchange if they deposit some payment.

**Definition 2.5.** A potential $\phi$ has non-zero payment if for all states $X \in \text{dom}(\phi)$, we have that $Y(X, \text{Sell}(0)) = 0$.

One can check that all the potential functions from Section 2.1 when restricted to states $X > 0$ have non-zero payment. The product potential does not have zero payments if the domain contains some state $X$ where $X_i = 0$ for some $i$. To show an exchange satisfies non-zero payment, Lemma 2.1 shows that it suffices to check that $\phi$ is strictly increasing with the mild assumption that $\text{dom}(\phi)$ is open and upward closed.

**Definition 2.6 (Open and upward closed sets).** Let $B_\varepsilon(x) = \{ y \in \mathbb{R}^n : ||x - y|| \leq \varepsilon \}$ be the a ball around $x$. A set $D \subseteq \mathbb{R}^n$ is open if for all $x \in D$, there is $\varepsilon > 0$ such that for all $y \in B_\varepsilon(x)$, we have that $y \in D$. A set $D \subseteq \mathbb{R}^n$ is upward closed if for all $x \in D$, if $y \geq x$, then $y \in D$.

**Lemma 2.1.** Let $\text{dom}(\phi)$ be open and upward closed. A potential $\phi$ has non-zero payment if and only if $\phi$ is strictly increasing.

Proof. We prove the lemma in two parts.

**Claim 2.1.** If $\phi$ is strictly increasing, then $\phi$ has non-zero payment.

**Proof.** Fix any $x \in \text{dom}(\phi)$. Fix $\varepsilon > 0$ and $X' = X - y \cdot e_2$ for any $y \in (0, \varepsilon)$. Because $\text{dom}(\phi)$ is open, there is a $r > 0$ such that $X' \in \text{dom}(\phi)$. Because $\phi$ is strictly increasing and $X'$ is strictly bigger than $X$, we conclude $\phi(X') < \phi(X)$. This proves $Y(X, \text{Sell}(0)) \leq y \leq \varepsilon$. Taking the limit as $\varepsilon \to 0$ proves that $Y(X, \text{Sell}(0)) = 0$ as desired.

**Claim 2.2.** If $\phi$ has non-zero payment, then $\phi$ is strictly increasing.

**Proof.** Fix any $X \neq X' \in \text{dom}(\phi)$ and w.l.o.g. assume $X \geq X'$. Let $p = X - X' \geq 0$. Define $Z_j = X - \sum_{i=1}^{j} p_i \cdot e_i$ for all $j$ and observe $Z_0 = X$ and $Z_2 = Z'$. Note $Z_j \in \text{dom}(\phi)$. To see, observe $Z_j \geq Z_2 = Z'$ and $Z' \in \text{dom}(\phi)$ which combined with the assumption $\text{dom}(\phi)$ is upward closed implies $Z_j \in \text{dom}(\phi)$. Now observe that $\phi(Z_j) < \phi(Z_{j-1})$ for all $j$ where $p_j > 0$; otherwise, the event $\phi(Z_j) \geq \phi(Z_{j-1})$ implies $Y(Z_{j-1}, \text{Sell}(0)) \geq p_j > 0$, a contradiction to the assumption $\phi$ has non-zero payment. Note there is at least one $p_j > 0$ because $X \neq X'$. This proves $\phi(X) = \phi(Z_0) > \phi(Z_2) = \phi(X')$ as desired. Thus $\phi$ is strictly increasing.

Combining both claims proves Lemma 2.1.

Our second assumption is that potential functions are quasiconcave. In Lemma 2.2, we show that assuming $\phi$ is quasiconcave and strictly increasing ensures that buying token 1 only increases the price of token 1 relative to token 2 and selling token 1 only decreases the price of token 1 relative to token 2.

**Lemma 2.2 (Pricing Lemma).** Consider states $X$ and $X'$ where $\phi(X) = \phi(X')$ and $X'_1 < X_1$ and assume the potential function $\phi$ is quasiconcave and strictly increasing. Then the following hold:

- If $\text{Buy}(q)$ can successfully execute at both $X$ and $X'$, then $Y(X, \text{Buy}(q)) \leq Y(X', \text{Buy}(q))$.
- If $\text{Sell}(q)$ can successfully execute at both $X$ and $X'$, then $Y(X, \text{Sell}(q)) \leq Y(X', \text{Sell}(q))$.

We provide the proof of the Pricing Lemma in Appendix B which follows from first principles.

### 3 COMMUNICATION MODEL

The risk of market manipulation in liquidity pool exchanges arrives from how users communicate their transactions with the exchange. Suppose users $1, 2, \ldots, |A|$ want to execute transactions $A_1, A_2, \ldots, A_{|A|}$ at state $X_0$. Note a single entity could control multiple users, but that is not relevant for our analysis. Each user privately sends their transaction to the miner. The miner aggregates observed transactions into a block $B$, which we model as a set of potentially unbounded size.

The order of transactions in the block defines the execution ordering—the order by which transactions execute in the decentralized exchange. In our model, miners pick the block (i.e., the transactions to include), but use a sequencing rule $S$ to determine the execution ordering.

**Definition 3.1 (Sequencing Rule).** A sequencing rule $S$ is a function from a state $X$ (of the liquidity reserves before any transaction executes) and a set of transactions $B$ to a non-empty set system $S(X, B)$ containing permutations of $B$.

First, we would like a sequencing rule to be efficiently computable in order to minimize the computational burden on miners.

**Definition 3.2 (Efficient Sequencer).** A sequencing rule $S$ is (computationally) efficient, if for all initial state $X = (X_1, X_2)$ and block $B$, there is an algorithm that takes $(X_0, B)$ and outputs some $T \in S(X_0, B)$ in time $O(\log(X_1 + X_2)|B|)$.

Any block sequencing algorithm requires at least $\log(X_1 + X_2)|B|$ computation to read the content of $B$. Thus our definition requires that a sequencing rule imposes at most a constant multiplicative computational overhead when compared with the status quo, i.e., where the miner computes their favorite ordering of $B$.

