Order-Fair Consensus in the Permissionless Setting

Mahimna Kelkar
Cornell Tech

Soubhik Deb
University of Washington, Seattle

Sreeram Kannan
University of Washington, Seattle

ABSTRACT

Transaction-order-manipulation attacks have become commonplace in public blockchains such as Ethereum, costing hundreds of millions of dollars. In these blockchains, a miner can unilaterally determine the order of transactions inside a block, and this ordering is not checked by other users, leaving room for the miner to manipulate the order for its own benefit. This gap is also evident from existing security results for permissionless blockchains. As prime examples, the breakthrough work of Garay et al. (Eurocrypt 2015) and Pass et al. (Eurocrypt 2017) showed the security properties of *consistency* and *liveness* for Nakamoto's seminal proof-of-work protocol. However, consistency and liveness do not provide any guarantees on the relationship between the order in which transactions arrive into the network and the finalized order in the ledger.

As a solution, a recent paper by Kelkar et al. (Crypto 2020) introduced a third useful property for consensus protocols: *(transaction)-order-fairness*, which proposes a strong relationship between the transaction arrival order and their order in the ledger. Their model was limited to the classical (permissioned) setting however, where the set of protocol nodes is fixed a priori, and does not fit well for permissionless environments where order-manipulation attacks have been most prominent. In this work, we initiate the investigation of order-fairness in the permissionless setting and design two protocols that realize this new property addition to standard requirements of consistency and liveness. The key insight behind our protocols in providing order-fairness is that a miner can no longer unilaterally determine ordering and proposals from many miners are combined in a fair way to construct the finalized ordering.

CCS CONCEPTS

• **Security and privacy** → *Distributed systems security*.

KEYWORDS

Order-Fair Consensus; Blockchain; Decentralized Finance

ACM Reference Format:
Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. 2022. Order-Fair Consensus in the Permissionless Setting. In *Proceedings of the 9th ACM ASIA Public-Key Cryptography Workshop (APKC '22), May 30, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3494105.3526239

1 INTRODUCTION

Transaction-order-manipulation attacks. Decentralized finance (DeFi) systems have become massively popular on permissionless blockchains in the last few years. As of Jan 2022, the Ethereum blockchain houses more than 101 billion USD [5] of locked capital in DeFi smart contracts. For DeFi contracts, transaction ordering is crucially important; execution in an unfavorable order can result in unnecessary losses for the user. Unfortunately, a long line of work [19, 42, 48] has shown the rise of *adversarial* manipulation of transaction ordering on public blockchains such as Ethereum. This manipulation is extraordinarily lucrative for attackers and has resulted in the loss of hundreds of millions of dollars for users [36, 41, 45] as well as the now infamous *Black Thursday* event [13]. In fact, attackers often compete with each other in an attempt to finalize the ordering most profitable to them [19].

Order-manipulation attacks have been shown to result in adversaries extracting profit at the expense of ordinary users [19, 20], systemic bribery [34], and even fundamental protocol instability [19, 30]. The recently incorporated Ethereum improvement proposal (EIP-1559) [1], which cuts transaction fees sent to the miner, has exacerbated the problem of order-manipulation by incentivizing miners to add order-manipulation software [18] or auction of the right to order transactions in their block [21, 23].

Fair Ordering. Order-manipulation attacks are enabled because a miner can unilaterally determine the order of transactions inside a block, and this ordering is not checked by other users, leaving room for the miner to manipulate the order for its own benefit. Such unilateral manipulation points to a core fundamental issue in blockchains: there is no relationship between the order in which transactions arrive into the network and the final transaction ordering in the log. Neither consistency nor liveness—the two pillars of distributed-system security—enforces any guarantees on the relationship between the order in which transactions arrive into the network and the final transaction ordering in the log.

As a solution, recent work by Kelkar, Zhang, Goldfeder, and Juels [29] (henceforth KZGJ) initiated the study of *fair ordering* protocols by adding a new consensus property, called *order-fairness*. Informally, if a transaction tx_1 was received before another one tx_2 by a large fraction of the nodes in the network, order-fairness dictates that tx_1 be sequenced in the output log first. A consensus protocol with such a property cuts the influence miners have on transaction ordering, and is able to handle any kind of adversarial censorship, reordering, and transaction insertions, even those based on user collusion. However, KZGJ only considered the permissioned setting, and consequently, their protocols do not directly apply to previously mentioned attacks which were largely on permissionless blockchains. Our work constructs the first permissionless fair ordering protocol by extending the work of KZGJ.

1.1 Our Contributions

Permissionless Order-Fair Consensus. We introduce the first formalization of a fair transaction ordering property in the permissionless setting by generalizing permissioned *order-fairness* from KZGJ (Section 2). We find that permissionless order-fairness is impossible to achieve and not particularly interesting in the presence of completely dynamic adversaries which can corrupt and kill nodes before blocks are mined (see Section 2.1). Therefore, we will only consider adversaries that do not churn through nodes too quickly.

We provide two permissionless fair ordering protocols that can be generically instantiated from any longest-chain protocol (e.g., proof-of-work, proof-of-stake etc.) although we will use PoW for concreteness. The key insight we use to get around the unilateral proposition of transaction ordering by a miner is the following: Ordering for each transaction (w.r.t. other transactions) is now proposed across multiple, say d, blocks. The proposed orderings within these blocks are then aggregated in a *fair* way to obtain the final transaction ordering. Each miner reports their own preferences on how to order a set of transactions; honest nodes will report their true received order of transactions from the peer-to-peer network, while an adversarial node can arrange transactions in arbitrary order in the block it mines so as to influence the final ordering obtained from consensus. We call our design structure Hammurabi.

In essence, the system as a whole will mine d independent transaction lists, which will be appended to as new transactions arrive. We call these "semantic chains" since transactions in the same list are logically or semantically connected even though they may directly follow a different mined block. All transactions will be mined d times, once in each semantic chain. The final fair ordering will be extracted from the d semantic chains. Notably, d is a function only of the security parameter κ and not of the number of nodes. The upshot of such an approach is that in our protocols, the adversary will not be able to manipulate the ordering within a majority of semantic chains, resulting in a fair overall ordering.

Our first protocol, Π_{mod} (Section 3.1), uses a single PoW Nakamoto chain. Here, the d semantic chains will arise from the chain *modulo* d. Specifically, blocks at indices that are congruent modulo d will become part of the same semantic or chain. Π_{mod} satisfies order-fairness when the corruption ratio is less than $1/3$. Our second construction, $\Pi_{\text{fairfruit}}$ (Section 3.2) modifies the Fruitchains protocol [38] in order to achieve order-fairness for any corruption up to $1/2$. Consistency is satisfied by both protocols up to $1/2$ corruption.

Fair extraction (Section 4). Finally, we detail how to extract an ordering from the d semantic chains in a way that satisfies order-fairness. Specifically, we introduce a novel streaming finalization problem that formulates the extraction of a fair ordering from d transaction ordering lists $\mathcal{L} = [\mathsf{List}_1, \dots, \mathsf{List}_d]$ that are provided as input in a streaming fashion.

To solve the streaming finalization problem, we start by distilling the key techniques from the permissioned Aequitas protocol from KZGJ into an algorithm Aequitas$(\cdot)$. We then prove a player replaceability lemma which allows us to represent the semantic chains as transaction orderings of d "virtual" nodes in a permissioned setting, as long as a natural completeness property is satisfied by the way the semantic chains are built. This effectively reduces the permissionless order-fairness problem to its permissioned analogue by constructing ledgers held by virtual nodes. Along the way, we highlight interesting connections to voting theory that may be of independent interest.

Applications. It is clear how fair ordering protocols are beneficial to DeFi applications where order-manipulation is most prominent. Apart from DeFi, we provide two interesting applications that can take advantage of our protocols. We briefly discuss them here and provide more details in Section 6.

Reorgs. Informally, for a longest-chain protocol, a reorg is an event where a block that was originally part of the canonical (longest) chain is displaced by a competing block; a reorg attack is forcefully causing such a displacement. This is most common over a short to medium timeframe since long reorgs are usually prevented unless the chain is under e.g., a 51% attack.

While some types of reorg attempts (e.g., double spends) have been seen since the early days of Bitcoin, recent events [2] have uncovered the gory relation between order-manipulation and reorgs—how potentially stability-threatening reorgs arise from the power to manipulate transaction ordering. In particular, if choosing the transaction ordering for a specific block can give significant profit, an adversarial miner who did not initially mine that block may seek to reorg the chain and mine the block and claim the profit itself. While such reorgs have not been seen to a large extent yet in practice, the threat is very real and has led to substantial renewed interest [6] around reorgs.

Unfortunately, countermeasures proposed today rely on moving to protocols with stricter finality guarantees (which usually reduces decentralization and/or may not actually reduce reorg potential for wealthy adversaries), or on exerting social pressure on miners. Regardless, there is no proposed solution that fits with standard PoW chains. We make a useful observation here: Order-fairness is surprisingly useful at preventing reorgs since it is now provably impossible to have a transaction received later displace a transaction mined in an earlier block in the final ordering.

Zero-block confirmation. Another surprising related result is that in any fair ordering protocol, honest transactions can be locally finalized very fast, *within two-times the network delay.* We note that while there is a large body of work trying to improve the confirmation latency of permissionless systems [10, 22, 32, 46], none of them are able to achieve fast confirmation at this timescale, since all of them require mining at least a certain number of blocks at a much slower rate than network delay. We call this *zero-block confirmation.* Suppose that an honest node sees tx in round r and does not see a conflicting transaction within 2Δ. Then the honest node can immediately infer that tx will be ordered ahead of any conflicting transaction in the final ledger and can finalize tx. We note that the node may still need to wait much longer (for blocks to be mined) to know the precise location in the final ledger.

Practitioners have previously used similar techniques [28] for quickly confirming transactions, especially as it makes everyday real-world purchases with e.g., Bitcoin, more practical. However, there is no guarantee that a double-spend transaction appearing much later than the original transaction doesn't get ordered first in the finalized ledger (e.g., as a result of an adversarial miner). In contrast, in a fair-ordered protocol, this is not possible, thus making zero-block confirmation after waiting for 2Δ time secure.

