

# SoK: Consensus for Fair Message Ordering

Zhuolun Li

*School of Computer Science  
University of Leeds  
Leeds, UK  
sczl@leeds.ac.uk*

Evangelos Pournaras

*School of Computer Science  
University of Leeds  
Leeds, UK  
e.pournaras@leeds.ac.uk*

**Abstract**—Distributed ledger systems, such as blockchains, rely on consensus protocols that commit ordered messages for processing. In practice, message ordering within these systems is often reward-driven. This raises concerns about fairness, particularly in decentralized finance applications, where nodes can exploit transaction orders to maximize rewards referred to as Maximal Extractable Value. This paper provides a systematic understanding of consensus protocols that order messages with different approaches, especially focusing on the ones that promote order fairness, using methods including First-In-First-Out (FIFO), random, and blind ordering. We review the challenges and trade-offs of deriving fair message ordering in a Byzantine fault-tolerant setting, and summarize the requirements for making a fair message ordering consensus protocol. We introduce a design guideline, with which we propose a latency optimization to the state-of-the-art FIFO ordering protocol of Themis [1]. This work provides a systematic way for assessing and enhancing message order fairness in blockchain systems.

**Index Terms**—blockchain, consensus, ordering, fairness

## 1. Introduction

Consensus protocols in blockchain systems establish agreements on the messages (or transactions, used interchangeably in this paper) to process in some specific order. Decentralized applications, particularly those utilizing smart contracts, often result in different states if messages are processed in different orders. Since nodes in blockchains have significant freedom to censor the messages and decide on an order, they can manipulate the states of decentralized applications by altering the order of message processing. In practice, nodes frequently manipulate message ordering to maximize rewards. For example, in major blockchains such as Bitcoin [2] and Ethereum [3], block proposers typically favor transactions with higher fees. Consequently, transactions with lower fees experience longer delays, leading to fairness concerns [4].

More importantly, in decentralized finance (DeFi) applications that rely on smart contracts, nodes have economic incentives to manipulate the order of message processing to

generate additional profits known as Maximal Extractable Value (MEV), often resulting in substantial financial losses for users [5]. For example, a node can create a transaction to buy an asset and prioritize it if the node observes a transaction in the message pool that can raise the asset price once processed. As a result of such profiting strategies through transaction ordering manipulation, around 200 million USD are extracted from DeFi users every year [5].

Although there are multiple definitions and approaches that label message order fairness in existing literature, a commonality among these definitions is that they are designed to prevent MEV in blockchains, where a single node in the system can decide message ordering out of its own interest.

The core challenge in deriving a fair ordering of messages in distributed ledger systems is the state asynchronization across nodes. As shown in Figure 1, in decentralized networks, each node can independently receive messages from external users, resulting in differing local views of the message pool. Even if all nodes eventually receive the same set of messages by constantly exchanging their message pool, they are likely to receive them at varying times and in differing orders due to network delay and asynchrony [6], making it difficult for the nodes to reach an agreement of a fair message processing order. An easy way to prevent asynchronous message pools is to have a centralized entity receiving messages [7], but it violates the nature of decentralization. Moreover, most distributed ledger systems require Byzantine fault tolerance (BFT), which adds additional considerations to the robustness of the fair ordering rule in designing a fair ordering protocol.

Awareness of the message ordering problem has arisen in the past five years and has shown a growing interest as the number of related works has grown. While there are proposals for fair ordering consensus under network asynchrony and Byzantine-fault nodes, a comprehensive understanding of the challenges, and design choices of these systems is yet to be developed. Therefore, we present the first systematization of knowledge of fair ordering consensus protocols by examining existing fair ordering consensus protocols. The key contributions of our work are as follows:

- 1) We review proposals for message ordering and

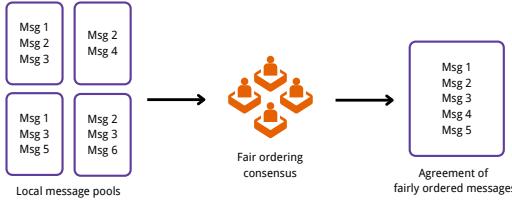


Figure 1. An example consensus protocol that achieves FIFO order agreements among the four nodes. The nodes agree to commit messages in the order of message 1, 2, 3, 4, and 5. The protocol does not guarantee to commit all messages in one agreement. In this example, message 6 is uncommitted in this round of agreement; it stays in the local message pool to get committed in upcoming rounds of agreement.

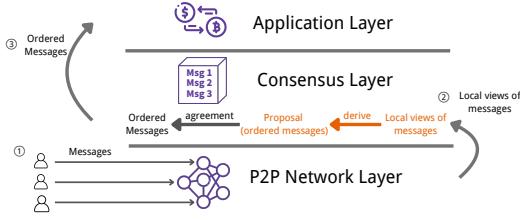


Figure 2. Message flow in distributed ledgers. Users send messages to nodes, which then commit an ordered batch through consensus.

offer a unified perspective on the requirements and limitations of using these ordering rules.

- 2) We introduce the “Fair Consensus Factory”, a framework to integrate message order fairness into consensus protocols.
- 3) A case study of the state-of-the-art FIFO ordering protocol, Themis [1]. Through the Fair Consensus Factory, we propose an alternative design of Themis that reduces latency.
- 4) Key research gaps identified as future directions for achieving fair and efficient decentralized systems.

The remaining paper is structured as follows: Section 2 provides background information on current message ordering approaches and MEV issues in distributed ledgers. Section 3 presents a unified perspective on existing message ordering rules. Section 4 presents the Fair Consensus Factory that adds fairness to consensus protocols. Section 5 provides a case study of Themis regarding its consensus design and a latency optimization provided by the fair consensus factory framework. Section 6 discusses the open challenges in fair ordering consensus for future studies. Finally, Section 7 concludes the paper.

## 2. Background

### 2.1. Problem Setting

Distributed ledgers rely on a set of nodes to receive, relay, and agree on an ordered sequence of messages for execution at the application layer (Figure 2). Messages may originate from external clients or be generated internally

by nodes. In each consensus round, a new ordered batch of messages is committed. This order determines the state transitions and application-level behavior.

We consider a setting in which nodes have equal opportunity to receive messages and participate in the ordering process. That is, the protocol assumes no geographic, topological, or infrastructural advantage among nodes. However, nodes are rational and may deviate from protocol behavior if doing so increases their individual utility, especially when granted special roles such as the consensus leader.

We adopt the standard Byzantine fault-tolerant (BFT) model. Among  $n = 3f + 1$  participating nodes, up to  $f$  may behave arbitrarily, including collusion and deviation from the protocol. Byzantine nodes may deviate arbitrarily from the protocol without preventing honest nodes from reaching consensus. Broadly speaking, this message-ordering challenge applies to any general model of Byzantine fault-tolerant state machine replication [8] systems that require consensus on message ordering.

### 2.2. Message Ordering in Existing Distributed Ledgers

In most blockchains and DAG-based systems, consensus protocols must determine not just which messages to include but also in what order to process them. Traditionally, Byzantine consensus protocols guarantee agreement on total order, but do not constrain how that order is chosen.

Li et al. [9] surveyed ordering policies in existing distributed ledger systems and found significant variation. Some systems prioritize by transaction fee, others by local reception time or message origin. However, these policies are implementation-level defaults rather than enforced rules. Individual nodes can and often do modify their ordering behavior in pursuit of profit.

For example, Avalanche [10], a DAG-based system, adopts a FIFO policy by default. Yet this behavior is not protocol-enforced—rational nodes can replace the FIFO strategy with custom rules if doing so yields financial gain. Because other nodes cannot detect or challenge such deviations, the final order remains valid under the protocol, even if manipulated.

### 2.3. Maximal Extractable Value (MEV)

This ability to manipulate message order gives rise to Maximal Extractable Value (MEV) [5], [11], [12], the profit a node can extract by controlling transaction sequencing. MEV is especially prevalent in decentralized finance (DeFi), where transaction order can influence token prices, arbitrage opportunities, and the outcome of smart contract interactions.