We are ready to define the trading game $(X_0, \{A_i\}, S)$ between users and a miner. The game takes as input a transaction, $A_i$, from each user $i$, the initial state $X_0$, and a sequencing rule $S$. The outcome of the game is an execution ordering on a set of transactions and associated sequence of states, where the transactions that are ordered can include a subset of user transactions and additional transactions that may be introduced by the miner. In the case of an honest miner, the game proceeds as in Algorithm 2:

Algorithm 2 assumes the miner commits to implement all the steps faithfully. However, we assume miners can perform the following deviations in the real trading game, these comprising the strategy of the miner, with knowledge of the input:

1. At step (3), by censoring transaction $A_i$ (not adding $A_i$ to block $B$).
ideal trading game \( X_0, A_i, S \)  
**Input** Initial state \( X_0 \); order \( A_i \) from each user \( i \); sequencing rule \( S \)  
**Output** Execution ordering \( T = (T_1, \ldots, T_{|T|}) \) and states \( X_1, X_2, \ldots, X_{|T|} \) where \( T_i \) executes at \( X_{t−1} \)  
1. The miner initializes the block \( B = \emptyset \).  
2. For all \( i \), user \( i \) privately sends order \( A_i \) to the miner.  
3. For all \( i \), the miner adds \( A_i \) to \( B \).  
4. The miner picks some \( (T_1, \ldots, T_{|B|}) \in S(X_0, B) \) as the execution ordering.  
5. For \( t = 1, \ldots, |B| \),  
   a. The exchange executes \( T_i \) at state \( X_{t−1} \).  
   b. Let \( X_t \) be the state after \( T_i \) executes on \( X_{t−1} \).  

**Algorithm 2:** Trading game with an honest miner.  

(2) Between step (3) and (4), by including their own transactions into \( B \).  
(3) At step (4), by picking an execution ordering \( T \not\in S(X_0, B) \).  

We assume a self-interested miner who will follow any set of deviations that are profitable and undetectable by an observer. For this, we assume that the observer can only see the blockchain state (which includes \( X_0 \) and the outcome of the trading game). By assuming an observer can see the outcome but not the set of outstanding orders, we avoid introducing (strong) assumptions over the communication channel. For example, a naive approach to mitigate censorship is to suggest that users broadcast their transaction. Thus, an observer who sees order \( A_i \) must conclude that the miner also saw \( A_i \). If the miner did not include \( A_i \) into \( B \), then the observer can conclude the miner censored \( A_i \). Unfortunately, it is not possible to confirm that a miner receives a transaction in the presence of latency. The miner can simply refuse to acknowledge the receipt of a message, with latency providing plausible deniability. Moreover, a malicious user could send \( A_i \) to the observer and not to the miner, harming the miner’s reputation. To avoid these concerns, we assume our observer can only rely on information stored in the blockchain to detect a miner deviation.

**Definition 3.3 (Safe deviation).** The outcome of the trading game \( X_0, (A_i), S \) with an honest miner is \( (X_0, T) \) where \( T \in S(X_0, (A_i)) \). A deviation for the miner is a strategy that results in an outcome \( (X_0, T) \) where \( T \) is not necessarily contained in \( S(X_0, (A_i)) \). A deviation resulting in an outcome \( (X_0, T) \) is safe if there is a trading game \( X_0, (A'_i), S \) where \( T \in S(X_0, (A'_i)) \).  

In other words, a deviation for the miner is safe if it can be explained by a different set of user transactions \( (A'_i) \). For a given sequencing rule, we define the language,  
\[
\mathcal{L}(X_0, S) = \{(X_0, T) : (A_i), T \in S(X_0, (A_i))\},  
\]
which is the set of all possible outcomes for the sequencer \( S \) with initial state \( X_0 \). An important constraint for a sequencing rule is that an observer must be able to efficiently check whether \( (X_0, T) \in \mathcal{L}(X_0, S) \), for any outcome \( (X_0, T) \). This can be formalized as follows:  

**Definition 3.4 (Verifiable sequencing rule).** A sequencing rule with language \( \mathcal{L} \) is verifiable if there is a polynomial time algorithm \( P \) that takes any outcome \( (X_0, T) \) and outputs True if \( (X_0, T) \in \mathcal{L}(X_0, S) \) or False if \( (X_0, T) \not\in \mathcal{L}(X_0, S) \).  

**User and miner utility.** Consider an outcome \( (X_0, T) \). Let \( T_{σ(𝑖)} \) denote the transaction owned by user \( i \) in the execution order \( T \), which is undefined if the miner censors user \( i \). Then, we define user \( i \)’s utility for the outcome as  
\[
\begin{align*}
  u_i(X_0, T) &= \begin{cases} 
    (0, 0) & \text{if } σ(𝑖) \text{ is undefined}, \\
    X_σ(𝑖) - X_σ(𝑖−1) & \text{otherwise}.
  \end{cases}
\end{align*}
\]

This is the difference in tokens owned by the user that correspond to the executed transaction, or no change in the case that the user’s transaction is not executed. For an entity that controls users \( i \) and \( j \), their utility would be \( u_i(X_0, T) + u_j(X_0, T) \).

Let \( I \subseteq \{1, \ldots, \lfloor T \rfloor \} \) be the time steps where one of the miner’s transaction executes (perhaps empty). Then the miner’s utility is  
\[
u_0(X_0, T) = \sum_{i \in I} (X_t - X_{t−1}).\]

For outcomes \( (X_0, T) \) and \( (X_0, T') \), we say \( (X_0, T) \) dominates \( (X_0, T') \) for agent \( i \) if \( u_i(X_0, T) \geq u_i(X_0, T') \). The outcome is a risk-free execution for agent \( i \) if \( u_i(X_0, T) \geq 0 \). A risk-free execution is profitable for agent \( i \) if the agent has a strictly positive quantity of some token and a non-negative quantity of the other token. A common feature of a front-running scheme is that the miner has an execution ordering and set of transactions to insert that provides it with a profitable, risk-free execution (with \( 0 \neq u_0(X_0, T) \geq 0 \)).