We emphasize that although zero-block-confirmation follows easily from order-fairness, current permissionless protocols, even ones in the synchronous setting, do not provably achieve such a property, and that it is not a direct artifact of network synchrony.

2 MODEL AND PRELIMINARIES

We describe the general execution model here. Much of our formalism is adapted from an extensive line of research on permissionless consensus [26, 37–40]. We also use some of the formalism related to order-fairness from KZGJ [29].

General execution model. As standard practice [26, 37], we model protocol nodes as ITMs [16], and use a special environment machine $\mathcal{Z}$ to direct execution. $\mathcal{A}$ denotes the adversary. We assume a synchronous model where execution proceeds in *rounds*. At the start of a round, nodes receive transactions as input from $\mathcal{Z}$; at the end of a round, nodes may deliver outputs to $\mathcal{Z}$. For ease of modeling (similar to e.g., [40]), for honest nodes, the final ordering will be taken as the result of a transformation function $\mathsf{linearize}(\cdot)$ (part of the protocol description) on the outputs to $\mathcal{Z}$ rather than the outputs themselves. This will extract the relevant information from the outputs into a total ordering. New transactions are taken as inputs from $\mathcal{Z}$ to avoid explicitly modeling clients.

κ denotes the security parameter. For a protocol Π, $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ represents the random variable for all possible execution traces of Π w.r.t. $(\mathcal{A}, \mathcal{Z})$. Any view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ is a fully specified instance (including inputs, outputs, random coins etc.) of an execution trace.

Corruptions. $\mathcal{Z}$ can spawn new nodes either as *honest* or *corrupt* (i.e., controlled by $\mathcal{A}$). $\mathcal{A}$ can also corrupt existing nodes, as well as kill corrupted nodes (which removes them from execution).

First, observe that if $\mathcal{A}$ can corrupt and kill nodes at will, it can cause the network to "forget" the transaction input orderings received by earlier honest nodes. While this was not important for prior work, this is critically detrimental for any (permissionless) fair ordering protocol. Therefore, we will assume a *(mildly)-respawning-adversary* that limits the node churn. Informally, a (τ, R)-respawning adversary can kill at most τ fraction nodes within any period of R rounds. This corresponds to the notion that $\mathcal{A}$ cannot quickly corrupt and kill many nodes. We will distinguish between the respawning parameter and the adversarial corruption parameter (see Definition 2.2).

Network formalism. We assume a synchronous communication model, i.e., messages sent are delivered within a (known) bounded number of rounds Δ (a function of κ). $\mathcal{A}$ can delay and reorder messages (subject to the bound Δ) but cannot drop messages. We define the synchrony assumption below.

Definition 2.1 (Δ-synchrony). $(\mathcal{A}, \mathcal{Z})$ respects Δ-synchrony if for all view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following hold:

- (Input Synchrony) If an honest node receives tx as input from $\mathcal{Z}$ in round r, then in any round $r' \geq r + \Delta$, any honest node in the system at round r' will have already received tx as input from $\mathcal{Z}$. This includes any node that may have spawned in round r'.
- (Ledger Synchrony) If an honest node outputs tx to $\mathcal{Z}$ in round r (recall that this is not the same as being ordered in the final ledger), then in any round $r' \geq r + \Delta$, any honest node in the system at r' (including nodes spawned at r') will have already received tx as input from $\mathcal{Z}$.
- (Broadcast Synchrony) If an honest node sends a message in round r, then in any round $r' \geq r + \Delta$, any honest recipient node will have received the message by round r' (including nodes spawned at r').
- $\mathcal{Z}$ provides Δ to all nodes when they are spawned.

As standard, the synchrony bounds need to hold only for valid (e.g., non double-spend) transactions which prevents any denial-of-service attack through double-spends in practice. Alternatively, a fee to submit transactions can also be used. We now recall the definition of the permissionless setting next, taken from [40], and add to it our respawning parameters.

Definition 2.2 ($(n, \beta, \Delta, \tau, R)$-permissionless environments). We say that $(\mathcal{A}, \mathcal{Z})$ respects $(n, \beta, \Delta, \tau, R)$-permissionless execution w.r.t. a protocol Π if for every $\kappa \in \mathbb{N}$, $(\mathcal{A}, \mathcal{Z})$ respects Δ-synchrony, and for all view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds: (1) In each round, there are exactly[1] n nodes online in the system, of which at most βn are corrupt; (2) Within any period of R rounds, at most τn notes are killed. We call such an adversary (τ, R)-respawning; (3) $\mathcal{Z}$ provides to all nodes the parameters $n, \beta, \Delta, \tau, R$ upon spawning. Note that n, Δ, R are functions of κ.

Blockchain formalism. An abstract blockchain [37] chain is an ordered sequence of blocks output by an honest node. We assume familiarity with the (by now) standard requirements for blockchain protocols, and refer the reader to [26, 37, 40] for more details. Informally, for honest chains, we require (1) T-common-prefix which allows confirmation of all except the last T blocks; (2) (T, g_0, g_1)-chain-growth which signals a growth of at least T in T/g_0 rounds and at most T blocks in T/g_1 rounds; (3) (T, μ)-chain-quality, which means that within any T consecutive blocks, at least μT are mined by honest nodes. Our proofs will require these properties from previous well-studied constructions only in a black-box way.

Abstract blockchain protocols directly satisfy the state machine replication (SMR) properties of consistency and liveness (see [37, 40]). For our purpose, we define $(T_{\text{warmup}}, T_{\text{confirm}})$-liveness as follows: If an honest node is input m by $\mathcal{Z}$ in round $r \geq T_{\text{warmup}}$, then for any $r' \geq r + T_{\text{confirm}}$ and any honest node in round r' that outputs $\mathcal{X}$ to $\mathcal{Z}$, m is in $\mathsf{linearize}(\mathcal{X})$. When there is no warmup time, we simply write T_{confirm}-liveness.

As standard, we also assume that nodes have access to a random function $H : \{0,1\}^* \rightarrow \{0,1\}^{\kappa}$. This is provided by two oracles: $\mathsf{H}(x)$ which outputs $H(x)$ and $\mathsf{H.ver}(x, y)$ which outputs 1 if $H(x) = y$ and 0 otherwise. In any round, all parties can make q queries to H and any number of queries to $\mathsf{H.ver}$. An adversary $\mathcal{A}$ controlling p parties is allowed to make pq sequential oracle queries.

Blocks. A block B is a tuple $(h_{-1}, h', \mathtt{m}, \eta, h)$, where h_{-1} denotes a pointer to a parent block, h' denotes a pointer to a reference block (an arbitrary previous block), $\mathtt{m}$ denotes the block record (e.g., a vector of transactions), η denotes the PoW nonce, and h denotes the hash of the current block. The reference block is an important modeling choice for us since it allows B to *logically* follow a different

[1]While we model a fixed n in each round similar to most blockchain papers, we note that our model can easily be extended to handle a varying n, using complementary techniques [17, 25] that are well understood in literature.

block than its preceding block, enabling the construction of different *semantic chains*. Notably, this allows the validity condition for a block to also be defined in terms of the reference block, and possibly separately from the parent block. We note that $h_{-1} = h'$ in many cases (eg., the Nakamoto protocol).

Protocols are parameterized by a hardness function $p(\cdot)$ that defines $D_p = p(\kappa) \cdot 2^{\kappa}$ such that $\Pr_{\eta}[H(h_{-1}, h', \mathtt{m}, \eta) < D_p] = p(\kappa)$ for all $(h_{-1}, h', \mathtt{m})$. Now, we define a valid block B as a tuple $(h_{-1}, h', \mathtt{m}, \eta, h)$ where $H(h_{-1}, h', \mathtt{m}, \eta) = h$ and $h < D_p$. We use $\Pi_{\text{nak}}(p)$ to denote Nakamoto's protocol and $\Gamma^{p}_{\text{nak}}(n, \beta, \Delta) = 1$ to denote a Nakamoto-compliant [37] execution.

2.1 Order-Fairness

Background. KZGJ initiated the first formal study for fairness of transaction ordering in the permissioned setting. Informally, KZGJ defines γ-receive-order-fairness (resp. γ-batch-order-fairness) as the following property: If γ fraction of nodes receive as input a transaction tx before another one tx′, then tx should be included before (resp. no later than) tx′ in the final ordering.

We emphasize (as shown in KZGJ) that the properties considered by KZGJ are strictly stronger than previously considered notions like censorship resistance, random leader election, threshold encryption that also attempt to restrict adversarial influence on transaction ordering etc. We provide a brief comparison in Appendix B. We will therefore extend the definition of KZGJ to the permissionless setting. Note that since all of the previously mentioned real-world order-manipulation attacks were on permissionless systems, the protocols from KZGJ do not directly apply, our work constructs the first fair ordering protocols in the permissionless setting.

Choosing an order-fairness definition in the permissionless setting. Consider two transactions tx and tx′. First, we note that in the permissionless setting, nodes that are spawned far later (eg., after tx and tx′ are output), will receive tx and tx′ in the same round (see the input synchrony definition). Consequently, we cannot naïvely use the same order-fairness definition from the permissioned setting, since the antecedent will never be satisfied making the definition vacuous. Instead, we will only consider the first time that the n nodes in the system receive either transaction, corresponding to the initial propagation of the transactions. This is formalized in Definition 2.3.

Definition 2.3 (Order-Fairness). For a given view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, we define γ-receive-order-fairness (resp. γ-batch-order-fairness) as follows: For any tx and tx′, let r be the first round such that at least n nodes alive in round r have received either tx or tx′. Now, if γ fraction of those nodes have received tx before tx′ as input, then for any round $t \geq r$, and any node i that is honest in round t and outputs $\mathcal{X}$, the final linearly ordered log, $\mathsf{linearize}(\mathcal{X})$, either (1) contains neither transaction; or (2) contains tx before (resp. no later than) tx′.

Note that tx before tx′ includes the case where only tx is present. We say that a protocol satisfies order-fairness if it satisfies the above, except with negligible probability over a random view.

Similar to KZGJ, the batch relaxation is used since receive-order-fairness is impossible to achieve except in very specific settings. Note that for batch-order-fairness, transactions within a batch can still be totally ordered for execution; the only difference is that ordering within a batch is no longer considered unfair. The relaxation also is somewhat minimal in the sense that it is used only when the stronger notion is impossible. See Section 4 for more details.

