BlindPerm: Efficient MEV Mitigation with an Encrypted Mempool and Permutation

Alireza Kavousi
University College London

Duc V. Le
Visa Research*

Philipp Jovanovic
University College London

George Danezis
Mysten Labs and University College London

Abstract

Maximal extractable value (MEV) in the context of blockchains and cryptocurrencies refers to the highest potential profit that an actor, particularly a miner or validator, can achieve through their ability to include, exclude, or re-order transactions within the blocks. MEV has become a topic of concern within the Web3 community as it impacts the fairness and security of the cryptocurrency ecosystem. In this work, we propose and explore techniques that utilize randomized permutations to shuffle the order of transactions of a committed block before they are executed. We also show that existing MEV mitigation approaches using an encrypted mempool can be readily extended by permutation-based techniques, thus providing multi-layer protection. With a focus on BFT consensus, we then propose BlindPerm, a framework enhancing an encrypted mempool with permutation at essentially no additional overheads and present various optimizations. Finally, we demonstrate how to extend our mitigation technique to support PoW longest-chain consensus protocols.

1 Introduction

Although blockchain and in particular cryptocurrencies initially emerged with a focus on presenting a robust financial ecosystem, the lack of attention to the issue of *ordering manipulation* turns out to be problematic. This issue became more serious with the introduction of decentralized finance (DeFi), and in particular decentralized exchange (DEX), as a platform to offer financial services without putting trust in intermediaries such as banks or exchanges. It is shown that the underlying execution mechanism behind these platforms called smart contract [51] has some level of transaction ordering dependence [33]. The threat essentially stems from the fact that such dependence allows ordering manipulation to cause a major impact on the traded crypto asset by the actors [19]. So, the attacker can take advantage of such opportunities to achieve some benefit beyond the regular transaction fee and block reward. An illustrative scenario on DEX known as *sandwich attack* involves front-running (*i.e.*, placing a transaction before) and then back-running (*i.e.*, placing a transaction after) a victim's transaction to exploit the forced price fluctuations in the traded asset at the cost of victims' loss. Generally speaking, such profits made via including, excluding, or re-ordering transactions within blocks is described as maximal extractable value (MEV)[19]. While utilizing MEV is a risk-free process for the block proposers (*i.e.*, miner, validator) due to their centralized role in preparing the

*The main part of the work was conducted while the author was at the University of Bern.

block, it can potentially occur by any vigilant actor in the system known as *searcher* [1] with (a priori) knowledge of a profitable transaction. This comes from the public nature of the blockchain that offers such knowledge, with speedy connections giving a better advantage. Note that performing the sandwich attack properly requires the correct placement of the transaction with respect to that of victim; otherwise, it may lead to a loss for the attacker. With the recent adoption of Flashbots [1], this process has become easier due to their capability to offer front-running as a service [41].

A range of solutions has been proposed to mitigate the negative effects of MEV. Arguably the most promising ones are so-called *order fairness* that consider providing a fair ordering for the transactions that appear in a finalized block. The notion of fairness, however, is not something universally agreed upon and may have various interpretations. Timed-order fairness [31, 34, 30, 13] and blind-order fairness [55, 36, 17] are two well-known terms in the literature that aim at providing protection at the consensus layer. In essence, the former determines the final ordering of transactions according to their arrival times at the system and the latter hides the content of transactions until their ordering is fixed, preventing any conscious manipulation in the meantime. It is worth pointing out that dependency on time helps to model the real world by appreciating the same flow that is occurring in reality. However, blind ordering cannot achieve such strong guarantees but try to make the manipulation hassle for the attacker by cutting the relation between the ordering and the content of transactions. Therefore, the final ordering might not appreciate the real world as the attacker still can (blindly) impact the ordering at their will.

Given that the existing proposals following timed-order fairness have expensive configuration costs and usually demand low fault tolerance and high latency, considerable attention has been devoted to realizing blind ordering in various settings [28]. However, there are two principle problems with this approach that might affect its usefulness. First, the block proposer can easily front-run other transactions regardless of their contents. Consider a scenario where a popular non-fungible token (NFT) is dropped and the block proposer decides to buy some NFT. They can place their transaction in the first spot, front-running others and buying at a lower cost. Second, blinding is concerned with the payload of the transaction, and the leakage of side information (*i.e.*, metadata such as gas price or address) may be enough for the attacker to carry out the attack.