**Example 3.1.** Let \( u_i(X_0, T) \) be the utility of agent \( i \). If \( u_i(X_0, T) = (-1, 1) \), then \( (X_0, T) \) is not a risk-free execution for agent \( i \), because agent \( i \) pays 1 unit of token 1. If \( u_i(X_0, T) = (0, 0) \), then \( (X_0, T) \) is a risk-free execution for agent \( i \), but not a profitable one. If \( u_i(X_0, T) = (1, 0) \), then \( (X_0, T) \) is a profitable, risk-free execution for agent \( i \).

**4 Market Manipulation.**

One desirable property for an exchange with the product and stable potential functions is that a user cannot completely deplete token 1 reserves because they would need to deposit an unbounded quantity of token 2. We can formalize this property as follows:  

**Definition 4.1 (Liquidity-preserving).** An exchange is liquidity-preserving if, for all states \( X > 0 \), we have  
\[
\lim_{\substack{q \to X \\downarrow}} Y(X, \text{BUY}(q)) = \infty.
\]

Unfortunately, users interacting with any liquidity-preserving decentralized exchange are vulnerable to front-running if the miner can arbitrarily choose an execution ordering. We first illustrate this in the following example for the product potential.

**Example 4.1 (Front-running, product potential).** Consider the product potential, \( \phi(X) = X_1 - X_2 = c \), and suppose a user submits a market order \( T_u = \text{BUY}(q) \). Recall the set of reachable states is the level set \( L_c(\phi) \). Also observe that \( X_1 \) uniquely determines \( X_2 \) given \( c \) (because \( \phi \) is strictly increasing, Lemma B.1). In the front-running scheme from Figure 1, the miner sequences their own buy order, \( T_1 \), before \( T_u \) and their own sell order, \( T_3 \), after \( T_u \). This is a sandwich
attack. The effect of “running ahead” of the user’s transaction is that the price increases after the miner’s buy order. As a result, this is a profitable risk-free manipulation. For the user, the effect is a worse execution price.

Theorem 4.1 generalizes Example 4.1 to the case of a user’s transaction that includes a limit price, and for a general exchange rule (not just the product potential), and shows that the miner can obtain a profit that increases as the user increases the limit price.

Theorem 4.1. Consider a buy order \( \text{Buy}(q, p) \), for quantity \( q \) at limit price \( p \), and assume this is feasible at state \( X \). Assume the exchange is liquidity-preserving. Then there is an execution ordering where the user swaps \( q \cdot p \) units of token 2 for \( q \) units of token 1, and the miner receives \( q \cdot p - Y(X, \text{Buy}(q)) \) units of token 2 for free.

Proof. Because \( \text{Buy}(q, p) \) is feasible at state \( X \), it satisfies the constraints \( q \leq X_1 \) and \( Y(X, \text{Buy}(q, p)) \leq q \cdot p \). Instead of executing only \( \text{Buy}(q, p) \) at \( X \), the miner injects their own buy and sell by picking the execution ordering

\[
(\text{Buy}(X_1 - q - w), \text{Buy}(q, p), \text{Sell}(X_1 - q - w)).
\]

The miner picks the smallest \( w \) \( \geq 0 \) that still allows the user transaction to successfully execute. Observe such a constant exists because setting \( w = X_1 - q \) ensures the user transaction successfully executes.

Next, we claim the user pays exactly \( q \cdot p \) for this value of \( w \). To see this, let \( x \geq q \), and observe that the liquidity-preserving assumption implies

\[
\lim_{x \to X_1} Y(X, \text{Buy}(x-q)) + Y(X, \text{Buy}(q)) = Y(X, \text{Buy}(x)) = \infty,
\]

which is continuous. From the intermediate value theorem (Lemma A.1), we conclude there is a value \( w \geq 0 \) such that the user pays \( q \cdot p \) units of token 2.

The state after all transactions execute is \((X_1 - q, X_2 + Y(X, \text{Buy}(q)))\). Because the user deposits \( q \cdot p \cdot e_2 \), the miner receives the difference \( q \cdot p - Y(X, \text{Buy}(q)) \).

In the case of the additive potential function (5), which is not liquidity-preserving, the deviation used by the miner in the proof of Theorem 4.1 is not profitable and, in particular, the miner’s utility is \((0,0)\). Whenever \( \text{Buy}(q) \) successfully executes, the miner trades \( q \) units of token 2 for \( q \) units of token 1. Equivalently, whenever \( \text{Sell}(q) \) successfully executes, the miner trades \( q \) units of token 1 for \( q \) units of token 2. As a result, the miner’s net utility from executing orders \( \text{Buy}(q) \) and \( \text{Sell}(q) \) is always zero regardless of their position in the execution ordering. Although robust to market manipulation, non-liquidity preserving exchanges fail to execute transactions once they have exhausted their liquidity reserves, making then unsuitable for settings where secondary-market prices fluctuate.

**Impossibility result.** We already established (Theorem 4.1) that miners can obtain large profits when they pick the block content along with the execution ordering. Next, we ask what is possible if miners can only choose the block content, but not the execution ordering. One might ask if there is a sequencing rule \( S \) where, for any block \( \{B_i\} \) (containing miner and user transactions), and initial state \( X_0 \), there is no execution ordering \( T \in S(X_0, B) \) that gives risk-free profits for the miner. Unfortunately, Theorem 4.2 shows that this is impossible even when \( \{B_i\} \) contains only three user transactions. For simplicity, we state Theorem 4.2 under the assumption that the exchange uses the product potential (which suffices for an impossibility result); however, one can generalize the statement for any liquidity pool exchange that suffers price impact, i.e., the token 1 (resp. 2) price strictly increases as token 1 (resp. 2) reserves decreases.