3 FAIR ORDERING PROTOCOLS

We describe two fair ordering protocols, Π_{mod} and $\Pi_{\text{fairfruit}}$, that each satisfy consistency, liveness, and batch-order-fairness in the permissionless setting.

Hammurabi design. To better motivate our design, we first take a step back to look at transaction ordering in standard Nakamoto consensus. Here, when the adversary mines a block, it has full control over the inclusion, exclusion, and ordering of transactions in the mempool. This essentially results in an *ephemeral* ordering centralization. To achieve fairness, we need to somehow decentralize the ordering process; this forms the basis for our design.

Our constructions follow the same outline. Through PoW mining, nodes mine d *semantic* chains, each of which represents a separate transaction ordering. We use the term *semantic* since the transaction ordering within different blocks can be logically linked even if the blocks themselves are not in the PoW sense. All transactions will be mined once into each of the semantic chains, with the overall fair ordering being extracted from the different orderings. The intuition is that an adversary will not be able to manipulate the ordering within a majority of chains. For our protocols, d is a function only of the security parameter κ and not of the number of nodes.

In the subsections that follow, we describe how our two protocols form these semantic chains. For now, we assume access to an algorithm $\mathsf{Aequitas}(\cdot)$ that extracts a fair final ordering from the semantic chains. Later, in Section 4, we distill the key structure of the protocol from KZGJ [29], and prove a novel player replaceability lemma which allows us to implement $\mathsf{Aequitas}(\cdot)$ in the permissionless setting.

Scalability Tradeoffs. As a brief comment on scalability, we note that the latency of our protocols is only $5/2$ times that of the underlying e.g., Nakamoto chain. Further, note that even though transactions are mined d times, only one semantic chain needs to contain the full transaction; others can simply contain a reference-pointer. Consequently, the storage/bandwidth costs are only a constant overhead more. In practice, the number of semantic chains d can be set to the confirmation window of the underlying chain, e.g., 6 for the PoW Bitcoin chain.

The transaction throughput however is reduced by a factor of d. We conjecture that a more parallel design may help increase the throughput, as well as decrease the latency further (i.e., even smaller than a standard PoW chain) by leveraging techniques from e.g., [10, 46]. We leave this exploration open for future work.

3.1 Modulo-T Longest Chain Protocol

Our first order-fair protocol $\Pi^{\kappa,T}_{\text{mod}}$ uses a single PoW-style longest chain, where κ is the security parameter, and T is the modular parameter. $d = T$ semantic chains are present but we use T here since it also corresponds to the consistency parameter of the underlying chain. Π_{mod} is secure up to $1/3$ corruption.

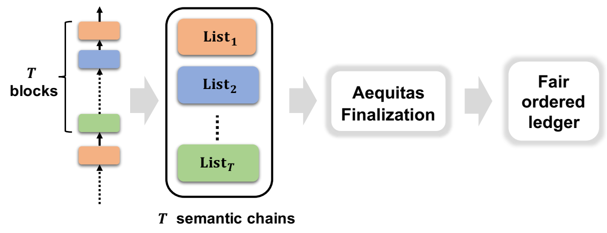

Figure 1: Pictorial representation of $\Pi^{\kappa,T}_{\text{mod}}$

```
$\Pi^{\kappa,T}_{\text{mod}}(p)$
Initialize: chain = genesis; TxPool = $\emptyset$

Upon receiving a valid chain$'$ such that $|\text{chain}'| > |\text{chain}|$:
  • Set chain := chain$'$ and output any newly seen transactions to $\mathcal{Z}$.
In round $r$, upon receiving txs from $\mathcal{Z}$:
  • For each tx in txs, add $(\text{tx}, r)$ to TxPool.
  • Set $h_{-1}$ to be the reference of chain$[-1]$ and $h'$ to be the reference
    of chain$[-T]$ if $|\text{chain}| > T$ otherwise set $h'$ to be the reference of
    chain$[0]$. Suppose that chain$[-1]$ belongs to modular chain $j$.
  • Let txincludes be the transactions in TxPool that are not in any block
    in modular chain $(j+1 \bmod T)$, ordered by increasing round number.
  • For $i = 1$ to $q$:
      • Pick a random nonce $\eta$ and let $h = \mathsf{H}(h_{-1}, h', \text{txincludes}, \eta)$.
      • If $h < D_p$, then set chain := chain $\|$ $(h_{-1}, h', \text{txincludes}, \eta, h)$,
        broadcast chain and break.
  • Let $\mathcal{L} = [\text{List}_1, \ldots, \text{List}_T] = \text{modularize}(\text{chain}[: -T], T)$
  • Output $\mathcal{L}$ to $\mathcal{Z}$. The final linearly ordered log output is taken as
    linearize$(\mathcal{L})$.
linearize$(\mathcal{L})$
  • Output Aequitas$(\mathcal{L})$.
```

Figure 2: Protocol $\Pi^{\kappa,T}_{\text{mod}}$

Notation. We use chain$[i]$ to denote the i^{th} (indexed from 0) block, chain$[: i]$ to denote the first i blocks, and chain$[i : j]$ to denote blocks from chain$[i]$ to chain$[j]$ (both inclusive). Further, chain$[-i]$ denotes the i^{th} block from the rear and chain$[: -i]$ denotes chain excluding the last i blocks.

Protocol description. In $\Pi^{\kappa,T}_{\text{mod}}$, nodes mine a single PoW chain. For a given chain, chain$[b]$ is considered to logically follow chain$[b-T]$, despite its PoW hash following chain$[b-1]$. In other words, the PoW chain will be split into T same semantic or "modular" chains, with blocks in the same index modulo T (e.g., indices $1, T+1, 2T+1$ and so on) describing a single transaction ordering. To obtain the fair ordering, Aequitas$(\cdot)$ will be run on the semantic chains in the confirmed part of the PoW chain.

For a given chain, let $\mathcal{T}_b$ be the set of transactions that were first seen in block chain$[b]$. Now, if chain$[b]$ was honestly mined, then the first honestly mined block after chain$[b]$ in each modular chain should contain these transactions. In other words, all honest nodes can reject blocks in the range chain$[b+1 : b+T-1]$ that do not contain all transactions in $\mathcal{T}_b$. Consequently, in a valid chain, the record $\mathtt{m}$ in any block chain$[b]$ should be such that $\bigcup_{i=b-T+1}^{b-1} \mathcal{T}_i \subseteq \mathtt{m}$. Figure 1 contains a simplified pictorial representation and Figure 2 contains the detailed pseudocode.

Valid chain. A chain is valid iff (1) chain$[0] = (0, 0, \perp, 0, H(0, 0, \perp, 0))$ is the *genesis* block; (2) chain$[b].h_{-1}$ = chain$[b-1].h$ for all $b \in \{1, \cdots, |\text{chain}|\}$; (3) chain$[b].h'$ equals *genesis.h* for $b \in \{1, \ldots, T\}$ and chain$[b-T].h$ for all $b \in \{T+1, \cdots, |\text{chain}|\}$; and (4) For $b \in \{1, \cdots, |\text{chain}|\}$, $\bigcup_{i=b-T+1}^{b-1} \mathcal{T}_i \subseteq$ chain$[b].\mathtt{m}$ holds.

Modular chains. For a given valid chain, we say that two blocks chain$[b]$ and chain$[b']$ where b, b' are positive integers are in the same modulo-T chain if $b \equiv b' \mod T$. The genesis block, chain$[0]$ is assumed to be included in all modular chains. We use modularize(chain, T) to denote the operation of splitting chain into modulo-T chains. Specifically, the output of modularize(chain, T) is $\mathcal{L} = [\text{List}_1, \ldots, \text{List}_T]$ such that List$_j$ concatenates the transaction orderings in all chain$[b]$ where $b \equiv j \mod T$. We will use $\mathtt{ref} = h'$ to point to the previous block in the same modular chain.

Dynamic corruption. Recall that we restrict $\mathcal{A}$ to be (τ, R)-respawning, i.e., at most τn nodes can be killed within any R round period. Now, for $\Pi^{\kappa,T}_{\text{mod}}$, we will choose R such that, except with negligible probability, R is larger than the time it takes for a transaction tx to be confirmed in all modular chains. It turns out that choosing R to be larger than the time to mine $5T/2$ blocks suffices.

Compliant parameters. We define a compliance predicate for $\Pi_{\text{mod}}(p)$ as $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta, \gamma, T, \tau, R) = 1$ if n, Δ, T, R are polynomials in κ; the parameters β, γ, τ are constants such that $\gamma - (\beta + \tau) > \frac{2}{3}$; and for all κ, $\Gamma^{p}_{\text{nak}}(n, \beta, \Delta) = 1$, $\Pi_{\text{nak}}(p)$ satisfies T-common-prefix and $(T, \mu > \frac{1}{2})$-chain-quality, $R \geq \frac{5T+2}{2g_0}$ where g_0 is such that $\Pi_{\text{nak}}(p)$ also satisfies $(T, g_0, \cdot)$-chain-growth.

We now state our main result below. The full proof is given in Section 5.

Theorem 3.1. *Consider $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta = 1, \gamma, T, \tau, R) = 1$. Then, $\Pi^{\kappa,T}_{\text{mod}}(p)$ satisfies consistency, R-liveness, and γ-batch-order-fairness for $(n, \beta, \Delta, \tau, R)$ environments.*

3.2 Fruitchains-based Protocol $\Pi_{\text{fairfruit}}$

Our second protocol, $\Pi_{\text{fairfruit}}$, is similar to Π_{mod} but uses Fruitchains protocol from [38] as the underlying protocol instead of Π_{nak}. This allows it to achieve order-fairness for $\gamma - (\beta + \tau) > 1/2$ (i.e., any minority corruption $(\beta + \tau < 1/2)$ for $\gamma = 1$).