Nodes may front-run or back-run user transactions, placing their own strategically crafted messages before or after victim transactions to extract value [12]. Nodes may also observe transactions with profit potential, copy them, and submit imitated versions with higher priority [13].

Although some researchers argue that MEV can improve market efficiency [14], the dominant view is that MEV constitutes an attack vector and undermines fairness [11]. While application-level defenses exist [15], [16], they do not address the underlying vulnerability: that the message ordering mechanism is manipulable.

## 2.4. Permissioned and Permissionless Fair Ordering

In a permissionless setting for consensus protocols, the set of participating nodes is open and dynamic, meaning that nodes can join or leave the network frequently. In contrast, permissioned settings assume a fixed or controlled set of participants.

In the context of fair ordering protocols, there is no widely accepted definition of what it means for a fair ordering protocol to be permissionless. The closest formal notion is player replaceability [17], which captures the idea that participants in the consensus can be replaced without affecting the protocol’s security or fairness guarantees.

In practice, many fair ordering protocols designed for permissioned settings can be adapted to permissionless environments through techniques such as committee rotation or network reinitialization [18]. For example, a system can periodically select a new subset of nodes to participate in consensus or reconfigure itself when nodes join or leave. This transformation is primarily a matter of efficiency, not of fundamental design.

The main challenge in adapting fair ordering to a permissionless setting lies in the cost of reinitialization, especially when the protocol relies on threshold cryptographic schemes that require expensive bootstrap procedures such as distributed key generation. In such cases, frequent committee changes can introduce significant overhead. Nevertheless, this issue can be mitigated by reducing the frequency of committee reconfiguration, striking a balance between security and performance. Given these observations, we do not draw a strict distinction between permissioned and permissionless fair ordering protocols in this work.

## 3. Message Ordering Approaches and Objectives

This section formalizes different message ordering rules in the Byzantine fault-tolerant setting and discusses in detail the requirements of executing these ordering rules and their limitations.

We define a message ordering rule as a function that sorts a set of messages into a specific sequence based on certain metadata. Let  $\mathcal{S}^m$  and  $\mathcal{O}^m$  be the family of the sets of  $m$  unordered messages and the lists of ordered messages ( $|S| = |O| = m$ ), respectively. Let  $\mathcal{D}$  denote the set of metadata used to order messages (varies depending on the

actual ordering rule). Now we define a *message ordering rule function*:

$$\begin{aligned} r : \mathcal{S}^m \times \mathcal{D} &\longrightarrow \mathcal{O}^m \\ (S, D) &\longmapsto O \end{aligned}$$

That is, a message ordering rule function orders a set of messages  $S$ , which comes from the aggregation of all local message pools of the network nodes, using some metadata  $D$  and outputs the messages in an ordered list. The output  $O$  is a permutation of  $S$ , i.e., it contains the same elements as  $S$  but arranged in a specific order. Below, we discuss in detail the types of ordering rules  $r$  and the corresponding metadata  $D$  used to decide the ordering of the messages using these rules.

### 3.1. FIFO Ordering Rule

The most-studied ordering rule is FIFO ordering [1], [4], [17], [19]–[36], which seeks to process messages at a first-come-first-commit manner.

As first mentioned by Kelkar et al. [19], it is impossible to measure the time when external messages are *sent* to the system without making additional assumptions on the network status of message senders or having independent trustworthy timestamping services. Thus, FIFO ordering primarily targets processing messages as they are first *received* by the system nodes.

However, since messages can be received by different nodes at different times and orders at the network layer, the major challenge of FIFO ordering rules is to derive a fair global message order using these different local views. Moreover, in the Byzantine fault-tolerant setting, the rules have to consider that up to  $f$  local views are missing or indistinguishably malicious. The two main methods to achieve FIFO ordering under these conditions are ranked voting and median timestamping.

**3.1.1. FIFO via Ranked Voting.** The process of collectively agreeing on an order when each voter node has a different local order preference can be modeled by ranked-choice voting. Therefore, there are some proposals to derive the order of messages by collecting ranked local message arrival orders from each individual node and aggregating them [1], [17], [19], [21]–[24], [37]. Different from normal ranked-choice voting aggregation, the algorithms have to account for that not all messages are received by all nodes, and therefore the local rankings are incomplete from the global perspective.

Given an unordered message set  $S$ , let  $O_i^S$  be the ordered list of messages based on the arrival order of messages locally observed by node  $i$ . FIFO order via voting is defined as the following ordering rule:

$$\text{FIFO}(S, [O_1^S, O_2^S, \dots, O_n^S])$$

In FIFO via ranked voting, each node’s local order of messages acts as a “ballot” that ranks the messages, and the final ordering is decided by majority preference among



Figure 3. An example of a Condorcet cycle from ranked voting. Among the three nodes, two nodes receive message 1 before message 2, two nodes receive message 2 before message 3, also two nodes receive message 3 before message 1.

nodes. However, this method may not always yield a total FIFO order due to the cases of draw Condorcet cycles [38], where cyclic preferences among nodes are found [19]. Figure 3 is a demonstration of how a Condorcet cycle is formed in ranked voting. In this example, the order of the three messages is indistinguishable because the result of the ranked voting is a draw. Even worse, such cycles can be infinitely extended to involve an arbitrary number of messages. This raises an attack opportunity of deliberately creating or extending Condorcet cycles in the case where malicious external message senders submit messages to different nodes at carefully designed times [25].

To tolerate Condorcet cycles, different relaxed notions of order fairness are proposed. Block order fairness [19] (later referred to as batch order fairness) and similarly differential order fairness [21] are proposed that allow multiple messages to be deemed as received at the same time. In this case, any ordering among these messages is deemed acceptable. For completeness, the definition is provided below:

**Definition 1 ( $\gamma$ -Batch Order Fairness).** Let  $m_1$  and  $m_2$  be two messages that are both received and processed by all honest nodes. Let each honest node maintain a local reception order over messages. We say that a protocol satisfies  $\gamma$ -batch order fairness for some  $\gamma \in (0.5, 1]$  if the following holds: If at least a  $\gamma$ -fraction of honest nodes observed  $m_1$  before  $m_2$  in their local reception order, then all honest nodes output  $m_1$  no later than  $m_2$ .

However, this definition is under the setting that messages can be output in batch, in which messages are regarded as indistinguishable in their order. This does not apply to the setting of blockchain, where a total ordering must be derived. Practically, this means block proposers still have the freedom to order messages in batches to maximize personal utilities.

Moreover, as Condorcet cycles can be arbitrarily long, making sure messages in Condorcet cycles commit at the same time also impacts the liveness of the protocol [19]. A solution to this problem is to assume an external synchrony time bound [19], so that when the time of receiving two messages is over the time bound, they must not be in a Condorcet cycle. The drawback of this method is the high latency to commit a message and the requirement of an additional external synchrony assumption. To improve on that, one way is to use ranked pairs [39] to break Condorcet cycles without violating FIFO order fairness [25], [26]. Themis [1] also introduces another method called batch-unspooling that commits messages that are part of a cycle

TABLE 1. AN EXAMPLE OF TAKING THE MEDIAN TIMESTAMP TO ORDER MESSAGES

Message	Node 1	Node 2	Node 3	Output
Msg 1	3:00	3:02	3:04	3:02
Msg 2	3:01	3:03	3:05	3:03

in a round without requiring observing the full cycle, and allows remaining messages in the cycle to commit in following rounds.

To summarize, this line of research attempts to handle Condorcet cycles in a way that minimizes their effect on latency. However, taking a step back, improvements can also be made by actively preventing the formation of Condorcet cycles. Therefore, a potential research direction that can advance FIFO via voting proposals is to reduce the probability of forming Condorcet cycles, for example, by requiring more frequent message propagation [25].