Theorem 4.2. Consider a liquidity pool exchange with the product potential. Then for any sequencing rule \( S \), there is an initial state \( X_0 \) and a block \( B \), containing miner transactions and a set of three user transactions, such that the miner receives a strictly positive quantity of token 2 and pays nothing if the execution ordering is contained in \( S(X_0, B) \).

The idea is to consider a block that contains \( n \geq 3 \) identical buy orders, \( \text{Buy}(2) \), and the same number of identical sell orders, \( \text{Sell}(1) \), where one of the buy orders and two of the sell orders are the miner’s. We assume \( X_1 > 4n \) in the initial state, so that no transaction ever fails to execute. We fix the execution ordering. Then because the execution ordering from a sequencing rule does not depend on whom owns which transaction, we choose which buys and sells the miner owns after observing the ordering.

It is clear that after the miner executes their orders, their utility in token 1 is zero. It suffices to argue that for any permutation of the \( 2n \) transactions, there is always one buy order and two sell orders where the buy order buys at an average price \( p \) and the two sell orders sells at an average price that is strictly larger than \( p \). Letting the miner own these particular, three transactions, implies profit to the miner because there is a gain in token 2 and no change in position for token 1. See Figure 2 for one of the permutations, where if the miner owns orders \( \{T_1, T_4, T_5\} \), they receive a positive quantity of token 2 for free.

Proof of Theorem 4.2. Let \( X_{0,1} = X_{0,2} = 4n \), where \( n \geq 3 \). The miner picks a block \( B \) that contains \( n \) buy orders equal to \( \text{Buy}(2) \) and \( n \) sell orders equal to \( \text{Sell}(1) \). One of the buys and two of the sells will be the miner’s. Let the execution order be \( T \in S(X_0, B) \).
We reach a contradiction, since $T$ always receives a strictly positive quantity of token $2$.

Next, we will pick which particular buys and sells the miner owns in a way that depends on $T$.

Recall $X_t$ is the state after order $T_t$ executes on $X_{t-1}$. $X_{t,i}$ is the token $i$ reserve at state $X_t$. Let $Z_t = X_{t,1} − X_{0,1}$, and observe $Z_0 = 0$ and $Z[T] = −n$ for any execution ordering. Moreover, $Z_t = Z_{t-1} + 1$ whenever $T_t$ is a sell order and $Z_t = Z_{t-1} − 2$ whenever $T_t$ is a buy order.

Let $i ∈ arg\ max_t{Z_t}$ ($T_t$ is a buy order), then $T_i$ is the buy order that pays the lowest price. Observe $Z_{t-1} = k$ for some $k ≥ 0$ because $Z_0 = 0$ and $Z[T] < 0$. Next consider the following cases:

**Case 1.** There are two sell orders $T_2$ and $T_3$ that execute at a state where $Z_{t-1} < (k − 2)$ and $Z_{t-1} < (k + 2)$, respectively. If the miner owns orders $T_i = Buy(2)$, $T_2 = Sell(1)$, and $T_3 = Sell(1)$, then the miner ends the trading game with a strictly positive quantity of token 2 because they sell two units of token 1 at a lower average price than they were purchased.

**Case 2.** There are $m ≥ n − 1$ sell orders that execute at states where $Z_{t-1} ≥ k − 1$. Let $T_i$ be one of these sell orders. We first argue that $T_i$ executes at a state where $Z_{t-1} = k − 1$. To see this, observe that if $Z_{t-1} > k − 1$, then $Z_t = Z_{t-1} + 1 ≥ k + 1 > 0$.

We reach a contradiction, since $k = Z_{t-1} ≥ Z_{t-1} ≥ k + 1$. This proves that if $T_t$ is a sell order executing a state where $Z_{t-1} ≥ k − 1$, then $Z_{t-1} = k − 1$. This also implies that, for all times steps $t$, we have $Z_t = k_i$. Now for any sell order $T_i$ where $Z_{t-1} = k_i$, we will have that $Z_t = k_i$. Then $T_i$ must be a buy order, and we will have that $Z_{t+2} = k − 2$. Therefore, we can only have $m$ sell orders executing at states where $Z_{t-1} = k − 1$ if there are $m$ sell orders executing at states where $Z_j = k − 2$. Because $m = n − 1 ≥ 2$, we reach a contradiction to the assumption that at most one sell order executes at a state where $Z_{t-1} ≤ k − 2$. This proves that Case 2 never happens.

Cases 1 and 2 cover all scenarios. This proves that the miner always receives a strictly positive quantity of token 2 and pays nothing. □

### 5 Greedy Sequencing Rule

The Greedy Sequencing Rule (Algorithm 3) takes a set of transactions $B$ and an initial state $X_0$ (denoting the state before a transaction in this block executes on the chain), and recursively constructs an execution ordering $(T_1, \ldots, T_t)$ (a permutation of the transactions in $B$). The set of transactions $B$ may include both user and miner transactions. We refer to $(T_1, \ldots, T_t)$ as the execution ordering up to step $t$ and state $X_t$ denotes the state after $(T_1, \ldots, T_t)$ executes on state $X_0$. We partition the set of outstanding transactions, that is the transactions in $B$ that are not in $\{T_1, \ldots, T_t\}$, into two groups: $B^{buy}$ that contains the outstanding buy orders and $B^{sell}$ that contains the outstanding sell orders. At step 0, $B = B^{buy} \cup B^{sell}$.

What would be a good choice for $T_{t+1}$? We suppose that when user $i$ communicated their order $A_i$ to the miner, the user could observe $X_0$ as the current state of the exchange, and is comfortable with $A_i$ executing at $X_0$. However, $A_i$ will execute at state $X_t$ if this forms the $(t + 1)$-th transaction in the block. If $A_j$ is a buy order, the user prefers $Y(X_t, A_j)$ to be as small as possible. However, if $A_j$ is a sell order, the user prefers $Y(X_t, A_j)$ to be as large as possible.