Background on Fruitchains. In the Fruitchains protocol, nodes create two types of objects: fruits and blocks. Fruits are the ones that actually contain transactions while blocks are used to order fruits (and thereby order transactions). Fruits are required to "hang" from recent blocks, where recency is defined using a recency parameter Y. Nodes use the same PoW mining (using the 2-for-1 mining technique [26]) to mine both fruits and blocks. The random oracle output is now of size 2κ. The hash computation will also now include a digest digest(F) of the current list of fruits. A block is said to be mined if the first κ bits h_f of the hash are less than the mining hardness threshold D_p, while a fruit is said to be mined if the last κ bits h_s of the hash are less than a threshold D_{p_f}.

A block will be propagated to the rest of the network as part of the chain it gets mined in, whereas a fruit can be simply broadcast. The recency parameter mandates that only recent fruits—those hanging from one of the last $Y\kappa$ blocks—are added to a new block.

Fair ordering protocol description. Consider the protocol $\Pi_{\text{fairfruit}}^{\kappa,T,Y}$ that mines a single fruitchain with recency parameter Y. T is the modular parameter for the protocol where $T = \Theta(\kappa)$ will be based on the consistency parameter for the underlying fruitchain. For a given block chain$[b]$, the miner of the block indexes the fruits starting from one more than the index of the last fruit in the parent block (chain$[b-1]$). Fruits in the same index modulo T will be considered to be semantically or logically connected. For example, fruits at indices $1, T+1, 2T+1$, and so on will be part of the same "modular chain." This divides the single PoW chain into T modular chains. We modularize the chains using fruits instead of blocks to allow us to use the chain-quality property for Fruitchains which is given in terms of fruits. Since the modular chain of a fruit is not known while mining, honest nodes will include all transactions that are absent from at least one modular chain.

Now, in the same structure as Π_{mod}, from the T modular chains, T separate transaction orderings will be deduced which will be used as input for the Aequitas$(\cdot)$ algorithm to retrieve the final transaction ordering. Aequitas$(\cdot)$ will be run on the confirmed parts of the fruitchain i.e., with the last T blocks cut off, where $T = \Theta(\kappa)$. This will extract the fair ordering from the fruitchain. Note that the consistency parameter here is larger than the $\omega(\log\kappa)$ requirement for the standard Π_{mod}. Figure 3 contains detailed pseudocode.

Theorem 3.2. *Consider $(n, \beta, \Delta, \gamma, T, \tau, R, Y)$ such that n, Δ, T, R are polynomials in κ, $Y = \Theta(1)$; the parameters β, γ, τ are constants such that $\gamma - (\beta + \tau) > \frac{1}{2}$; for all κ, the Fruitchains protocol [38] satisfies T-consistency, and $(T, \mu > 1/2)$-chain-quality. Then, for a sufficiently large R (such that any* tx *is confirmed in all modular chains within R rounds), $\Pi_{\text{fairfruit}}^{\kappa,T,Y}(p, p_f)$ satisfies consistency, R-liveness, and γ*-batch-order-fairness *for $(n, \beta, \Delta, \tau, R)$ environments.*

```
Π^{κ,T,Y}_{fairfruit}(p, p_f)
Initialize: chain = genesis; $F = \emptyset$; TxPool = $\emptyset$

Upon receiving a valid chain′ such that |chain′| > |chain|:
  • Set chain := chain′ and output any newly seen transactions to $\mathcal{Z}$.
Upon receiving a valid fruit:
  • Add fruit to $F$.
In round $r$, upon receiving txs from $\mathcal{Z}$:
  • For each tx in txs, add (tx, $r$) to TxPool.
  • Let $F'$ be all fruits that are recent (for parameter $Y$) w.r.t. chain but
    are not in chain.
  • Set $h_{-1}$ to be the reference of chain$[-1]$ and $h'$ to be the reference
    of chain$[-\kappa]$ if $|\text{chain}| > \kappa$ otherwise set $h'$ to be the reference of
    chain$[0]$.
  • Let txincludes be the transactions in TxPool that are not in any fruit
    in modular chain $(j+1 \bmod T)$, ordered by increasing round number.
  • For $i = 1$ to $q$:
      • Pick a random nonce $\eta$ and let $(h_f, h_s) = \mathsf{H}(h_{-1}, h', \text{txincludes}, \eta)$.
      • $h_f < D_p$, then set chain := chain $\|$
        $((h_{-1}, h', \text{digest}(F'), \text{txincludes}, \eta, h), F')$, and broadcast chain.
      • $h_s < D_{p_f}$, then set fruit = $(h_{-1}, h', \text{digest}(F'), \text{txincludes}, \eta, h)$.
        Add fruit to $F$ and broadcast.
      • Break if either a fruit or block was mined.
  • Let fruitlist be a list of distinct fruits in the order of appearance in
    chain.
  • Let $\mathcal{L} = [\text{List}_1, \ldots, \text{List}_T]$ = modularize(fruitlist, $T$)
  • Output $\mathcal{L}$ to $\mathcal{Z}$. The final linearly ordered log output is taken as
    linearize($\mathcal{L}$).
-----------------------------------------------------------------
linearize($\mathcal{L}$)
  • Output Aequitas($\mathcal{L}$).
```

Figure 3: Protocol $\Pi_{\text{fairfruit}}^{\kappa,T,Y}$

4 EXTRACTING THE FAIR ORDERING

We now detail how to extract a fair ordering from the semantic chains. To accomplish this, we first distill the key structure of the Aequitas protocol from KZGJ, that provides order-fairness in the permissioned setting. We generalize their techniques to be useful in the permissionless setting we consider.

Background on Aequitas. Consider a permissioned system with n nodes, of which at most f are corrupt. We consider the synchronous Aequitas protocol which is parameterized by the order-fairness parameter $1/2 < \gamma < 1$ and works when $n > \frac{2f}{2\gamma-1}$. Roughly, Aequitas takes place in three stages that each transaction goes through. The first two stages involve each node gossiping the order in which they have received transactions and then agreeing on which nodes' local orderings to use to determine the final ordering. Intuitively, after this, all nodes have agreed on the same "data" used to order a given transaction. The third stage, dubbed the finalization stage, then, details a non-interactive algorithm used by nodes to extract the final ordering. For our purpose, we abstract away the consensus components, i.e., the first two stages, and formulate the last stage as a *finalization* problem in Section 4.1.

4.1 The Finalization Problem

We begin with some notation. For our finalization problem parameterized by $\Upsilon = (n, f, \gamma)$, we will consider as input, n ordered lists $\mathcal{L} = [\text{List}_1, \ldots, \text{List}_n]$ where an entry in a list is a set of messages. $\Upsilon = (n, f, \gamma)$ is admissible if $\frac{1}{2} < \gamma \le 1$ and $n > \frac{2f}{2\gamma-1}$. A problem instance is now given by $\Lambda = (\Upsilon = (n, f, \gamma), \mathcal{L})$. While the inputs to the finalization problem are independent of any underlying consensus problem, in the permissioned setting, intuitively, these lists describe the order in which messages were input to a particular node. List$_i[r]$ denotes the set of messages received by node i in round r (if i is honest). Lists corresponding to adversarial nodes, can of course deviate arbitrarily from their true input ordering. For messages m and m', $m \prec_i m'$ denotes that m is ordered before m' in List$_i$. The goal now, is to output a list OutputList of transactions that are ordered in a "fair" manner.

Definition 4.1 (γ-Global Preference). Given $\Lambda = (\Upsilon = (n, f, \gamma), \mathcal{L})$, for messages m and m', we say that m is *globally preferred* over m', denoted by $m' \lhd^{\Lambda} m$, if there are at least $\gamma n - f$ input lists List$_j$ in $\mathcal{L}$ s.t. $m \prec_j m'$.

We define Condorcet cycles for non-transitivity in the global preference[2], which can arise from the Condorcet paradox [3] (also

[2]As a simple example, suppose that three nodes receive transactions in the order $[a, b, c]$, $[b, c, a]$ and $[c, a, b]$. Here, each of "a before b", "b before c", and "c before a" holds for a majority of nodes, resulting in a non-transitive global preference.

see [29]). Intuitively, this also makes receive-order-fairness impossible in the general case resulting in us using the batch relaxation.

Definition 4.2 (Condorcet Cycle)**.** Given $\Lambda = (\Upsilon, \mathcal{L})$, we say that $(m_1, m_2, \ldots, m_l)$ is an l-length Condorcet cycle if $m_1 \lhd^{\Lambda} m_l$ and $m_{i+1} \lhd^{\Lambda} m_i$ for all $i \in \{1, \ldots, l\}$.

A static version of the Aequitas finalization has interesting connection to voting theory where transactions represent candidates and nodes represent voters. Here, we formulate a novel *streaming* or *online* version of the problem, which better represents the order-fairness problem at hand.

4.1.1 Streaming Finalization

The primary difference that separates the Aequitas finalization protocol from prior work in voting theory is that the list of preferences is not fully determined at the start of the protocol. In the consensus setting, the protocol is ongoing as new transactions are continuously input to nodes[3]. Clearly nodes should not have to wait for all transactions to be input to decide on the fair ordering. Here, as noted by KZGJ, future transactions can cause changes in the condensation graph. Similarly, current transactions may not have been seen sufficiently many times so far. This means that current transactions might need to wait for future ones in order to be output in a fair order. The key technical challenge is to identify when transactions can be delivered without compromising on their fair ordering. We formalize this as the streaming finalization problem (Problem 4.3). Looking ahead, within our protocols, the d semantic chains will correspond to lists in this problem.

List $\preccurlyeq$ List$'$ denotes that List is a prefix of List$'$. For $\mathcal{L} = [\text{List}_1, \ldots, \text{List}_k]$ and $\mathcal{L}' = [\text{List}'_1, \ldots, \text{List}'_k]$, we also use the notation $\mathcal{L} \preccurlyeq \mathcal{L}'$ to denote that $\text{List}_j \preccurlyeq \text{List}'_j$ for all j.

Problem 4.3 (Strong (resp. Weak) Streaming Finalization)**.** Consider an admissible $\Upsilon = (n, f, \gamma)$. At each timestep r, consider as input $\mathcal{L}^r = [\text{List}^r_1, \ldots, \text{List}^r_n]$ such that $\mathcal{L}^{r_1} \preccurlyeq \mathcal{L}^{r_2}$ whenever $r_1 \leq r_2$. At timestep r, output a list OutputList^r such that:

- (Prefix-Consistency) $\text{OutputList}^{r_1} \preccurlyeq \text{OutputList}^{r_2}$ whenever $r_1 \leq r_2$.
- (Strong (resp. Weak) Fairness of ordering) For all (m, m') such that $m' \lhd^{\Lambda} m$, where $\Lambda = (\Upsilon, \mathcal{L}^r)$, if OutputList^r contains m or m', then m is ordered before (resp. no later than) m'.