**3.1.2. FIFO via Timestamping.** Another type of proposal for FIFO ordering is to assign a timestamp to each message and order based on the assigned timestamps [27]–[36], [40].

Based on the existing proposals, there are two ways to timestamp a message. The first way is to make Byzantine agreements using the median value of the timestamps that the nodes locally receive for a message [27]–[30], [32], [33], [36]. The median value is used because it eliminates the effect of the malicious timestamp proposals from the Byzantine nodes. The median timestamp is always within the range of time the message was received by the honest nodes.

Given a set of messages  $S$ , let  $T_i^S$  denote the map of timestamps at which every message of  $S$  is locally observed by node  $i$ . FIFO ordering via median timestamping is defined as the following ordering rule:

$$\text{FIFO}(S, [T_1^S, T_2^S, \dots, T_n^S])$$

Using the median timestamp  $t_m$  implies that it comes from one single node (or two nodes) in the network. Table 1 provides an example scenario of FIFO via median timestamping. Based on the decided timestamp, message 1 is ordered before message 2. However, a problem with using timestamps from a single node (the median) is that the node can manipulate the relative ordering of messages. If node 2 is malicious, it can change the order of the messages simply by claiming message 1 is received at 3:03, and message 2 is received at 3:02. Although it is believed that such manipulation is difficult in practice for an adversary [27], [28] as it requires an adversary to submit a timestamp that is exactly the median, the possibility of such manipulation cannot be fully eliminated.

Existing definitions [1], [27], [28] attempting to formalize the fairness guarantee provided by the median timestamping approach do not precisely cover the above scenario. Kelkar et al. [1] conclude that, in this approach, if all honest nodes receive a message  $m_1$  before any honest node receives another message  $m_2$ , then all honest nodes output  $m_1$  before  $m_2$ . However, this is not the case if a malicious node swaps

the median timestamps of the two transactions. Therefore, we provide a definition that captures the fairness guarantee provided by this approach:

**Definition 2 (*f*-Robust Median Order).** Let  $n$  be the number of nodes and  $f$  the maximum number of Byzantine nodes. Let  $\mathcal{T}(a)$  and  $\mathcal{T}(b)$  be the multisets of timestamps for transactions  $a$  and  $b$ , respectively, with  $n$  entries each. Let  $\text{ts}(a)$  denote the median of  $\mathcal{T}(a)$ .

We say that the relative order  $\text{ts}(a) < \text{ts}(b)$  is *f*-robust if:

$$T_{(n/2+f+1)}(a) < T_{(n/2-f)}(b)$$

where  $T_{(i)}(a)$  denotes the  $i$ -th smallest timestamp in  $\mathcal{T}(a)$ , and likewise for  $b$ .

This definition accounts for an adversary controlling up to  $f$  timestamps per transaction that can reverse the order by manipulating the median, and only guarantees fair ordering between messages that are not vulnerable to such an attack, given that they share fewer overlapping timestamps.

Moreover, whenever  $t_m$  is used to compare with timestamps from any other nodes in the network, there must be an assumption that these two local clocks are synchronized, otherwise the comparison does not provide any conclusions. Since all existing median timestamping proposals involve cross-node timestamp comparisons, they all require synchronized clocks. This does not mean median timestamping methods are infeasible, as BFT clock synchronization protocol exists and it can be applied to ensure synchronized clocks [41], [42].

Byzantine agreement of the median value given a set of values has been extensively studied [43], and it is also used in this line of research to find out the best median value among the honest nodes [36]. However, we argue that the problem here is somewhat different from finding a median value. Instead, the aim of this problem is to find the minimum honest value, as it is the closest timestamp to when the message is actually sent. This brings up a research gap that has not yet been studied, that is, how accurate the timestamps derived from these protocols are from the actual time when the message is sent by the sender, and how to minimize the deviation between the actual and derived timestamps.

Apart from median timestamping, another method to derive timestamps for messages is to infer the message arrival timestamp according to the known network delay between the peers [34], [35]. This requires an additional assumption that the network delay remains relatively stable.

Given a set of messages  $S$ , let  $L$  denote a mapping of network latency from all combinations of source and destination nodes.  $I_{src}$  is the list of source nodes that broadcast  $S$ . FIFO ordering via inference timestamping is defined as:

$$\text{FIFO}(S, L, I_{src})$$

In this category of approach, both proposals [34], [35] assume a message is broadcast from one node to all other nodes. In the case of unstable network delay and the network

is not synchronized, nodes can not reach an agreement because the resulting timestamping is not within the acceptable range of their expectation, therefore liveness is lost. Given such strict conditions, the scalability and resilience of this approach are yet to be verified.

**3.1.3. Comparison of FIFO Protocols.** A comparison of FIFO protocols can be found in Themis [1]. We provide a more comprehensive version of such a comparison in Table 2. The table only includes proposals from which clear, fair ordering algorithms with fairness guarantees are proposed.

Proposals based on FIFO via voting incur higher computational complexity in their ordering algorithms. This is primarily due to the need to construct a dependency graph of messages, based on the local orderings reported by all participating nodes. In contrast, timestamping-based approaches tend to have lower ordering complexity, as their primary computational bottleneck lies in sorting timestamps.

It is also important to note that voting-based proposals generally tolerate fewer Byzantine faults than their timestamping counterparts, even under the most relaxed fairness parameter (i.e., when  $\gamma = 1$ ). Both Aequitas [19] and Themis [1] can tolerate up to one-fourth of the nodes being Byzantine. This limitation arises from the fairness condition, which requires that at least  $\gamma n$  nodes observe one message before another for the ordering to be considered fair. Since up to  $f$  of these nodes may behave maliciously and lie about their observations, the protocol must ensure that  $\gamma n - f > \frac{n}{2}$  to maintain correctness, thus limiting the fault tolerance.

## 3.2. Random Ordering Rule

**3.2.1. Methods for Random Ordering.** Another approach of fair ordering is to randomly order messages [44]–[50].

Given a set of messages  $S$ , denote  $R$  as the randomness agreed upon by all nodes. Random ordering is defined as:

$$\text{Random}(S, R)$$

The core of random ordering is to obtain a randomness that all nodes agree on. The existing proposals [44]–[50] use information from the previous blocks as the public randomness. To convert randomness into a message selection criterion, a hash function is used to combine the randomness and the messages to output a pseudo-random value for each message at each message selection round. Then the messages with the lowest pseudo-random values are selected.

Early proposals for random ordering [44]–[47] are non-deterministic because these proposals do not require a unified set of messages  $S$  as the input of the ordering rules. A proposer derives a randomly ordered message set using its own message pool, and the peers verify the proposal by inferring the probability of the proposer being honest given their own message pools. For example, if too many messages in the proposal are not known by a peer, this peer probabilistically rejects the proposal. The order derived from these proposals, even though agreed by all nodes, may still contain maliciously constructed messages with

TABLE 2. COMPARISON OF FIFO ORDERING CONSENSUS PROTOCOLS ACROSS DESIGN, ASSUMPTIONS, AND PERFORMANCE DIMENSIONS

Proposal	Design	Ordering Rule	Ordering Guarantee	Security Byzantine Fault Tolerance	Assumptions	Performance
	Leader-Based/Leaderless			Network Model	Sync. Clocks	Comm. Complexity Ordering Complexity
Aequitas [19]	Leaderless	Voting	$\gamma$ -batch order fairness	$n \geq \frac{2f(\gamma+1)}{2\gamma-1}$ (weak liveness) <sup>a</sup>	Asynchronous	$O(n^3)$
Quick fairness [21]	Leaderless	Voting	$\gamma$ -batch order fairness	$0^b$	Asynchronous	$O(n^2)$
Themis [1]	Leader-based	Voting	$\gamma$ -batch order fairness	$n \geq \frac{2f(\gamma+1)}{2\gamma-1}$	Partially synchronous	$O(r^2)$
Pompé [27]	Leader-based	Timestamping	$f$ -Robust Median Order	$n \geq 3f + 1$	Partially synchronous	$O(r^2)$
Wendy [28]	Leader-based <sup>c</sup>	Timestamping	$f$ -Robust Median Order	$n \geq 3f + 1$	Asynchronous <sup>d</sup>	$O(n \log n)$
Hashgraph [30]	Leaderless	Timestamping	$f$ -Robust Median Order	$n \geq 3f + 1$	Asynchronous	$O(n)$

Note:  $n$ : the number of nodes in the network;  $f$ : the number of Byzantine faults;  $m$ : the number of messages in a round of consensus.