Our main observation is that as long as both $B^{buy}$ and $B^{sell}$ are not empty, there is at least one transaction that would be at least as happy when executing at $X_t$ as when executing at $X_0$. To be concrete, we show that if the token 1 reserves at $X_t$ are smaller than those at $X_0$, then any sell order would be at least as happy by executing at $X_t$ than $X_0$. Conversely, if the $X_t$ token 1 reserves are higher than at $X_0$, then any buy order would be at least as happy by executing at $X_t$ than $X_0$. We formalize this property in the Duality Theorem (Theorem 5.1) which we prove in Appendix C.

This immediately suggests a good choice for $T_{t+1}$. The Greedy Sequencing Rule allows the miner to pick $T_{t+1}$ to be any buy order from $B^{buy}$ if it is the buy orders that prefer to execute at $X_t$ than $X_0$; otherwise, the rule allows the miner to pick $T_{t+1}$ to be any sell order from $B^{sell}$. Duality ensures that one of these conditions is satisfied as long as neither $B^{buy}$ or $B^{sell}$ are non-empty. Once one of $B^{buy}$ or $B^{sell}$ is empty, we allow the miner to append the remaining transactions in any arbitrary order (just as in the status quo). Interestingly, we will argue that from the moment that $B^{buy}$ or $B^{sell}$ are empty, the miner has nothing to profit (or lose) from manipulating the execution ordering going forward.

```
procedure Greedy Sequencing Rule($X_0, B$)
input Initial state $X_0$, set of transactions $B$.
output Execution ordering $T$.

Procede as follows:
(1) Initialize $T$ as an empty list.
(2) Let $B^{buy} \subseteq B$ be the collection of buy orders in $B$. Let $B^{sell} \subseteq B$ be the collection of sell orders in $B$.
(3) While $B^{buy}$ and $B^{sell}$ are both non-empty:
   (a) Let $t = |T|$.
   (b) If $X_{t+1} \geq X_{0,1}$:
      (i) Let $A$ be any order in $B^{buy}$.
      (ii) Append $A$ to $T$ and remove $A$ from $B^{buy}$.
      (c) Else:
         (i) Let $A$ be any order in $B^{sell}$.
         (ii) Append $A$ to $T$ and remove $A$ from $B^{sell}$.
   (d) Let $X_{t+1}$ be the state after $A$ executes on $X_t$.
(4) If $B^{buy} \cup B^{sell}$ is non-empty, append the remaining transactions in $B^{buy} \cup B^{sell}$ to $T$ in any order.
end procedure
```

**Definition 5.1 (Execution quality).** An order $A$ receives a better execution at state $X$ than $X'$ if either:
where w wants to pick the execution ordering.

**Theorem 5.1** (Duality Theorem). Consider any liquidity pool exchange with potential φ. For any pair of states, X, X′ ∈ L(φ), either:
- any buy order receives a better execution at X than X′, or
- any sell order receives a better execution at X than X′.

The premise for a front-running scheme is that the miner manipulates the liquidity reserves to force a sell order to execute at a state X0 where the token 1 reserves are higher than at X0 and a buy order to execute at a state X0 where the token 1 reserves are smaller than at X0. In particular, if the miner wants a sell order to execute at a state X0 where the token 1 reserves are higher than at X0, then the miner intends to execute a buy order immediately after this transaction. The Greedy Sequencing Rule avoids this pattern because it forces the miner’s buy order to execute before the sell order.

To illustrate this, we consider an example with a single user order, Buy(q) (see Figure 3). Recall for a front-running attack, the miner wants to pick the execution ordering (Buy(q′), Buy(q), Sell(q′)) where Sell(q′) and Sell(q′) are the miner’s own transactions. The crucial observation is that the Greedy Sequencing Rule would never output this as a valid execution ordering. In fact, with the Greedy Sequencing Rule, the miner can only output the following four execution orderings on these transactions:

\[ T^{(1)} = (\text{Buy}(q), \text{Sell}(q′), \text{Buy}(q′)), \text{ or} \]
\[ T^{(2)} = (\text{Buy}(q′), \text{Sell}(q′), \text{Buy}(q)), \text{ or} \]
\[ T^{(3)} = (\text{Sell}(q′), \text{Buy}(q′), \text{Buy}(q)), \text{ or} \]
\[ T^{(4)} = (\text{Sell}(q′), \text{Buy}(q), \text{Buy}(q′)). \]

In T^{(1)}, T^{(2)}, and T^{(3)}, the miner’s orders simply cancel each other out, resulting in no profit. The miner prefers T^{(3)} over T^{(4)} since their buy order executes at a better price. This shows that, given this sequencing rule, the miner is indifferent as to whether Buy(q) is in the block or not.

### 5.1 Execution Quality

In this section, we show that users obtain a predictable execution price that is immune to front-running schemes. Our analysis accounts for an unbounded number of user transactions per block and allows the miner to inject as many transactions as they desire.

**Definition 5.2** (Core/Tail Decomposition). For an outcome (X₀, T) from any trading game, we partition transactions in T into two sets: the core denoted Core(X₀, T) and the tail denoted Tail(X₀, T). A transaction Tᵢ is in the core, if Tᵢ receives a better execution at Xᵢ−1 than X₀. The tail contains the remaining transactions.

**Lemma 5.1.** Let (X₀, T) be an outcome from a trading game with the Greedy Sequencing Rule S. Then Tail(X₀, T) contains only buy orders or only sell orders.