An algorithm $F(\cdot)$ solves the streaming finalization problem for an admissible Υ if for any valid input sequence $\mathcal{L}^1, \mathcal{L}^2, \ldots$, for every timestep r, $F(\mathcal{L}^r)$ satisfies prefix-consistency and fairness of ordering. Note that if $\mathcal{L} \preccurlyeq \mathcal{L}'$, then $m' \lhd^{(\Upsilon,\mathcal{L})} m$ implies $m' \lhd^{(\Upsilon,\mathcal{L}')} m$ since for any $\text{List}_j \in \mathcal{L}$ where $m \prec m'$, the corresponding $\text{List}'_j \in \mathcal{L}'$ will also have $m \prec m'$. In other words, the fairness of ordering condition will not contradict the prefix-consistency condition. Note that liveness is not included in Problem 4.3 (analogous to how delivering no transactions still satisfies "consistency" in the consensus abstraction). Liveness will be a separate protocol requirement.

Aequitas finalization [29]. Given the n input orderings from the protocol nodes, the Aequitas finalization stage builds a dependency graph of transactions. An edge (m, m') is added whenever

[3]Intuitively, this is analogous to a growing list of potential candidates in an election, which is not particularly relevant for voting theory.

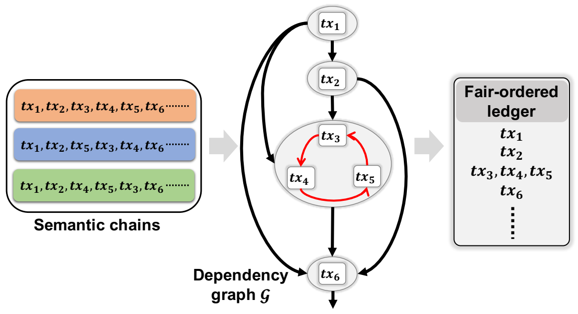
$tx_1, tx_2, tx_3, tx_4, tx_5, tx_6$ ....
$tx_1, tx_2, tx_5, tx_3, tx_4, tx_6$ ....
$tx_1, tx_2, tx_4, tx_5, tx_3, tx_6$ ....
Semantic chains
tx_1
tx_2
tx_3
tx_4
tx_5
tx_6
Dependency graph $\mathcal{G}$
Fair-ordered ledger
tx_1
tx_2
tx_3, tx_4, tx_5
tx_6

Figure 4: Simple Finalization Example

m is globally preferred over m'. Transactions can be output, when intuitively, they can no longer have incoming edges from a different strongly-connected-component (see [29]). KZGJ showed that whenever m is globally preferred to m', the Aequitas finalization will order m no later than m' in the general case, and m strictly before m' when in a setting where there are no Condorcet cycles. In turn, this directly implies that the Aequitas finalization solves our (weak) streaming finalization abstraction where the input lists for honest nodes are the actual ordering of inputs from $\mathcal{Z}$. For completeness, we also prove this below. Our streaming formalism primarily enables us to move to the permissionless setting through player replaceability.

Fact 4.4. For an admissible Υ, $\text{Aequitas}_{\Upsilon}(\cdot)$ solves the weak streaming fair-order finalization problem.

Proof. First, from the self-consistency of the Aequitas protocol, we can infer that if $\mathcal{L} \preccurlyeq \mathcal{L}'$, then $\text{Aequitas}_{\Upsilon}(\mathcal{L}) \preccurlyeq \text{Aequitas}_{\Upsilon}(\mathcal{L}')$. Furthermore, $\text{Aequitas}_{\Upsilon}(\cdot)$ satisfies the weak fairness of ordering property since the Aequitas protocol from KZGJ satisfies batch-order-fairness in the permissioned setting. □

For our purpose, we can abstract out the underlying protocol nodes and run the Aequitas finalization only on the lists. Given $\Upsilon = (n, f, \gamma)$ and $\mathcal{L} = [\text{List}_1, \ldots, \text{List}_n]$, we will use $\text{Graph}_{\Upsilon}(\mathcal{L})$ to denote the dependency graph and $\text{Aequitas}_{\Upsilon}(\mathcal{L})$ to denote the resultant outputs of applying the finalization step of the Aequitas protocol on the lists in $\mathcal{L}$. We drop the subscript when Υ is clear from context. Note that within the consensus abstraction, since different nodes may construct the semantic chains at potentially different times, the actual dependency graph of a node will be based on the parts of the semantic chains the node has confirmed, and therefore different nodes may have slightly different graphs. Figure 4 contains a simple example of the finalization. Cycles represent non-transitive preferences that will be output in the same "batch" for batch-order-fairness. Recall that transactions within a batch can still be totally ordered for execution; the only difference is their ordering is no longer considered unfair.

Remark (Minimally-relaxive batch-order-fairness). We can also show that although $\text{Aequitas}(\cdot)$ solves the weak fairness of ordering property, the version it satisfies is in some sense a minimally relaxed form of the strong fairness of ordering property. Specifically, when $m' \lhd^{\Lambda} m$, m will be ordered before m' *except* when there is a Condorcet cycle that contains both m and m'. In other words, m and m' will be output at the same index only when a paradoxical ordering prevents them from being ordered any other way. To see why this holds, recall that the Aequitas finalization protocol from KZGJ outputs m and m' together if an only if they are part of the same strongly-connected-component in the dependency graph. Consequently, in such a case, there is a path from m to m' and from m' to m which creates a cyclic ordering dependency.

Our protocols also satisfy this minimally relaxive property.

4.1.2 Player Replaceability

So far, the input lists used for $\mathsf{Aequitas}(\mathcal{L})$ were the actual transaction input orderings (for honest nodes) in a permissioned protocol. The key idea we use to move to the permissionless setting is to enable nodes to "continue" the orderings of other nodes in the system. In particular, we will fix a number of input lists, say d, at the start of the protocol. All nodes now extend the existing transaction orderings in these d lists. This can be done through any permissionless consensus technique (e.g., PoW, PoS etc). The d lists can be thought of as the input orderings of d "virtual" nodes in a permissioned network. The Aequitas finalization algorithm will now be run on the d lists to determine the final output ordering. To allow for a seamless reduction of the permissionless problem to a permissioned one, we use a new *player replaceability* lemma. Intuitively, this lemma will directly let us conclude order-fairness for our permissionless protocols, by proving order-fairness for the permissioned transformation with d virtual parties.

Transformation property. The ability to use the "virtual" party transformation can be formalized as a completeness property of the dependency graph. Informally, there should be an edge from tx to tx′ in the dependency graph of any honest node whenever $(\mathrm{tx}, \mathrm{tx}')$ satisfies the antecedent of the order-fairness definition. Note that we consider the graph built from the full output $\mathcal{L}$ of the node and not the graph that it retains after delivering other transactions. This allows us to discuss an edge between transactions even though one of them might have been delivered in the final ledger.

Property 4.5 (Completeness of Dependencies)**.** Consider $(n, \beta, \Delta, \tau, R, \gamma)$. We say that Π is *γ-dependency-complete* if for any $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \tau, R)$-permissionless execution, the following holds: If $(\mathrm{tx}, \mathrm{tx}')$ satisfies the antecedent of γ-batch-order-fairness, then for honest node N in any round r, if N outputs $\mathcal{L}$ where tx′ is in all lists in $\mathcal{L}$, then $\mathrm{tx}' \lhd^{\Lambda} \mathrm{tx}$ holds where $\Lambda = (\Upsilon = (|\mathcal{L}|, \lceil \frac{|\mathcal{L}|}{2} \rceil - 1, 1), \mathcal{L})$. In other words, $\mathsf{Graph}_{\Upsilon}(\mathcal{L})$ contains the edge $(\mathrm{tx}, \mathrm{tx}')$.

We can now describe our player replaceability lemma.

Lemma 4.6 (Player Replaceability)**.** *Consider an $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \cdot, \cdot)$-permissionless execution and an order-fairness parameter γ. Suppose that a protocol Π is γ-dependency-complete and satisfies the following consistency-like property (from Problem 4.3):*

- *If N is honest in round r, it outputs $\mathcal{L}_N^r = [\mathsf{List}_1^r, \ldots, \mathsf{List}_d^r]$ to $\mathcal{Z}$.*
- *If N is honest in any rounds $r-1$ and r, then $\mathcal{L}_N^{r-1} \preccurlyeq \mathcal{L}_N^r$.*

Suppose that an algorithm $F(\cdot)$ satisfies the strong (resp. weak) streaming finalization problem for Υ and that the linearize *function of Π is defined to be $F(\mathcal{L}_N^r)$ for node N and round r. Then, Π satisfies γ-receive-order-fairness (resp. γ-batch-order-fairness) w.r.t. $(\mathcal{A}, \mathcal{Z})$.*

Proof. For any pair $(\mathrm{tx}, \mathrm{tx}')$ that satisfies the antecedent of the γ-order-fairness definition, we are given that Π is such that tx will be γ-globally preferred to tx′ in the d lists that represent the virtual parties. Now, since $F(\cdot)$ satisfies the strong (resp. weak) fairness of ordering property, the final output ledger of an honest node in the given round will contain tx before (resp. no later than) tx′. □

Intuitively, as long as a protocol Π that satisfies the specified properties (i.e., it provides a transformation to d virtual parties), the upshot of this lemma is that now, we can run $F(\cdot) = \mathsf{Aequitas}(\cdot)$ on the d semantic chains to determine the final fair output ordering. In Section 5, we show why our protocols confer to the prerequisite of Lemma 4.6 thereby achieving batch-order-fairness.

5 SECURITY PROOFS

In this section, we prove order-fairness and liveness for our protocols Π_{mod} and $\Pi_{\mathrm{fairfruit}}$. Note that the consistency of our protocols will directly follow from the consistency of the underlying Nakamoto chain for $\beta < 1/2$. We start by introducing the key proof technique for each property.