<sup>a</sup> In the case of a long Condorcet cycle, Aequitas does not provide a liveness guarantee even when all nodes are honest.

<sup>b</sup> The proposal loses liveness in the presence of any Byzantine nodes.

<sup>c</sup> Wendy is only a pre-protocol for a consensus protocol, but this pre-protocol itself requires a leader.

<sup>d</sup> Also depends on the specific consensus protocol Wendy integrates with.

specific hashes to fit in a designated order position. With this approach, no deterministic fair ordering guarantee can be formulated based on this approach.

To prevent so, recent work proposes to randomly permute a set of committed messages for execution [48]–[50], providing a deterministic way of enforcing randomness. We provide a formal definition of the ordering guarantee produced by such a permutation approach.

**Definition 3 (Permuted Ordering).** Let  $M = \{m_1, m_2, \dots, m_n\}$  be a set of  $n$  committed messages, and let  $\sigma : M \rightarrow \{1, \dots, n\}$  be a random bijection drawn uniformly at random from the set of all permutations over  $M$ . The protocol enforces a *permuted ordering* if for all distinct messages  $a, b \in M$ , the following holds:

$$\Pr[\sigma(a) < \sigma(b)] = \Pr[\sigma(b) < \sigma(a)] = \frac{1}{2}.$$

**3.2.2. Comparison of Random Ordering Protocols.** Table 3 lists four random ordering proposals. The permutation-based approach offers the lowest ordering complexity among fair ordering protocols. It avoids the overhead of constructing dependency graphs or sorting timestamps, making it computationally lightweight. Additionally, it does not rely on synchronized clocks. Prior work [48], [50] has also demonstrated that permutation schemes can be easily integrated with a variety of consensus protocols, highlighting their modularity and practicality. These properties make the permutation approach a lightweight solution against MEV.

### 3.3. Blind Ordering Restriction

Another line of research is blind ordering [35], [44], [45], [51]–[61], [61], [62], [62], [63], [63]–[65]. These proposals use threshold cryptography [66], [67] to encrypt the messages before they are committed so that nodes are unable to read the content of the messages when ordering. It requires message senders to perform encryption that allows only a threshold of the nodes (e.g. three out of four) to collectively reveal the messages after they are committed. Except for threshold cryptography, computational time lock puzzles can also be used to keep messages encrypted before committing [68]. However, the time lock puzzles have

the intrinsic disadvantage of having unstable puzzle-solving time [69], meaning that messages can be revealed before ordering. Therefore, encrypting with time lock puzzles is less secure.

Strictly speaking, this approach on its own is not an ordering rule as it still allows nodes to arbitrarily order messages, therefore, it is considered rather a restriction than a rule. It can be orthogonally applied along with random ordering [44]–[47], [49], [50] or FIFO ordering [31], [35], [36].

Blind ordering restriction promotes order fairness by ensuring censorship resistance, as messages are encrypted before committing to the network. Moreover, blind ordering is, by far, the only effective protection against local reordering in the sense that the nodes are not able to find profit opportunities that motivate them to locally reorder messages.

## 4. Fair Consensus Factory: Building Efficient Fair Ordering Consensus Protocols

Inspired by the existing proposals that decouple fair ordering algorithms with consensus protocols [22], [28], [48], [50], the design of fair ordering consensus protocols can be improved by treating fair ordering as a modular refinement to existing consensus systems. We introduce the *Fair Consensus Factory*, a framework that systematically transforms Byzantine fault-tolerant consensus protocols to enforce fair message ordering. This section formalizes this idea as a protocol transformation, outlines the design space of ordering rules, and presents a transformation-based method for adding order fairness into both leader-based and leaderless consensus protocols.

### 4.1. Adding Order Fairness to Consensus Protocol

Let  $\mathcal{P}$  denote the class of Byzantine fault-tolerant consensus protocols that satisfy standard safety and liveness properties, but do not satisfy any order fairness property in Definition 1, 2, 3. Each protocol  $P \in \mathcal{P}$  assumes a distributed set of nodes, a message dissemination layer, and a decision procedure that determines a sequence of committed transactions.

Given a deterministic fair ordering rule  $r$ , the Fair Consensus Factory transforms  $P$  into  $P_{\text{fair}}$ , such that  $P_{\text{fair}}$  is a

TABLE 3. COMPARISON OF RANDOM ORDERING CONSENSUS PROTOCOLS ACROSS DESIGN, ASSUMPTIONS, AND PERFORMANCE DIMENSIONS

Proposal	Design		Ordering Guarantee	Security	Assumptions		Performance	
	Leader-Based/Leaderless	Ordering Rule			Network Model	Sync. Clocks	Comm. Complexity	Ordering Complexity
Helix [44]	Leader-based	Random Selection	Probabilistic random selection	$n \geq 3f + 1$	Synchronous	No	$O(n^2)$	$O(m \log m)$
$\Pi^3$ [48] <sup>a</sup>	Leader-based	Permutation	Permuted Ordering	N/A	N/A	No	N/A	$O(m)$
MEVade [50] <sup>a</sup>	Leader-based	Permutation	Permuted Ordering	N/A	N/A	No	N/A	$O(m)$
BlindPerm [49]	Leader-based	Permutation	Permuted Ordering	$n \geq 3f + 1$	Partially synchronous	No	$O(n^2)$	$O(m)$

Note:  $n$ : the number of nodes in the network;  $f$ : the number of Byzantine faults;  $m$ : the number of messages in a round of consensus.

<sup>a</sup>  $\Pi^3$  and MEVade are modules that can be adapted to different leader-based protocols, not an actual consensus protocol.

protocol in which message ordering adheres to a deterministic, fair ordering function  $r$  shared by all honest nodes.

This transformation is realized through two transformation rules applied to  $P$ :

**Transformation Rule 1: State Synchronization.** In standard consensus protocols, nodes may maintain inconsistent views of pending messages due to asynchronous delivery, selective dissemination, or adversarial interference. To enforce fairness, we introduce a synchronization step that ensures all honest nodes agree on the input set and associated metadata prior to ordering. Specifically, honest nodes must establish a common view of the candidate message set  $S$  and its metadata  $D$  through explicit communication or implicit dissemination during consensus rounds. This shared state is a precondition for fair ordering, and prevents adversaries from biasing the outcome by withholding or selectively propagating messages.

**Transformation Rule 2: Local Rule Execution.** Once a synchronized view of  $S$  and  $D$  is established, each honest node independently applies the fair ordering function  $r$  to produce an ordered output sequence. Because  $r$  is deterministic and operates on agreed-upon inputs, all honest nodes produce the same result without relying on a centralized proposal or aggregation phase. This eliminates opportunities for order manipulation and aligns the protocol output with a predefined fairness criterion.

The protocol no longer relies on arbitrary leader proposals or aggregation heuristics to determine message order. Instead, fairness is achieved by synchronizing input views and applying a locally computable, globally consistent rule.

The transformation only changes how  $P$  derives a proposal to agree on, but it does not change how  $P$  reaches agreement on the proposal.

## 4.2. Choosing Ordering Rules

The factory supports a variety of fair ordering rules  $r$ , selected depending on desired fairness criteria and system assumptions. Figure 4 shows options for ordering rules. A consensus can either use random ordering, FIFO via ranked voting, or FIFO via timestamping, with an optional add-on of requiring the nodes to apply the ordering rule without knowing their content. Details of these options are covered in the previous section.