Proof. The outcome (X₀, T) ∈ L(X₀, S) and there is a corresponding trading game (X₀, (Tᵢ)ₜ), S) that outputs (X₀, T). During the execution of the trading game, let Tᵢ be an order that is appended to T when neither B_buy nor S_sell are empty. We claim Tᵢ ∈ Core(X₀, T). If Tᵢ is a buy order, it must be that Tᵢ executes at a state Xᵢ−1 where token 1 reserves are higher than at X₀. Consider the following cases:

- Tᵢ fails to execute at Xᵢ−1. From Lemma C.1, if Tᵢ fails to execute at Xᵢ−1, then it also fails to execute at X₀. Thus Tᵢ receives a better execution at Xᵢ−1 than X₀.
- Tᵢ can successfully execute at Xᵢ−1. If Tᵢ fails to execute at X₀, then Tᵢ receives a better execution at Xᵢ−1 than X₀. If Tᵢ can successfully execute at X₀, the Pricing Lemma (Lemma 2.2) states that Y(Xᵢ−1, Tᵢ) ≤ Y(X₀, Tᵢ), since token 1 reserves are higher at Xᵢ−1. Thus Tᵢ receives a better execution at Xᵢ−1 than X₀.

The above proves if Tᵢ ∈ B_buy, then Tᵢ ∈ Core(X₀, T). Next, we consider the case Tᵢ ∈ S_sell. It must be that Tᵢ executes at a state Xᵢ−1 where token 1 reserves are lower than at X₀. Consider the following cases:

- Tᵢ fails to execute at Xᵢ−1. From Lemma C.1, if Tᵢ fails to execute at Xᵢ−1, then it also fails to execute at X₀. Thus Tᵢ receives a better execution at Xᵢ−1 than X₀.
- Tᵢ can successfully execute at Xᵢ−1. If Tᵢ fails to execute at X₀, then Tᵢ receives a better execution at Xᵢ−1 than X₀. If Tᵢ can successfully execute at X₀, the Pricing Lemma (Lemma 2.2) states that Y(Xᵢ−1, Tᵢ) ≥ Y(X₀, Tᵢ), since token 1 reserves are lower at Xᵢ−1. Thus Tᵢ receives a better execution at Xᵢ−1 than X₀.

The above proves if Tᵢ ∈ S_sell, then Tᵢ ∈ Core(X₀, T). Thus all transactions Tᵢ ∈ Tail(X₀, T) were added to the execution ordering when either B_buy or S_sell were empty. This proves that all transactions in Tail(X₀, T) must be of the same type, and only buy orders or only sell orders.

Let \( T_{-i} = \{ T₁, ..., Tᵢ₋₁, Tᵢ₊₁, ..., T | T \} \) denote the execution ordering \( T \) without \( Tᵢ \).

**Theorem 5.2.** Let (X₀, T) be an outcome from a trading game with the Greedy Sequencing Rule. Then for any user \( i \in A \) with a transaction \( Tᵢ \) ∈ \( T \), we have one of the following

1. **Indifference.** The execution ordering \( T₋ᵢ \) dominates \( T \) for the miner.
2. **Isolation.** The execution ordering \( T \) dominates \( Tᵢ \), the execution ordering that contains only order \( Tᵢ \), for user \( i \).

Proof. Fix a user \( i \in A \) with an order \( Tᵢ \) ∈ \( T \). Consider the following cases:
We conclude by providing a proof that the Greedy Sequencing Rule, to see this, consider the same example from Section ?? and suppose for contradiction the verifier outputs an input $\sigma$. The feasible outcomes for the sequencing rule is sequenced after $X$. They obtain a positive quantity of token 2 for free. Let us check the strongest positive result might come as a surprise given the execution ordering $T$. By letting the miner own order orders $X$, $X$ is sequenced after $X$. This proves the theorem.

The first case argues the miner can only improve their utility by excluding the user’s transaction from the block, while the second case argues the user receives an execution as good as the optimal execution they would receive in isolation. This proves the theorem. □

This strong positive result might come as a surprise given the impossibility result (Theorem 4.2). However, the miner can still profit from injecting their own transactions in the greedy sequencing rule, but provably without acquiring an execution price worse than what the user expected if sequenced at the start of the block. To see this, consider the same example from Section ??, where the block contains three identical buy orders, $\text{Buy}(2)$, and three identical sell orders, $\text{Sell}(2)$. For contradiction, the verifier outputs $\text{Buy}(2)$ and $\text{Sell}(2)$ are non-empty, this is always possible. This contradicts the second bullet, and proves the verifier outputs $\text{True}$.

Next, consider the case where $(X_0, T) \notin \mathcal{L}(X_0, S)$, and this is not a valid outcome from the greedy sequencing rule. Suppose for contradiction, the verifier outputs $\text{True}$. By inspection, we can construct a trading game instance $(X_0, \{A_i\}, S)$ that outputs $(X_0, T)$. Thus $T \in S(X_0, \{A_i\})$, implies $(X_0, T) \in \mathcal{L}(X_0, S)$, which is a contradiction.

```
procedure Verifier(X, T)
input Outcome (X, T).
output True | False.
(1) For $t = 1, 2, \ldots, |T|$: (a) If $T_t, T_{t+1}, \ldots, T_{|T|}$ are orders of the same type (i.e., all are buy orders or all are sell orders), then output True. (b) If $X_{t-1} \geq \max X_0.1$ and $T_t$ is a buy order, then output False. (c) If $X_{t-1} < X_0.1$ and $T_t$ is a sell order, then output False. (d) Let $X_t$ be the state after $T_t$ executes on $X_{t-1}$. (2) Output True.
end procedure
```

Algorithm 4: The Verifier for the Greedy Sequencing Rule.

5.2 The Greedy Sequencing Rule Is Verifiable

We conclude by providing a proof that the greedy sequencing rule is verifiable. Recall that $\mathcal{L}(X_0, S)$ (??) is a language representing the feasible outcomes for the sequencing rule $S$ when the initial state is $X_0$. We define Algorithm 4 as our verifier. Let us first check that the verifier outputs $\text{True}$ if $(X_0, T) \notin \mathcal{L}(X_0, S)$ is a valid outcome from the greedy sequencing rule. By definition, there is an input $(X_0, \{A_i\})$ to the greedy sequencing rule that outputs $(X_0, T)$. Suppose we run the greedy sequencing rule on $(X_0, \{A_i\})$, and suppose for contradiction the verifier outputs $\text{False}$ at step $t$.