Order-Fairness. Recall that honest nodes output the semantic chains to $\mathcal{Z}$ in an append-only fashion, and that the $\mathsf{linearize}(\cdot)$ function (which decides the final ledger) is defined as $\mathsf{Aequitas}(\cdot)$ (which solves the (weak) finalization problem) on the semantic chains. Therefore, it suffices to show dependency-completeness in order to use Lemma 4.6 and conclude batch-order-fairness.

Remark (Receive-Order-Fairness). KZGJ provided a synchronous permissioned protocol that achieves receive-order-fairness when the synchrony parameter in the external network (between clients and nodes) was $\Delta_{\mathrm{ext}} = 1$. This is in part because an adversary can no longer claim to have received transactions too far apart from honest users (otherwise they would be easily detected), thereby resulting in no Condorcet cycles. For our purpose, since the adversary's orderings become intermingled with the honest ones through the PoW mining, attempting to realize receive-order-fairness would require rewinding PoW chains which might lead to other complications.

Although the impossibility result for receive-order-fairness from KZGJ when $\Delta_{\mathrm{ext}} > 1$ (and the adversary is allowed to corrupt at least one party) carries over to the permissionless setting, we leave it as an open problem to resolve whether receive-order-fairness is achievable in the permissionless setting when $\Delta = 1$.

Liveness. We start by defining a faithfulness property. Informally, a protocol is *dependency-faithful* if some honest node has received tx before tx′ whenever an edge from tx to tx′ exists in the dependency graph of an honest node.

Property 5.1 (Faithfulness of Dependencies)**.** Consider $(n, \beta, \Delta, \tau, R)$. We say that Π is *dependency-faithful* if for any $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \tau, R)$-permissionless execution, the following holds: For honest node N in any round r, if N outputs $\mathcal{L}$ with $\mathrm{tx}' \lhd^{\Lambda} \mathrm{tx}$ where $\Lambda = (\Upsilon = (|\mathcal{L}|, \lceil \frac{|\mathcal{L}|}{2} \rceil - 1, 1), \mathcal{L})$, then there is some honest node who has received tx before tx′ as input from $\mathcal{Z}$. In other words, if $(\mathrm{tx}, \mathrm{tx}')$ is an edge in $\mathsf{Graph}_{\Upsilon}(\mathcal{L})$, then some honest node must have received tx earlier.

Now, through Lemma 5.2, we show that for a dependency-faithful Π, when $\Delta = 1$, if transactions tx and tx′ (even adversarial) are in the same Condorcet cycle, then they cannot have been received far apart. In other words, a transaction does not have to wait for an arbitrary length of time to ensure that it does not get combined with a later transaction in the same strongly connected component of the dependency graph. Consequently, this will enable transactions to be delivered in a bounded length of time allowing us to conclude liveness. Note that the specific liveness bound will be dependent on the actual protocol.

Lemma 5.2. *Consider $(n, \beta, \Delta = 1, \tau, R)$ and suppose that a protocol Π is dependency-faithful (Property 5.1). Suppose that some honest node in round r outputs $\mathcal{L}$ where* tx *and* tx′ *are in the same Condorcet cycle. Let r, r' be the first round that some honest node has received* tx *and* tx′ *respectively. Then $r = r'$.*

Proof. Let $(\mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_l)$ be a Condorcet cycle in $\mathcal{L}$ and suppose that tx and tx′ are in this cycle. Let round r_{first} be the first round that any one of $\{\mathsf{tx}_1, \ldots, \mathsf{tx}_l\}$ was received by an honest node. We can assume tx_l was first received by an honest node at round r_{first} without loss of generality by rotating the tuple cyclically. This means that any honest node that spawned no later than $r_{\text{first}} + 1$, received tx_l either in round r_{first} or $r_{\text{first}} + 1$, and any honest node that spawned in round $r' > r_{\text{first}} + 1$ received tx_l in round r'.

Now, suppose that tx_{l-1} was first received by an honest node at round r_{l-1} and suppose for the sake of contradiction that $r_{l-1} \geq r_{\text{first}} + 1$. This means that any honest node that spawned no later than r_{l-1} had already received tx_l (i.e., either tx_l before tx_{l-1} or tx_l and tx_{l-1} in the same round) and any node that spawned after received both tx and tx′ at the same time. However, since we have $\mathsf{tx}_l \lhd \mathsf{tx}_{l-1}$, this contradicts the dependency-faithfulness of Π since there needs to be some honest node that received tx before tx′. Furthermore, since no transaction in $\{\mathsf{tx}_1, \ldots, \mathsf{tx}_l\}$ was received before r_{first}, this means that tx_{l-1} was first received by an honest node in round r_{first}.

Continuing in the same fashion, the same argument shows that any transaction in $\{\mathsf{tx}_1, \ldots, \mathsf{tx}_l\}$, including tx and tx′, was first received by an honest node in round r_{first}.

Note that also applies to the case where some of the transactions are adversarial (i.e., first seen by the network as part of a block rather than transaction gossip). This is because the ledger synchrony assumption of our network ensures that transactions are provided as input to all honest nodes if some honest node outputs them to the environment. □

Equipped with these results, it will now be sufficient to show that both dependency-completeness and dependency-faithfulness are satisfied by our protocols Π_{mod} (Section 5.1) and $\Pi_{\text{fairfruit}}$ (Appendix A).

5.1 Proofs for Π_{mod}

We start by showing that Π_{mod} is dependency-complete which will allow us to conclude order-fairness.

Lemma 5.3. *Consider parameters that satisfy $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta, \gamma, T, \tau, R) = 1$. Then, $\Pi^{\kappa,T}_{\text{mod}}(p)$ is γ-dependency-complete for $(n, \beta, \Delta, \tau, R)$ environments.*

Proof. Consider any $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \tau, R)$ permissionless execution. Consider a pair (tx, tx′) that satisfies the antecedent of the order-fairness definition. For a given chain, suppose that chain$[b]$ is the first block that contains either tx or tx′ and that $|\mathsf{chain}| \geq b + 2T - 1$. Note that all blocks in chain$[b, b + T - 1]$, and consequently all modular chains, will include either tx or tx′. Now, since $\gamma - (\beta + \tau) > \frac{2}{3}$ and $\mathcal{A}$ is (τ, R)-respawning, there are more than $\frac{2n}{3}$ nodes that received tx before tx′ and are honest until all modular chains contain either tx or tx′ (i.e., until chain$[b + T - 1]$ has been confirmed). Let H be the set of these nodes. Now, from the chain-quality of $\Pi_{\text{nak}}(p)$, in the range chain$[b, b+T-1]$, more than $\frac{T}{2}$ of these blocks are mined by the nodes in H. In other words, more than half of the blocks in chain$[b, b + T - 1]$ will contain tx before tx′. Since these blocks have been confirmed in chain, more than half of the lists in $\mathcal{L}$ = modularize(chain$[: -T], T$) will contain tx before tx′. Since $\Upsilon = (T, \lceil \frac{T}{2} \rceil - 1, 1)$ is admissible, tx is globally preferred to tx′ w.r.t. $(\Upsilon, \mathcal{L})$, i.e. tx′ $\lhd^{(\Upsilon,\mathcal{L})}$ tx. We conclude that $\Pi^{\kappa,T}_{\text{mod}}$ is dependency-complete. □

Theorem 5.4 (Order-Fairness of Π_{mod}). *Consider parameters that satisfy $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta, \gamma, T, \tau, R) = 1$. Then, $\Pi^{\kappa,T}_{\text{mod}}(p)$ satisfies γ-batch-order-fairness.*

Proof. This is directly implied by Fact 4.4, Lemma 4.6, and Lemma 5.3. For the simplest case of $\gamma = 1$, we require $\beta + \tau < 1/3$. □

Next, we show that Π_{mod} is dependency-faithful.

Lemma 5.5. *Consider parameters that satisfy $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta, \gamma, T, \tau, R) = 1$. Then, $\Pi^{\kappa,T}_{\text{mod}}(p)$ is dependency-faithful for $(n, \beta, \Delta, \tau, R)$ environments.*

Proof. Consider any $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \tau, R)$ permissionless execution. Suppose that a node N is honest in round r, holds chain, and outputs $\mathcal{L}$ = modularize(chain$[: -T], T$) to $\mathcal{Z}$. Note that $\mathcal{L}$ only contains the confirmed part of each modular chain. Let $\Upsilon = (T, \lceil \frac{T}{2} \rceil - 1, 1)$ and $\Lambda = (\Upsilon, \mathcal{L})$. Suppose that tx′ $\lhd^{\Lambda}$ tx, and let chain$[b]$ be the first block that contains either tx or tx′. Then, the range chain$[b, b + T - 1]$ will determine the ordering between tx and tx′ in each of the T modular chains. Since tx′ $\lhd^{\Lambda}$ tx, more than $\frac{T}{2}$ of these blocks order tx before tx′. But, since $\beta < \frac{1}{3}$ (from $\gamma - (\beta + \tau) > \frac{2}{3}$), the chain-quality property of $\Pi_{\text{nak}}(p)$ implies that less than half of the blocks in the range chain$[b, b + T - 1]$ are mined by an adversary. This means that there is at least 1 block that contained tx before tx′ which was mined by an honest node. Therefore, some honest node received tx before tx′ from $\mathcal{Z}$. We conclude that $\Pi^{\kappa,T}_{\text{mod}}$ is dependency-faithful. □

We now use the chain-growth of Π_{nak} to bound the liveness for Π_{mod} in the following theorem.

Theorem 5.6 (Liveness of Π_{mod}). *Consider parameters such that $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta = 1, \gamma, T, \tau, R) = 1$ and $\Pi_{\text{nak}}(p)$ satisfies (T, g_0, g_1)-chain-growth. Then $\Pi^{\kappa,T}_{\text{mod}}(p)$ satisfies $(5T + 2)/(2g_0)$-liveness.*

Proof. Consider any transaction tx that is first input in round r. By the time more than $\frac{5T}{2}$ blocks are mined, all semantic chains will contain tx in the confirmed part of the chain as well as any transactions that are in the same cycle as tx. Furthermore, any transaction that is first received by an honest node at $r + 2$ will be a descendant of tx (as well as any other transaction without an edge to/from tx) in the dependency graph of an honest node. Note that such a transaction will also be in the confirmed part of the chain. Consequently, by this time Aequitas$(\cdot)$ will have delivered tx.