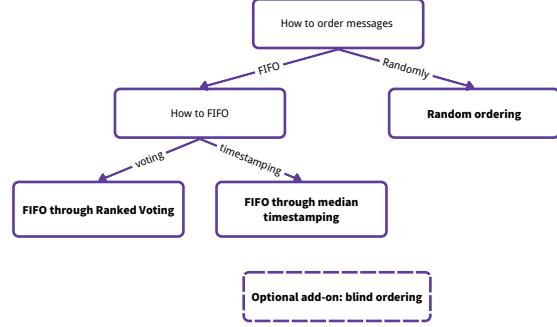


Figure 4. Fair Consensus Factory menu: selection of ordering rules

## 4.3. Application to Leader-Based and Leaderless Protocols

The Fair Consensus Factory applies to both leader-based and leaderless protocols by specializing in how transformation rules are embedded into the protocol logic. Figure 5 compares the transformation applied to each protocol family.

**Leader-Based Consensus.** A leader-based consensus protocol (e.g., HotStuff [70], Alea [58]) proceeds through the following phases per round:

- 1) **Proposal:** A designated leader collects a set of pending transactions  $S$  from its local pool and produces an ordered sequence  $O$  from an arbitrary ordering policy.
- 2) **Agreement:** The leader broadcasts the proposal  $O$  to other nodes. If they validate it, they vote to accept it, and the protocol reaches consensus on  $O$ .

The Fair Consensus Factory transforms this process as follows:

- At the beginning of the consensus round (e.g., during leader election) or at the last phase of the last consensus round, all nodes exchange  $S_i$  and  $D_i$  to synchronize the input set  $S$  and associated metadata  $D$ .
- Once  $S$  and  $D$  are agreed upon, each honest node independently computes  $O = r(S, D)$  using the fair, deterministic ordering rule  $r$ .
- The result  $O$  is passed into the standard voting and agreement phases of the protocol, unmodified.

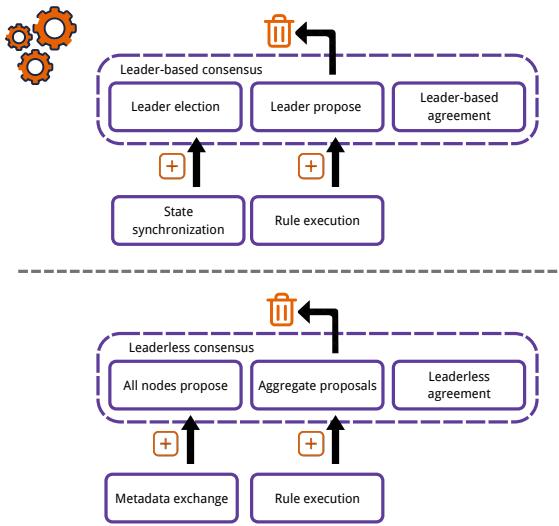


Figure 5. Transformation of leader-based (top) and leaderless (bottom) consensus via the Fair Consensus Factory

For leader-based consensus, state synchronization can often be amortized or embedded in leader election or finalization messages of the previous consensus round, minimizing latency overhead.

**Leaderless Consensus.** Leaderless consensus protocols (e.g., Narwhal/Bullshark [71], Avalanche [10], Mysiticeti [72]) typically follow a different structure:

- 1) Exchange proposals: Each node  $i$  proposes a local input set  $S_i$  of messages.
- 2) Aggregation: The protocol aggregates proposals  $S_1, \dots, S_n$  into a combined set  $S$ , and applies an predefined ordering policy to derive a sequence  $O$ .
- 3) Agreement: Nodes reach consensus on  $O$  using the standard agreement procedure of the protocol.

This process already satisfies the communication requirements of the transformation rules, therefore, the Fair Consensus Factory only modifies message exchange and the ordering rule as follows:

- The exchange proposal phase is also used to exchange metadata  $D_i$ . As a result, honest nodes synchronize on a common input set  $S$  and metadata  $D$ .
- The originally defined aggregation rule is replaced by the fair ordering rule  $r$ . Each node locally computes  $O = r(S, D)$ .
- The agreed-upon output  $O$  is then passed into the standard agreement procedure of the protocol.

#### 4.4. Preservation of Safety and Liveness

The Fair Consensus Factory transformation preserves the safety and liveness properties of the underlying consensus protocol.

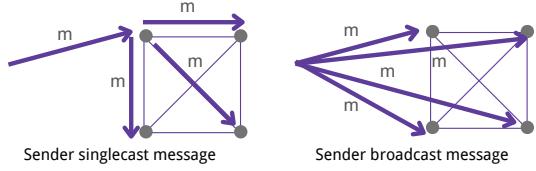


Figure 6. Two message dissemination modes: one-to-one (left), and broadcast-to-all (right)

**Safety.** Safety in consensus protocols ensures that no two honest nodes commit different outputs. Our transformation maintains the original protocol’s agreement mechanism, such as voting or quorum certificates, which guarantees this property.

**Liveness.** If a Byzantine node sends inconsistent views of  $(S, D)$  to different honest nodes during synchronization, it may cause the nodes to compute divergent ordered outputs. Since the agreement logic remains unmodified, safety is preserved, i.e., no two honest nodes will commit different outputs. However, this divergence can prevent the protocol from reaching consensus in the current round, potentially leading to the failure of agreement for this consensus round.

However, this is not a liveness concern as it is mitigable in the next consensus round. In the next round, honest nodes can reshare local views shared in the last round during state synchronization, and additionally exchange hashes of the local input views  $h(S_i, D_i)$  they receive from other nodes in the previous round. If a node is found to have equivocated, its input can be excluded from the new round. This restores consistency of the locally derived proposal, and allows the protocol to proceed normally.

#### 4.5. Practical Considerations and Limitations

A key limitation arises when messages are not sufficiently disseminated before consensus. If a message  $m$  exists in fewer than  $f + 1$  honest nodes’ local pools, Byzantine nodes may suppress  $m$  during synchronization or lie about its metadata, breaking the fairness guarantees.

To mitigate this, some protocols [1], [21] require that clients send messages to all nodes (Figure 6), or assume bounded message propagation delay [19]. Another partial mitigation is to apply blind ordering: nodes are required to execute  $r$  without knowing message content, removing the incentive for MEV-style reordering even if unfair metadata is injected.

Finally, we note that recent proposals suggest relying on trusted execution environments (TEEs) [54], [73], [74] or zero-knowledge proofs [1] to enforce ordering from a single node without requiring all-to-all synchronization. While promising, these approaches introduce trust or computational costs that may conflict with decentralization or

latency goals<sup>1</sup>. For this reason, our proposed framework prioritizes solutions based on all-to-all synchronization and deterministic execution.

## 5. Case Study: Themis

Themis [1] is a consensus protocol designed to achieve fair ordering through a FIFO-via-voting mechanism. It has been recognized as a promising solution to the blockchain order-fairness problem [11], [13], [75], [76]. In particular, Chen et al. [76] proposed execution-time optimizations for Themis’s ordering algorithm. In this section, we revisit Themis through the lens of the Fair Consensus Factory, illustrating how the framework enables systematic performance improvements while preserving the fairness guarantees of the original design.

### 5.1. Design of Themis

Themis implements batch order fairness using a FIFO-via-voting algorithm. In each round, given a collection of messages and metadata about their local arrival orders from  $t$  nodes, the algorithm constructs a directed graph that captures precedence constraints between messages. The algorithm then produces an ordered batch of messages consistent with this precedence structure. Importantly, unlike earlier work [19], Themis does not require observing a complete Condorcet cycle to output a valid ordering, thus improving responsiveness.

To embed this ordering process in a consensus protocol, Themis builds on HotStuff [70]. Each round, the leader collects local ordering metadata from all nodes, executes the ordering algorithm, and proposes the resulting message batch. To ensure correctness, the proposal includes all metadata so that followers can recompute and verify the leader’s output before voting.