The following must be true: 
- $T_t, T_{t+1}, \ldots, T_{|T|}$ are not all orders of the same type, and

$x_{t-1} \geq x_{0.1}$ and $T_t$ is a sell order, or $x_{t-1} < x_{0.1}$ and $T_t$ is a buy order.

The first bullet implies $b_{\text{buy}}$ and $b_{\text{sell}}$ were not empty at step $t$. Thus whenever $x_{t-1} \geq x_{0.1}$, the greedy sequencing rule would pick $T_t$ to be a buy order and whenever $x_{t-1} < x_{0.1}$, the greedy sequencing rule would pick $T_t$ to be a sell order (and since $b_{\text{buy}}$ and $b_{\text{sell}}$ are non-empty, this is always possible). This contradicts the second bullet, and proves the verifier outputs $\text{True}$.

6 CONCLUSION

We have proposed a decentralized exchange framework where we model miners as intermediaries between a liquidity pool exchange and users, acting to choose which transactions to include in a block. The miners are free to pick the block content, but the execution ordering must be a feasible output from the greedy sequencing rule that we introduce and that they commit to implement. Our design does not require users to trust miners because we require that the execution ordering is verifiable, i.e., a polynomial time algorithm can certify if the execution ordering is a valid output from the
Greedy Sequencing Rule. Our proposal is backward compatible with the current implementations of liquidity pool exchanges: miners incur no computational overhead for implementing the sequencing rule and the sequencing rule verifier runs in polynomial time. From a practical side, our proposal can be operationalized through a relay service that commits to following our sequencing rule and operates a private transaction pool.

A common desideratum in DeFi protocols is to limit the miner utility from protocol manipulations (i.e., removing so-called miner extractable value). In this regard, we prove that no sequencing rule can prevent miners from obtaining risk-free profits in liquidity pool exchanges that have price impact: there are always instances where the miner receives strictly positive utility. Given this impossibility result, we focused on sequencing rules that provide provable guarantees for users in a system populated by self-interested miners. Our main finding is the Greedy Sequencing Rule, which ensures that for any user transaction $A$ that is executed, either (1) transaction $A$ receives an execution price as good as if $A$ was the only transaction in the block, or (2) transaction $A$ receives an execution price worse than this standalone price, but the miner does not gain when including $A$ in the block. Thus a bad execution price is due to competition for the same token between users, and not the result of manipulation from the miner.

Our work leaves open the following questions:

- Our Greedy Sequencing Rule explicitly assumes the liquidity pool exchange only pools $n = 2$ tokens. Although two token pools have the highest trading volume, other liquidity pool exchanges allow for liquidity pools with three or more tokens. Namely,Balancer [36] uses the product potential $\phi(X) = \prod_{i=1}^{n} X_i$ where one might have $n \geq 3$ tokens in the same pool. Which guarantees can verifiable sequencing rules provide for $n \geq 3$ token liquidity pool exchanges?
- We propose a verifiable sequencing rule with provable execution price guarantees for users under a simple utility model for miners, i.e., we assume that a miner prefers token basket over
- We show that for any sequencing rule, there is a set of user transactions that allow miners to extract risk-free profits. This result shows that welfare loss for users (i.e., the price of anarchy) is inevitable when miners are strategic. Define an optimal sequencing rule, i.e., a rule that maximizes user welfare under optimal miner deviations. Can we characterize the class of optimal sequencing rules?
- Our impossibility result only applies to deterministic sequencing rules. A randomized sequencing rule takes not only the initial state $X$ and the block content $B$, but also a random string $r$ (potentially unknown by the miner before picking $B$) and outputs a set system $S(X, B, r)$ with valid execution orderings. Are there randomized sequencing rules where miners cannot obtain risk-free profits?
- In a setting where a user creates two or more transactions $(B_i, B_j)$, they might wish to specify a constraint that $B_i$ executes before $B_j$. Can one replicate our results with sequencing rules that must preserve the user’s ordering constraints?