Now, note that for a round r' such that $r' - r \geq \frac{5T+2}{2g_0}$, we have $g_0(r' - r) \geq \frac{5T}{2} + 1$, which implies that tx will be output by round r' by all honest nodes. Consequently, Π_{mod} satisfies $\frac{5T+2}{2g_0}$-liveness. □

6 APPLICATIONS

6.1 Zero-Block Confirmation

We show how fair ordering protocols can confirm non-conflicting transactions without mining a single block in the synchronous setting. We start by defining a *soft-ordering* property.

Definition 6.1 (δ-Soft-Ordering)**.** A protocol Π satisfies δ-soft-confirmation if the following property holds: If an honest node receives tx in round r and tx$'$ in a round greater than $r + \delta$, then the final log contains tx before tx$'$.

Now, define $\mathsf{conflict}(\text{tx}, \text{tx}') = \mathsf{conflict}(\text{tx}', \text{tx}) = 1$ if tx and tx$'$ conflict with each other according to some semantic (e.g., spend the same tokens in a UTXO system). If two conflicting transactions are present in the output chain, only the first one will be executed by honest nodes to determine the current system state. Suppose that a node N receives tx in round r and is honest till at least round $r + \delta$. If N does not receive any tx$'$ such that $\mathsf{conflict}(\text{tx}, \text{tx}') = 1$ till round $r + \delta$, then all honest nodes will output tx before any conflicting tx$'$, and thus tx will be confirmed in the finalized ledger (due to the δ-soft-ordering property). This gives rise to a primitive that can be used to soft-confirm transactions. If δ is small, no blocks containing tx will be mined by this time, yet N will be able to soft-confirm tx, i.e., we get zero-block confirmation. Finally, we note that a transaction submitted by an honest node will not have any conflicting transactions at a future time, and thus can be confirmed by any other honest node within δ rounds.

It is easy to see that receive-order-fair protocols will satisfy 2Δ-soft-ordering since transactions received more than 2Δ apart will always be such that all nodes receive the first transaction before the second. Consequently, they also provide zero-block-confirmation. Further, this also implies that reorg attacks cannot happen since transactions mined into a block cannot be “reorged” and end up in the finalized chain after transactions that are yet to be mined. Note that the probability that a soft-confirmed transaction is not valid in the finalized ledger is the same as the probability that the protocol does not satisfy order-fairness, which is negligible in κ

Theorem 6.2. *If Π satisfies γ-receive-order-fairness w.r.t. $(\mathcal{A}, \mathcal{Z})$ with network parameter Δ, then it also satisfies (2Δ)-soft-ordering.*

Proof. The proof is straightforward. Suppose that a node N receives tx in round r. Then all honest nodes have received tx latest by round $r+\Delta$. Since node N received tx$'$ at round later than $r+2\Delta$, that implies that tx$'$ was received at all other honest nodes earliest at round $r + \Delta + 1$. Therefore, (tx, tx') is such that all honest nodes received tx before they received tx$'$, Since Π satisfies receive-order-fairness, tx will be ordered before tx$'$ in the final output log by all honest nodes. Consequently, Π will satisfy (2Δ)-soft-ordering.

Now, if N sees no conflicting transaction by round $r + 2\Delta$, then any conflicting transaction was received by an honest node at the earliest by round $r + \Delta + 1$. Therefore, all honest nodes received tx before they received tx$'$. Since Π satisfies receive-order-fairness, tx will be ordered before tx$'$ in the final output log by all honest nodes. Consequently, Π will satisfy (2Δ)-soft-confirmation. □

Remark. Although our protocols satisfy batch-order-fairness, and not receive-order-fairness, when $\Delta = 1$, they will satisfy (2Δ)-soft-ordering since any Condorcet cycle cannot extend past this time (Lemma 5.2), and therefore we still get zero-block-confirmation.

Network considerations. While soft-ordering is an immediate corollary of order-fairness, we emphasize that it is not a direct artifact of network synchrony in a permissionless setting. Even when transactions are gossiped and reach all nodes within a synchronous Δ period, soft-confirmation within a small delay cannot be done if the underlying consensus protocol is not fairly ordered. For instance, suppose that a node N received tx in round r and a conflicting tx$'$ in round $r + 2\Delta + 1$, i.e. all nodes receive tx before tx$'$. Still N cannot soft-confirm tx, since a miner proposing a block with tx$'$ first may not be rejected by the network (since in the view of a honest node who receives tx in round $r + \Delta$ and tx$'$ in round $r + \Delta + 1$, it is plausible that other nodes have received tx$'$ first.)

Larger soft-confirm values (small enough to still be zero-block) do not work either. Even if it is common knowledge for the current nodes that everyone received tx before tx$'$, the rejection of an adversarial miner’s block with tx$'$ first, will require this additional information. This means that the determination of whether a chain is valid is no longer only dependent on the transactions in it. Consequently, when new nodes enter the system, they will not be able to determine the correct honest chain to mine on.

Another concern is nodes not gossiping double-spends as a mechanism to prevent DoS attacks. Fortunately, gossiping only the first instance of double-spend for a token is sufficient for an honest node to know not to locally soft-confirm a transaction; other double-spends for the same token need not be gossiped to the entire network. This allows us to retain our zero-block-confirmation property will still ensuring that flooding the network will not cause a DoS attack. It also does not affect honest transactions which are not double-spent.

6.2 Decentralized Finance

The use of fair ordering protocols can greatly benefit decentralized finance applications. Consider automated market makers (AMMs) like Uniswap [7], Curve [4], etc., that allow users to exchange between tokens using an in-built price function to calculate the exchange rate. Here, the token exchange price that a user gets depends on the time her transaction executes. Using a fair ordering protocol can guarantee that users receive a fair price for their trades. Furthermore, it will also prevent miners from inserting their own transactions to take advantage of short-term changes in token exchange rates.

Order-fairness also grants fairness to applications whose incentives are based on earlier ordering. For example, in sealed-bid auctions, bids submitted before the auction close time cannot be forcefully rejected by a miner claiming to have received them later. Similarly, in an initial coin offering (ICO) token launch, if the developer wants to provide a cheaper rate for the first 1000 tokens, it will be able to do so in a fair “first come first serve” manner.

REFERENCES

[1] 2019. Ethereum Improvement Proposal 1559. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md.
[2] 2021. Blockchain 'Immutability' Dispute Sparked by Ethereum Request for Reorg Contract. https://news.bitcoin.com/blockchain-immutability-dispute-sparked-ethereum-request-reorg-contract/.
[3] 2021. Condorcet Paradox. https://wikipedia.org/wiki/Condorcet_paradox.
[4] 2021. Curve. https://www.curve.fi/.
[5] 2021. DeFi Pulse: The DeFi Leaderboard. https://defipulse.com/
[6] 2021. Reorg.wtf. https://hackmd.io/cEw2Z-QcR1yvQ8wAeQZdnQ.
[7] 2021. Uniswap. https://uniswap.org/.
[8] Ittai Abraham, Benny Pinkas, and Avishay Yanai. 2020. Blinder – Scalable, Robust Anonymous Committed Broadcast. In *CCS*. 1233–1252.
[9] Kenneth J. Arrow. 1951. *Social Choice and Individual Values*. Wiley.
[10] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the Blockchain to Approach Physical Limits. In *CCS*. 585–602.
[11] Leemon Baird. 2016. The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance.
[12] Rida Bazzi and Maurice Herlihy. 2019. Clairvoyant State Machine Replication. arXiv:1905.11607
[13] Blocknative. 2020. Evidence of mempool manipulation on black thursday: Hammerbots, mempool compression, and spontaneous stuck transactions. https://www.blocknative.com/blog/mempool-forensics.
[14] Lorenz Breidenbach, Phil Daian, Ari Juels, and Florian Tramèr. 2017. To Sink Frontrunners, Send in the Submarines. https://hackingdistributed.com/2017/08/28/submarine-sends/.
[15] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and Efficient Asynchronous Broadcast Protocols. In *CRYPTO*. 524–541.
[16] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*. 136–147.
[17] T-H. Hubert Chan, Naomi Ephraim, Antonio Marcedone, Andrew Morgan, Rafael Pass, and Elaine Shi. 2020. Blockchain with Varying Number of Players. Cryptology ePrint Archive, Report 2020/677. https://eprint.iacr.org/2020/677.
[18] Coindesk. 2021. Ethermine Adds Front-Running Software to Help Miners Offset EIP 1559 Revenue Losses. https://www.coindesk.com/ethermine-adds-front-running-software-to-help-miners-offset-eip-1559-revenue-losses.
[19] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *IEEE S&P*. 585–602.
[20] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. 2019. SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In *FC*. 170–189.
[21] Ed Felton. 2020. MEV auctions considered harmful. https://medium.com/offchainlabs/mev-auctions-considered-harmful-fa72f61a40ea.
[22] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2020. Ledger Combiners for Fast Settlement. Cryptology ePrint Archive, Report 2020/675. https://ia.cr/2020/675.
[23] Flashbots. 2021. Flashbots: Frontrunning the MEV Crisis. https://medium.com/flashbots/frontrunning-the-mev-crisis-40629a613752.
[24] Juan Garay and Aggelos Kiayias. 2020. SoK: A Consensus Taxonomy in the Blockchain Era. In *CT-RSA*. 284–318.
[25] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2017. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. In *CRYPTO*. 291–323.
[26] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT*. 281–310.
[27] Allan Gibbard. 1973. Manipulation of Voting Schemes: A General Result. *Econometrica* 41, 4 (1973), 587–601.
[28] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double spending fast payments on Bitcoin. In *CCS*. 906–917.
[29] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. 2020. Order-Fairness for Byzantine Consensus. In *CRYPTO*. 451–480.
[30] Ariah Klages-Mundt and Andreea Minca. 2020. (In)Stability for the Blockchain: Deleveraging Spirals and Stablecoin Attacks. arXiv:1906.02152
[31] Klaus Kursawe. 2020. Wendy, the Good Little Fairness Widget. Cryptology ePrint Archive, Report 2020/885. https://ia.cr/2020/885.
[32] Chenxin Li, Peilun Li, Dong Zhou, Zhe Yang, Ming Wu, Guang Yang, Wei Xu, Fan Long, and Andrew Chi-Chih Yao. 2020. A decentralized blockchain with high throughput and fast confirmation. In *ATC*. 515–528.
[33] Alex Manuskin. 2020. The fastest draw on the Blockchain: Ethereum Backrunning. https://medium.com/@amanusk/the-fastest-draw-on-the-blockchain-bzrx-example-6bd19fabdbe1.
[34] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. 2018. Smart contracts for bribing miners. Cryptology ePrint Archive, Report 2018/581. https://eprint.iacr.org/2018/581.
[35] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *CCS*. 31–42.
[36] Alex Obadia. 2021. Quantifying MEV: Introducing MEV-Explore v0. https://medium.com/flashbots/quantifying-mev-introducing-mev-explore-v0-5ccbee0f6d02.
[37] Rafael Pass, Lior Seerman, and abhi shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *EUROCRYPT*. 643–673.
[38] Rafael Pass and Elaine Shi. 2017. FruitChains: A Fair Blockchain. In *PODC*. 315–324.
[39] Rafael Pass and Elaine Shi. 2017. Rethinking Large-Scale Consensus. In *CSF*. 115–129.
[40] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with Optimistic Instant Confirmation. In *EUROCRYPT*. 3–33.
[41] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2021. Quantifying Blockchain Extractable Value: How dark is the forest? arXiv:2101.05511
[42] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. 2021. Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit. In *FC*.
[43] Michael K. Reiter and Kenneth P. Birman. 1994. How to Securely Replicate Services. *ACM Trans. Program. Lang. Syst.* 16, 3 (1994), 986–1009.
[44] Mark Allen Satterthwaite. 1975. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory* 10, 2 (1975), 187–217.
[45] Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. 2021. SoK: Decentralized Finance (DeFi). arXiv:2101.08778
[46] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. 2020. Ohie: Blockchain scaling made simple. In *IEEE S&P*. 90–105.
[47] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. 2020. Byzantine Ordered Consensus without Byzantine Oligarchy. In *OSDI*. 633–649.
[48] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2021. High-Frequency Trading on Decentralized On-Chain Exchanges. In *IEEE S&P*.