Figure 7 illustrates this process. During the commit phase of the previous round, each node sends its local ordering state to the next-round leader. The leader then runs the FIFO-via-voting algorithm and proposes an ordered batch, along with the metadata used to compute it. All nodes independently re-execute the algorithm and validate the result before proceeding with the HotStuff voting phases.

### 5.2. An Alternative Themis Design with Latency Optimization

With Fair Consensus Factory, we propose an alternative design of Themis that still uses  $P_{\text{Hotstuff}}$  and  $r_{\text{Themis}}$ . We show that this alternative design achieves lower latency than the original Themis protocol without sacrificing throughput.

1. With zero-knowledge proofs, the node responsible for applying the ordering rule is also required to generate proof of the computation, which is computationally expensive and therefore results in high latency. To the best of our knowledge, no experimental evaluations have demonstrated otherwise.

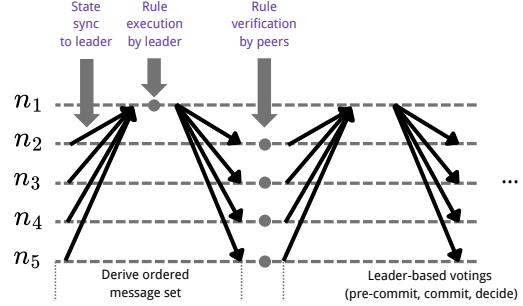


Figure 7. Themis round structure: the leader gathers local states in the commit phase of the previous round, runs the ordering algorithm, and proposes the ordered batch. The followers re-execute the algorithm to verify the proposal.

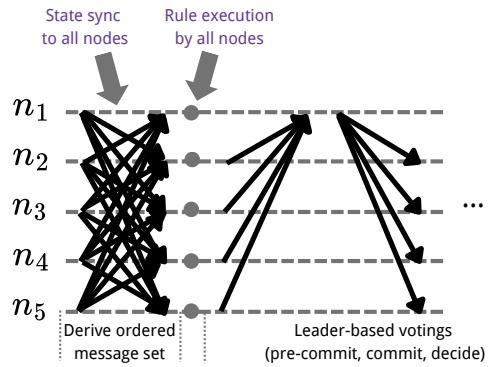


Figure 8. Optimized Themis structure: nodes exchange metadata in a single all-to-all stage and locally compute the ordering result.

Figure 8 visualizes the resulting protocol after applying the leader-based protocol transformation presented in Section 4.3. This alternative protocol has two key differences compared to the original Themis protocol. First, Themis uses a two-stage synchronization (all-to-leader, leader-to-all), while the alternative protocol uses a single-stage all-to-all synchronization. Second, in Themis, rule execution is separated from rule verification (as shown in Figure 7); in the alternative protocol, every node executes the ordering algorithm locally in the same round.

As discussed in Section 4.4, the optimized Themis protocol preserves the safety and liveness guarantee from Hotstuff.

### 5.3. Experimental Evaluation

We implemented both the original and optimized protocols using a Rust-based HotStuff implementation [77]. To isolate communication and synchronization effects, the running time of the ordering algorithm is abstracted as a fixed 100 ms delay. We measure throughput (transactions per second) and latency (from the time a transaction is sent to a network node to the time the transaction is committed) across 4, 7, 10, and 20 node configurations in a local

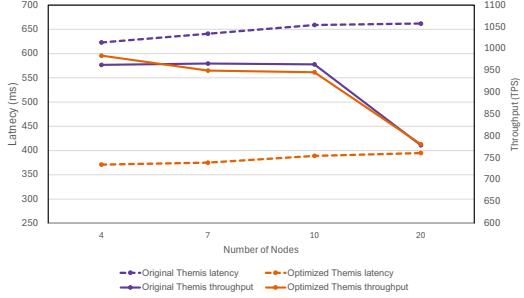


Figure 9. Throughput and latency comparison between original and optimized Themis protocols.

network, running each configuration for 10 minutes and repeating it three times to ensure stability.

As shown in Figure 9, throughput remains comparable across both versions. However, the optimized protocol achieves a significant reduction in latency, approximately 40% lower, demonstrating the effectiveness of early rule execution and symmetric communication. Importantly, the result shows that the all-to-all synchronization pattern has little negative impact on latency and throughput even as the number of nodes grows. This result supports the claim that the Fair Consensus Factory enables systematic construction and optimization of fair consensus protocols.

## 6. Fair Ordering Research Roadmap

This section highlights some open questions and potential research directions in the future development of fair ordering consensus.

### 6.1. How to Choose Fair Ordering Rules

With these categories of fair ordering rules, there are few discussions on which ordering rule is the “better” one, or which one is more suitable for distributed ledgers serving which applications. A possibility is that the ordering rules should be application-specific [78], e.g., first-come-first-served is more suitable for financial applications [4]. Random ordering approach could be the alternative that is a more specific solution to the problem of MEV with a lower ordering algorithm complexity. However, outside financial applications, it is argued that the current most popular fee-based prioritization approach [79] results in higher overall social welfare [80] compared to FIFO.

### 6.2. Adding Flexibility to Order-Insensitive Messages

In practical distributed ledgers, not all messages are order-sensitive. Currently, there are proposals to classify order-sensitive and order-insensitive messages and execute different protocols on these messages [24], [65], [72], [81].

However, this classification is not yet applied to fair ordering consensus to reduce latency by having to order fewer messages.

The Sui blockchain [72] pictured the scenario that in hybrid blockchains some messages do not need to go through consensus if the resulting state changes only affect the message senders themselves and designed a fast commit path for such messages. Another approach [81] is to introduce a consensus that includes a new type of token that can be paid to prioritize messages, allowing order-sensitive messages to be prioritized at a higher cost.

Amiri et al. [24] define order-sensitive messages as those that require accessing a shared resource, where the resulting state changes if the execution order changes. However, messages that also require accessing shared resources may also be order-insensitive.

Here we present a potential way to distinguish order-sensitive messages: instead of having the system decide which messages are order-sensitive, message senders should define whether their messages are order-sensitive or not. Given such input from message senders, the consensus protocol can apply fair ordering rules only to the ones that are flagged order-sensitive. This reduces the work of the system and results in a more precise separation of order-sensitive messages. However, this is only an initial idea we left as future work to further explore.

### 6.3. Message Ordering as a Service

When implementing message ordering, most existing proposals integrate them into total ordering consensus protocols to minimize the overhead of exchanging the metadata and verifying the ordering. In line with the rising awareness of fair ordering, there is a trend to isolate, and potentially outsource, the process of message sequencing from reaching consensus in distributed ledger systems [71], [82]–[85]. In such a design, specialized resources are used to receive messages from users and sequence the messages before they are passed to the validating nodes to commit to the network.

This can potentially lead to a new design paradigm for achieving order fairness in distributed ledgers. Distributed fair ordering can be taken as an isolated service that serves multiple distributed ledgers. A related proposal is found in an advancement idea for Ethereum [86], which proposed to enable faster execution of ordering in consensus by separating block proposal and transaction ordering [86]. This is also similar to the concept of modular blockchain [87], which separates a blockchain system into multiple sub-systems. From this perspective, a fair-ordering service can be viewed as a standalone system in a modular blockchain, which can also serve multiple consensus and application layer sub-systems. However, there are no other works, as far as we know, that explore the design of an outsourced fair ordering system.

## 7. Conclusion

This paper explores fair message ordering in distributed ledgers, addressing a critical need for fairness in distributed ledgers where manipulation of transaction order, such as Maximal Extractable Value, becomes a prominent problem. By examining FIFO, random, and blind ordering, we outline the challenges and trade-offs of implementing fair ordering in Byzantine fault-tolerant settings. Our proposed Fair Consensus Factory framework enables flexible integration of fairness into consensus protocols and was demonstrated through redesigning the FIFO ordering consensus protocol, Themis, to reduce latency. This approach shows the potential of modular fairness frameworks to improve both performance and equity in distributed systems. Having a better understanding of fair message ordering accelerates the adoption of blockchain technologies. Our finding provides good guidance for researchers in this area by providing a systematic understanding of existing works and identifying future research directions.