REFERENCES

[1] [n. d.]. DeFi Pulse - The Decentralized Finance Leaderboard. https://www.defipulse.com/
[2] [n. d.]. Flashbots. https://docs.flashbots.net/
[3] [n. d.]. Top stablecoin tokens by market capitalization. https://coinmarketcap.com/view/stablecoin/
[4] [n. d.]. Uniswap Overview. https://info.uniswap.org/#/
[5] 2020. Gnosis Protocol. URL: https://docs.gnosis.io/protocol/docs/devguide01/ (2020).
[6] 2022. https://ccaf.io/cche/index. Accessed: 2022-8-25.
[7] Hayden Adams, Noah Zinnmreiter, Moody Salem, River Keeser, and Dan Robinson. 2021. Uniswap v3 core.
[8] Mohammad Akharpour and Shengwu Li. 2020. Credible auctions: A trilemma. Econometrica 88, 2 (2020), 425–467.
[9] Guillermo Angeris and Tarun Chitra. 2020. Improved price oracles: Constant function market makers. In Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. 80–91.
[10] Guillermo Angeris, Alex Evans, Tarun Chitra, and Stephen Boyd. 2022. Optimal routing for constant function market makers. In Proceedings of the 23rd ACM Conference on Economics and Computation. 115–128.
[11] Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S Matthew Weinberg. 2019. Formal barriers to longest-chain-proof-of-stake protocols. In Proceedings of the 2019 ACM Conference on Economics and Computation. 459–473.
[12] Markus Brunnermeier, Andrew Crockett, Charles AE Goodhart, Avinash Persaud, Hyun Song, Shin, et al. 2009. The fundamental principles of financial regulation. Vol. 11. ICMB, Internat. Center for Monetary and Banking Studies Geneva.
[13] Eric Badish, Peter Cranton, and John Shin. 2015. The high-frequency trading arms race: Frequent batch auctions as a market design response. The Quarterly Journal of Economics 130, 4 (2015), 1547–1621.
[14] Agostino Capponi, Ruihze Jia, and Ye Wang. 2022. The evolution of Blockchain: from Lit to Dark. arXiv preprint arXiv:2202.05779 (2022).
[15] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. 2016. On the instability of bitcoin without the block reward. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 154–167.
[16] Jing Chen and Silvio Micali. 2019. Algorand: A secure and efficient distributed ledger. Theoretical Computer Science 777 (2019), 155–183.
[17] Tarun Chitra, Mathieu VX Ferreira, and Kshitij Kulkarni. 2023. Credible, Optimal Auctions via Blockchains. arXiv preprint arXiv:2301.12522 (2023).
[18] Philip Daian, Steven Goldfeder, Tyler Kell, Yungi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Bredenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 910–927.
[19] Philip Duan, Rafael Pass, and Elaine Shi. 2019. Snow white: Robustly configurable consensus and applications to provably secure proof of stake. In International Conference on Financial Cryptography and Data Security. Springer, 23–41.
[20] Michael Egorov. 2019. StableSwap-efficient mechanism for Stablecoin liquidity. Retrieved Feb 24 (2019), 2021.
[21] Meryem Essaidi, Mathieu V. X. Ferreira, and S. Matthew Weinberg. 2022. Credible, Strategyproof, Optimal, and Bounded Expected-Round Single-Item Auctions for All Distributions. In 13th Innovations in Theoretical Computer Science Conference (ITCS 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 215). Mark Braverman (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 66:1–66:19.
[22] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In International conference on financial cryptography and data security. Springer, 436–454.
[23] Zhou Fan, Francisco J Marmolejo-Cossio, Ben Altschuler, He Sun, Xintong Wang, and David Parkes. 2022. Differential Liquidity Provision in Uniswap v3 and Implications for Contract Design. In Proceedings of the Third ACM International Conference on AI in Finance. 9–17.
[24] Mathes V. X. Ferreira, Ye Lin Sally Hahn, S Matthew Weinberg, and Catherine Yu. 2022. Optimal Strategic Mining Against Cryptographic Self-Selection in Proof-of-Stamp. In Proceedings of the 23rd ACM Conference on Economics and Computation. 89–114.
[25] Mathes V. X. Ferreira, Daniel J Moroz, David C Parkes, and Mitchell Stern. 2022. Dynamic posted-price mechanisms for the blockchain transaction-free market. In Proceedings of the 3rd ACM Conference on Advances in Financial Technologies. 86–99.
[26] Mathes V. X. Ferreira and S Matthew Weinberg. 2020. Credible, truthful, and two-round (optimal) auctions via cryptographic commitments. In Proceedings of...
the 21st ACM Conference on Economics and Computation. 683–712.

[27] Matheus V. X. Ferreira and S Matthew Weinberg. 2021. Proof-of-Stake Mining Games with Perfect Randomness. In Proceedings of the 22nd ACM Conference on Economics and Computation. 433–453.

[28] Lioba Heimbach, Eric Schertenleib, and Roger Wattenhofer. 2022. Risks and Returns of Uniswap V3 Liquidity Providers. arXiv preprint arXiv:2205.08904 (2022).

[29] Lioba Heimbach and Roger Wattenhofer. 2022. Eliminating Sandwich Attacks with the Help of Game Theory. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. 153–167.

[30] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. 2020. Order-fairness for byzantine consensus. In Annual International Cryptology Conference. Springer, 451–480.

[31] Aggelos Kiayias, Elias Koutoupias, Maria Kyropoulou, and Yiannis Tselekounis. 2016. Blockchain mining games. In Proceedings of the 2016 ACM Conference on Economics and Computation. 365–382.

[32] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Annual international cryptology conference. Springer, 357–388.

[33] Sunny King and Scott Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August 19, 1 (2012).

[34] Kshitij Kulkarni, Theo Diamandis, and Tarun Chitra. 2022. Towards a Theory of Maximal Extractable Value I: Constant Function Market Makers. arXiv preprint arXiv:2207.11835 (2022).

[35] Ron Lavi, Or Sattath, and Aviv Zohar. 2022. Redesigning Bitcoin’s fee market. ACM Transactions on Economics and Computation 10, 1 (2022), 1–31.

[36] Fernando Martinelli and Nikolai Mushkian. 2019. A non-custodial portfolio manager, liquidity provider, and price sensor. URL: https://balancer.finance/whitepaper (2019).

[37] Conor McMenamin, Vanessa Daza, and Matthias Fitzi. 2022. FairTraDEX: A Decentralised Exchange Preventing Value Extraction. arXiv preprint arXiv:2202.06384 (2022).

[38] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review (2008), 21260.

[39] Michael Neuder, Rithvik Rao, Daniel J Moroz, and David C Parkes. 2021. Strategic Liquidity Provision in Uniswap v3. arXiv preprint arXiv:2106.12033 (2021).

[40] Andreas Park. 2021. The conceptual flaws of constant product automated market making. Available at SSRN 3805350 (2021).

[41] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying blockchain extractable value: How dark is the forest? In 2022 IEEE Symposium on Security and Privacy (SP). IEEE, 198–214.

[42] Tim Roughgarden. 2021. Transaction Fee Mechanism Design. In Proceedings of the 22nd ACM Conference on Economics and Computation (Budapest, Hungary) (EC ’21). Association for Computing Machinery, New York, NY, USA, 792.

[43] Ayelet Sapirstein, Yonatan Sompolinsky, and Aviv Zohar. 2016. Optimal selfish mining strategies in bitcoin. In International Conference on Financial Cryptography and Data Security. Springer, 515–532.

[44] Ben Weintrub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. 2022. A flash (bot) in the pan: measuring maximal extractable value in private pools. In Proceedings of the 22nd ACM Internet Measurement Conference. 458–471.

[45] Luyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2021. High-frequency trading on decentralized on-chain exchanges. In 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 428–445.

Received 2022-11-07; accepted 2023-02-06