A PROOF SKETCH FOR $\Pi_{\text{fairfruit}}$

The proofs for $\Pi_{\text{fairfruit}}$ are essentially identical to the ones for Π_{mod} so we only a provide a sketch here.

Pass and Shi [38] showed that there is some recency parameter $Y = \Theta(1)$ and some $\kappa_f = \Theta(\kappa)$ such that the Fruitchains protocol satisfies κ_f-prefix-consistency, and $(\frac{5\kappa_f}{\delta}, \mu = (1-\delta)(1-\beta))$-chain-quality for any $0 < \delta < 1$. For $\beta < \frac{1}{2}$, there exists a $\delta = \delta_0 > 0$ such that $\mu > \frac{1}{2}$. This means that for $T = \frac{5\kappa_f}{\delta_0}$, Fruitchains satisfies both T-consistency and $(T, \mu > \frac{1}{2})$-chain-quality.

Now, exactly the same proofs for Π_{mod} can be used to show that $\Pi^{\kappa,T,Y}_{\text{fairfruit}}$ is both dependency-complete and dependency-faithful when $\gamma - (\beta+\tau) > \frac{1}{2}$. The only difference is that the modular chains are now taken from fruits instead of blocks. This now implies order-fairness and liveness for $\Pi_{\text{fairfruit}}$. Note that consistency once again comes for free from the consistency of the underlying Fruitchains protocol when $\beta < 1/2$.

B RELATED WORK

We describe related works in this section. The SMR abstraction has been studied in literature for several decades but almost no prior permissioned protocols, and no permissionless protocols consider any notion of fairness in transaction ordering.

Fair ordering protocols. Very little research has been done on fair ordering consensus protocols. Hashgraph [11] provided an informal description of a first-in-first-out (FiFo) transaction ordering property. KZGJ [29] formally defined (receive)-order-fairness (resp. batch-order-fairness) as the property of ordering tx_1 before (resp. no later than) tx_2 globally if a large fraction of nodes (parameterized by γ) received tx_1 before tx_2 in their local ordering. KZGJ provided permissioned protocols that achieved their definitions. Concurrently to KZGJ, works by Kursawe [31], and Zhang et al. [47] gave an alternative definition for ordering fairness which we compare below. Here, the fair ordering problem is tackled at a *pre-protocol* stage, i.e., the ordering is determined independently and before the actual consensus protocol is run. Zhang et al. [47] also found interesting connections to social choice theory, including the works of Arrow [9] and Gibbard-Satterthwaite [27, 44].

Comparison to other fairness definitions. Kursawe [31], and Zhang et al. [47] independently defined a different notion of fair ordering (called *timed relative fairness* in [31] and *ordering-linearizability* in [47]) in the permissioned setting where tx_1 is ordered before tx_2 if all honest nodes have seen tx_1 before *any* honest node has seen tx_2. A similar property was also stated as a desideratum in [24] although no protocols achieving this property were given. Note that this is strictly weaker than the receive-order-fairness property from KZGJ and in Section 2.1. Further we find that its antecedent can easily be made false, and therefore it is not enough to prevent natural order-manipulation attacks. This is because it enforces no guarantees on transactions whose receive times are globally interleaved. An adversarial transaction that attempts to front-run an honest user transaction, will end up getting interleaved with the user transaction in its receive times. Consequently, for most practical attack vectors, the antecedent of ordering-linearizability will be false, making the definition vacuous and not particularly useful. We also note that this definition lacks a parameterization by γ, compared to the order-fairness definition.

Another natural definition is based on median timestamps; specifically, tx_1 is ordered before tx_2 if the median receive time for tx_1 is smaller than that for tx_2. However, KZGJ showed that that this does not model a FIFO notion as a single adversarial node can cause tx_2 to be ordered earlier even when all nodes have received tx_1 earlier.

We therefore chose to extend the order-fairness definition from KZGJ. We emphasize however, that our techniques are general enough to be adapted to provide other notions of fair ordering.

Alternate ordering techniques. We compare the general idea of fair ordering protocols with several prior techniques used to constrain adversarial influence over transaction ordering.

Censorship Resistance. An extensive line of research ([26, 37, 38] among others) in the past few years has considered the consensus problem in the permissionless setting. By design, most permissionless protocols achieve *censorship resistance*, which ensures that any transaction submitted by an honest user will eventually be confirmed by the network. Despite this, no guarantees are provided on how the transaction is eventually ordered.

Content Hiding. Some protocols make the use of threshold encryption [15, 35, 43], or a commit-and-reveal scheme [8, 14], to order transactions before their content is revealed in an attempt to prevent adversarial ordering. In practice, this might still not prevent censorship and reordering based on transaction metadata (user identifier, client IP address etc) or front-running based on information received through side channels from colluding clients [29]. Furthermore, an increasing number of attacks (e.g., displacement [20] and certain backrunning [33]) do not depend on transaction contents and therefore will not be stopped by these techniques.

Random Ordering. It is important to distinguish a protocol that chooses a *random* ordering from a protocol that chooses a *fair* ordering. A *random* ordering protocol chooses a random ordering of transactions in the current pool, in an attempt to avoid any adversarial manipulation. On the other hand, a fair ordering protocol will guarantee that transactions which arrive earlier will be sequenced earlier by the network (instead of in a random order).

Even in a hypothetical perfect random ordering protocol where no metadata or side-channel information is leaked and where adversarial miners are provably forced to choose a random ordering of transactions in the mempool, there is still a 50% chance that an adversarial transaction gets ordered before an honest one even if the honest one was propagated earlier. In addition, the adversary can flood the network with many transactions to exponentially increase the probability that at least one is ordered before the honest transaction (e.g., clogging attacks [41]). We highlight that even if the adversary has no control over the ordering of transactions in the pool, it is able to implicitly influence the ordering by introducing its own adversarial transactions.

Random ordering is especially problematic if the adversary can use user transaction metadata to construct its own transaction, or if the adversarial transaction does not need to depend on the contents of any honest user's transaction (e.g., displacement attacks [20] where the adversary wants to be the first one through the door to for instance, buy an asset with only limited quantities available, or get the best price in an auction). This is not a hypothetical problem as such a flooding attack was performed during the BZRX token launch [33] to guarantee a better token buy price for the adversary. Therefore, a random ordering protocol is not sufficient to prevent order-manipulation attacks.

In contrast, our fair ordering protocols ensure that an adversary cannot tip the scales in its favor by creating many transactions.

Latency reduction methods. Reducing the confirmation latency of longest-chain protocols has been the subject of intensive recent research [10, 22, 32, 46]. Two particularly relevant works, Prism [10] and Ledger-Combiner [22], also use multiple parallel chains to achieve faster confirmation. They can confirm honest transactions within a constant multiple of the network delay Δ, much faster than Bitcoin, where the latency is a multiple of the security parameter. However, in both methods, the constants are large and become worse as the adversarial power increases. Other methods [32, 40] offer low latency only under optimistic conditions, and fall back to standard Bitcoin latency when there are any adversarial actions. In comparison, our soft-confirmation method can confirm honest transactions in 2Δ rounds independent of the tolerable adversary ratio or whether there is an active attack. Conceptually, our soft-confirmation method can confirm transactions that are yet to get into any block, which is impossible with the other methods. Surprisingly, Ledger-Combiner, which was specifically designed to get low latency when combining transactions from multiple ledgers is slower than the soft-confirmation our constructions achieve as a byproduct. Recent work in distributed systems [12] has shown how to utilize "non-skipping time-stamps" to locally commit transactions as soon as nodes know that no conflicting transaction can be ordered earlier, and termed this property *clairvoyance*, emphasizing the look-ahead ability. [12] proposes protocols that achieve clairvoyance, but are specific to the permissioned setting, and even there, are not designed for, and hence do not achieve, our stronger definition of fair ordering.