## Acknowledgments

This project is funded by a UKRI Future Leaders Fellowship (MR-W009560-1): ‘*Digital Assisted Collective Governance of Smart City Commons–ARTIO*’.

## References

- [1] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan, “Themis: Fast, strong order-fairness in byzantine consensus,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 475–489.
- [2] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Satoshi Nakamoto*, 2008.
- [3] G. Wood *et al.*, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.
- [4] Y. Sokolik and O. Rottenstreich, “Age-aware fairness in blockchain transaction ordering,” in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–9.
- [5] K. Qin, L. Zhou, and A. Gervais, “Quantifying blockchain extractable value: How dark is the forest?” in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 198–214.
- [6] M. Raynal, *Communication and agreement abstractions for fault-tolerant asynchronous distributed systems*. Springer Nature, 2022.
- [7] M. Correia, N. F. Neves, and P. Veríssimo, “How to tolerate half less one byzantine nodes in practical distributed systems,” in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004*. IEEE, 2004, pp. 174–183.
- [8] T. Distler, “Byzantine fault-tolerant state-machine replication from a systems perspective,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–38, 2021.
- [9] R. Li, X. Hu, Q. Wang, S. Duan, and Q. Wang, “Transaction fairness in blockchains, revisited,” *Cryptology ePrint Archive*, 2023.
- [10] T. Rocket, “Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies,” Available [online]. [Accessed: 4-12-2018], 2018.
- [11] L. Heimbach and R. Wattenhofer, “SoK: Preventing transaction reordering manipulations in decentralized finance,” in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 2022, pp. 47–60.
- [12] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, “High-frequency trading on decentralized on-chain exchanges,” in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 428–445.
- [13] K. Qin, S. Chaliasos, L. Zhou, B. Livshits, D. Song, and A. Gervais, “The blockchain imitation game,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3961–3978.
- [14] T. Chitra and K. Kulkarni, “Improving proof of stake economic security via mev redistribution,” in *Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*, 2022, pp. 1–7.
- [15] L. Zhou, K. Qin, and A. Gervais, “A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges,” *arXiv preprint arXiv:2106.07371*, 2021.
- [16] P. Züst, T. Nadahalli, and Y. W. R. Wattenhofer, “Analyzing and preventing sandwich attacks in ethereum,” *ETH Zürich*, pp. 1–29, 2021.
- [17] M. Kelkar, S. Deb, and S. Kannan, “Order-fair consensus in the permissionless setting,” in *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, 2022, pp. 3–14.
- [18] R. Pass and E. Shi, “Hybrid consensus: Efficient consensus in the permissionless model,” *Cryptology ePrint Archive*, 2016.
- [19] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, “Order-fairness for byzantine consensus,” in *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*. Springer, 2020, pp. 451–480.
- [20] S. Shyamsukha, P. Bhattacharya, F. Patel, S. Tanwar, R. Gupta, and E. Pricop, “Porf: Proof-of-reputation-based consensus scheme for fair transaction ordering,” in *2021 13th International conference on electronics, computers and artificial intelligence (ECAI)*. IEEE, 2021, pp. 1–6.
- [21] C. Cachin, J. Mićić, N. Steinhauer, and L. Zanolini, “Quick order fairness,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2022, pp. 316–333.
- [22] A. Kiayias, N. Leonidas, and Y. Shen, “Ordering transactions with bounded unfairness: definitions, complexity and constructions,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 34–63.
- [23] G. Wang, L. Cai, F. Gai, and J. Niu, “Phalanx: A practical byzantine ordered consensus protocol,” *arXiv preprint arXiv:2209.08512*, 2022.
- [24] M. J. Amiri, H. Nagda, S. P. Singhal, and B. T. Loo, “Rashnu: Data-dependent order-fairness,” *Cryptology ePrint Archive*, Tech. Rep., 2022.
- [25] M. A. Vafadar and M. Khabbazian, “Condorcet attack against fair transaction ordering,” *arXiv preprint arXiv:2306.15743*, 2023.
- [26] G. Ramseyer and A. Goel, “Brief announcement: Fair ordering via streaming social choice theory,” in *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, 2024, pp. 279–282.
- [27] Y. Zhang, S. Setty, Q. Chen, L. Zhou, and L. Alvisi, “Byzantine ordered consensus without byzantine oligarchy,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 633–649. [Online]. Available: <https://www.usenix.org/conference/osdi20/presentation/zhangyunhao>
- [28] K. Kursawe, “Wendy, the good little fairness widget: Achieving order fairness for blockchains,” in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 25–36.

- [29] ——, “Wendy grows up: More order fairness,” in *Financial Cryptography and Data Security: FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25*. Springer, 2021, pp. 191–196.
- [30] L. Baird, “The swirls hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance,” *Swirls Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, vol. 34, pp. 9–11, 2016.
- [31] B. Xue and S. Kannan, “Travelers: A scalable fair ordering bft system,” *arXiv preprint arXiv:2401.02030*, 2024.
- [32] V. Gramoli, Z. Lu, Q. Tang, and P. Zarbafian, “Optimal asynchronous拜占庭共识 with fair separability,” *Cryptology ePrint Archive*, 2024.
- [33] ——, “Aoab: Optimal and fair ordering of financial transactions.”
- [34] P. Zarbafian and V. Gramoli, “Aion: Secure transaction ordering using tees,” in *European Symposium on Research in Computer Security*. Springer, 2023, pp. 332–350.
- [35] ——, “Lyra: Fast and scalable resilience to reordering attacks in blockchains,” in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 929–939.
- [36] A. Constantinescu, D. Ghinea, L. Heimbach, Z. Wang, and R. Wattenhofer, “A fair and resilient decentralized clock network for transaction ordering,” *arXiv preprint arXiv:2305.05206*, 2023.
- [37] C. Cachin and J. Mićić, “Quick order fairness: Implementation and evaluation,” in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2024, pp. 230–234.
- [38] W. V. Gehrlein, “Condorcet’s paradox,” *Theory and decision*, vol. 15, no. 2, pp. 161–197, 1983.
- [39] T. N. Tideman, “Independence of clones as a criterion for voting rules,” *Social Choice and Welfare*, vol. 4, no. 3, pp. 185–206, 1987.
- [40] Z. Zhang, L. Zhang, Z. Wang, Y. Li, R. Lu, and Y. Yu, “Chronos: An efficient asynchronous拜占庭共识 ordered consensus,” *The Computer Journal*, vol. 67, no. 3, pp. 1153–1162, 2024.
- [41] J. Lundelius and N. Lynch, “A new fault-tolerant algorithm for clock synchronization,” in *Proceedings of the third annual ACM symposium on Principles of distributed computing*, 1984, pp. 75–88.
- [42] E. Pournaras, “Proof of witness presence: Blockchain consensus for augmented democracy in smart cities,” *Journal of Parallel and Distributed Computing*, vol. 145, pp. 160–175, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0743731520303282>
- [43] D. Stoltz and R. Wattenhofer, “Byzantine agreement with median validity,” in *19th International Conference on Principles of Distributed Systems (OPODIS 2015)*, vol. 46. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016, p. 22.
- [44] A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, R. Tamari, and D. Yakira, “A fair consensus protocol for transaction ordering,” in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 55–65.
- [45] D. Yakira, A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, and R. Tamari, “Helix: A fair blockchain consensus protocol resistant to ordering manipulation,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1584–1597, 2021.
- [46] A. Orda and O. Rottenstreich, “Enforcing fairness in blockchain transaction ordering,” *Peer-to-peer Networking and Applications*, vol. 14, no. 6, pp. 3660–3673, 2021.
- [47] M. Nassar, O. Rottenstreich, and A. Orda, “Cfto: Communication-aware fairness in blockchain transaction ordering,” *IEEE Transactions on Network and Service Management*, 2023.
- [48] O. Alpos, I. Amores-Sesar, C. Cachin, and M. Yeo, “Eating sandwiches: Modular and lightweight elimination of transaction reordering attacks,” *arXiv preprint arXiv:2307.02954*, 2023.
- [49] A. Kavousi, D. V. Le, P. Jovanovic, and G. Danezis, “Blindperm: Efficient mev mitigation with an encrypted mempool and permutation,” *Cryptology ePrint Archive*, 2023.
- [50] J. Piet, V. Nair, and S. Subramanian, “Mevade: An mev-resistant blockchain design,” in *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2023, pp. 1–9.
- [51] J. H.-y. Chiang, B. David, I. Eyal, and T. Gong, “Fairpos: input fairness in permissionless consensus,” *Cryptology ePrint Archive*, 2022.
- [52] D. Malkhi and P. Szalachowski, “Maximal extractable value (mev) protection on a dag,” *arXiv preprint arXiv:2208.00940*, 2022.
- [53] M. K. Reiter and K. P. Birman, “How to securely replicate services,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 986–1009, 1994.
- [54] C. Stathakopoulou, S. Rüsch, M. Brandenburger, and M. Vukolić, “Adding fairness to order: Preventing front-running attacks in bft protocols using tees,” in *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2021, pp. 34–45.
- [55] H. Zhang, L.-H. Merino, Z. Qu, M. Bastankhah, V. Estrada-Galinanes, and B. Ford, “F3b: A low-overhead blockchain architecture with per-transaction front-running protection,” *arXiv preprint arXiv:2205.08529*, 2022.
- [56] S. Duan, M. K. Reiter, and H. Zhang, “Secure causal atomic broadcast, revisited,” in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 61–72.
- [57] A. R. Choudhuri, S. Garg, J. Piet, and G.-V. Policharla, “Mempool privacy via batched threshold encryption: Attacks and defenses,” *Cryptology ePrint Archive*, 2024.
- [58] D. S. Antunes, A. N. Oliveira, A. Breda, M. G. Franco, H. Moniz, and R. Rodrigues, “Alea-BFT: Practical asynchronous拜占庭共识 fault tolerance,” in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 313–328.
- [59] M. Ciampi, A. Kiayias, and Y. Shen, “Universal composable transaction serialization with order fairness,” in *Annual International Cryptology Conference*. Springer, 2024, pp. 147–180.
- [60] H. Zhang, L.-H. Merino, V. Estrada-Galinanes, and B. Ford, “Flash freezing flash boys: Counteracting blockchain front-running,” in *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2022, pp. 90–95.
- [61] J. Bormet, S. Faust, H. Othman, and Z. Qu, “Beat-mev: Epochless approach to batched threshold encryption for mev prevention,” *Cryptology ePrint Archive*, 2024.
- [62] A. R. Choudhuri, S. Garg, G.-V. Policharla, and M. Wang, “Practical mempool privacy via one-time setup batched threshold encryption,” *Cryptology ePrint Archive*, 2024.
- [63] A. Agarwal, R. Fernando, and B. Pinkas, “Efficiently-thresholdizable selective batched identity based encryption, with applications,” *Cryptology ePrint Archive*, 2024.
- [64] A. Misra and A. D. Kshemkalyani, “Towards stronger blockchains: security against front-running attacks,” in *International Conference on Networked Systems*. Springer, 2024, pp. 171–187.
- [65] B. Riva, A. Sonnino, and L. Kokoris-Kogias, “Seahorse: Efficiently mixing encrypted and normal transactions.”
- [66] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
- [67] Y. Desmedt, “Threshold cryptosystems,” in *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1992, pp. 1–14.
- [68] A. Khajehpour, H. Akbarinodehi, M. Jahanara, and C. Feng, “Mitigating mev via multiparty delay encryption,” *Cryptology ePrint Archive*, 2023.

- [69] Z. Li, S. Majumdar, and E. Pournaras, “Blockchain-based decentralized time lock machines: Automated reveal of time-sensitive information,” *arXiv preprint arXiv:2401.05947*, 2024.
- [70] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, “Hotstuff: Bft consensus in the lens of blockchain,” *arXiv preprint arXiv:1803.05069*, 2018.
- [71] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, “Narwhal and tusk: a dag-based mempool and efficient bft consensus,” ser. EuroSys ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 34–50. [Online]. Available: <https://doi.org/10.1145/3492321.3519594>
- [72] K. Babel, A. Chursin, G. Danezis, L. Kokoris-Kogias, and A. Sonnino, “Mysticeti: Low-latency dag consensus with fast commit path,” *arXiv preprint arXiv:2310.14821*, 2023.
- [73] “The Future of MEV is SUAVE | Flashbots Writings,” Nov. 2022. [Online]. Available: <https://writings.flashbots.net/the-future-of-mev-is-suave>
- [74] IC3, “PROF: Fair Transaction-Ordering in a Profit-Seeking World,” Jul. 2024. [Online]. Available: <https://initc3org.medium.com/prof-fair-transaction-ordering-in-a-profit-seeking-world-b6dadd71f086>
- [75] S. Yang, F. Zhang, K. Huang, X. Chen, Y. Yang, and F. Zhu, “SoK: Mev countermeasures: Theory and practice,” *arXiv preprint arXiv:2212.05111*, 2022.
- [76] W. Chen, Y. Feng, J. Zhang, Z. Cai, H.-N. Dai, and Z. Zheng, “Auncel: Fair byzantine consensus protocol with high performance,” in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 1251–1260.
- [77] “asonnino/hotstuff: Implementation of the HotStuff consensus protocol.” <https://github.com/asonnino/hotstuff>.
- [78] T. Chitra, “Towards a theory of maximal extractable value ii: uncertainty,” *arXiv preprint arXiv:2309.14201*, 2023.
- [79] J. Messias, M. Alzayat, B. Chandrasekaran, K. P. Gummadi, P. Loiseau, and A. Mislove, “Selfish & opaque transaction ordering in the bitcoin blockchain: the case for chain neutrality,” in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021, pp. 320–335.
- [80] T. Diamandis and G. Angeris, “A note on the welfare gap in fair ordering,” *arXiv preprint arXiv:2303.15239*, 2023.
- [81] A. Vedula, S. B. Venkatakrishnan, and A. Gupta, “Masquerade: Simple and lightweight transaction reordering mitigation in blockchains,” *arXiv preprint arXiv:2308.15347*, 2023.
- [82] L. Heimbach, L. Kiffer, C. Ferreira Torres, and R. Wattenhofer, “Ethereum’s proposer-builder separation: Promises and realities,” in *Proceedings of the 2023 ACM on Internet Measurement Conference*, 2023, pp. 406–420.
- [83] S. Motepalli, L. Freitas, and B. Livshits, “SoK: Decentralized sequencers for rollups,” *arXiv preprint arXiv:2310.03616*, 2023.
- [84] “Welcome to Flashbots | Flashbots Docs,” Jun. 2024. [Online]. Available: <https://docs.flashbots.net/>
- [85] J. Bearer, B. Bünz, P. Camacho, B. Chen, E. Davidson, B. Fisch, B. Fish, G. Gutoski, F. Krell, C. Lin *et al.*, “The espresso sequencing network: Hotshot consensus, tiramisu data-availability, and builder-exchange,” *Cryptology ePrint Archive*, 2024.
- [86] K. Mu, B. Yin, A. Asheralieva, and X. Wei, “Separation is good: A faster order-fairness byzantine consensus,” in *Network and Distributed System Security Symposium*, 2024.
- [87] S. Cohen, G. Goren, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, “Proof of availability and retrieval in a modular blockchain architecture,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2023, pp. 36–53.