We observe that another method to cut the relation between the ordering of transactions and their content is via applying a *random permutation* on the (committed) block before execution. This technique turns out to be a useful strategy in mitigating the power of the block proposers in imposing their desired ordering. We can consider it as a solution at the execution layer where the permutation mitigates any ordering manipulation already occurring in the committed block before it affects the state of the system. It essentially has the same effect as an encrypted mempool in mitigating the information asymmetry in the blockchain state between the user and the validator.[1] Moreover, another benefit of deploying permutation is its usefulness to protect against those types of MEV that negatively affects users, particularly *sandwich* attack. This is due to the fact that a random permutation can turn a definite profit into a possible loss simply by shuffling the front-running and the back-running transactions, creating a dilemma for the attacker. We take a step further and argue about the importance of a combinatory solution, where the blind ordering and shuffling enhance each other in a complementary manner. That is, a random permutation on the committed block mitigates both issues mentioned with the blind ordering, and an encrypted mempool - that is the core of blind ordering – provides *spam protection* for an effective permutation.

Contributions. The contributions of this work are as follows.

- We introduce randomized permutation as a mechanism for MEV mitigation at the execution layer

[1] In fact, the knowledge of the new state of blockchain does not give any advantage to the validator in choosing a desired order.

and propose an efficient construction for BFT style consensus. This technique reduces the power of block proposers in imposing their desired ordering and, in particular, hampers sandwich attacks.

- We introduce BlindPerm, a framework that combines encrypted mempools with transaction shuffling to provide enhanced MEV protection. Particularly noteworthy is the fact that the permutation-based enhancement can come at no additional cost in comparison to a regular encrypted mempool without shuffling.
- We present several optimization techniques that might be of independent interest including *selective encryption* which enables users to only protect their MEV-potential transactions through encryption, thereby offering efficiency gains over a fully encrypted mempool.
- We show how to extend the permutation-based approach to proof-of-work (PoW) longest-chain protocols.

2 Background

2.1 Threat Model

We consider the setting of a Byzantine fault-tolerant (BFT) system. In this setting, there are n parties at most t of which are corrupted by a computationally bounded adversary to do any arbitrary behavior. We assume the existence of authenticated point-to-point channels between each pair of parties. As a permissioned system, the set of BFT parties is fixed and they are known to each other prior to the protocol execution. The network model is partially synchronous [23], meaning that it may oscillate between periods of synchrony and asynchrony. The common way to treat this is to consider some unknown point as global stabilization time (GST), where it triggers the periods of synchrony that allow message delivery within a known time bound Δ. The optimal fault tolerance in this setting is shown to be $t < n/3$ [5].

2.2 Consensus

Consensus is a fundamental problem that aims at providing a set of n parties on possibly different inputs with a common decision despite adversarial behavior by at most t of them [39, 35]. The two core properties of a consensus protocol are safety and liveness. The former ensures all the honest parties decide on the same value and the latter ensures an honest party terminates with a valid value. Byzantine broadcast is a pivotal type of consensus that enables a sender to send its input to the other parties such that all the honest ones terminate with the same value. It has been shown by the famous impossibility result of [24] that there is no deterministic protocol for Byzantine broadcast in the partial synchrony (and asynchrony) tolerating even one fault. This comes from the fact that in asynchrony it is not possible to distinguish a slow sender from a faulty one, violating the safety in the event of making a decision or liveness in the event of waiting to hear eventually. Bracha's broadcast [9] circumvents this by relaxing the liveness property such that either all the honest parties terminate or no one does.

State Machine Replication (SMR). Among different formulations of consensus problem, state machine replication (SMR) [44] is notable as it enables agreement on *ever-growing* inputs received from external users. It can be naively instantiated by repeated execution of a single-shot consensus protocol, enabling parties to agree on an ordered sequence of inputs proposed by each proposer. Blockchain protocols implement this type of consensus in two general forms: BFT style such as

PBFT [16] or Hotstuff [54], and longest-chain style such as Bitcoin [38] or Ethereum [51]. In the BFT style consensus, which is the focus of this work, BFT parties (*i.e.*, validators) get to agree on a proposal including a batch of transactions. The protocol is typically operated view-by-view and derived by a leader (*i.e.*, block proposer) that is elected in a deterministic way. Although the surge in popularity of (permissioned) blockchain protocols has led to considerable innovations in the literature of SMR as its core [54, 47, 20], the following protocol flow is a common paradigm in almost all. First, the leader prepares a block of transactions and sends the proposal to all the other validators. Second, each validator votes on the proposal if it is properly formed and sends the vote back to the leader. Third, upon collecting $n - t$ votes the leader creates a quorum certificate (QC) and disseminates it to the validators. This process repeats more than once in each view of the protocol to commit (*i.e.*, make a decision). The protocol might require a *view-change* if the validators suspect the leader is faulty, moving to the next leader.

2.3 Secret Sharing

A (t, n) Shamir secret sharing [46] allows a dealer to distribute a secret s among a set of n shareholders via $\mathsf{SS.Share(s)} \rightarrow s_1, \ldots, s_n$, such that it can only be reconstructed uniquely by at least $t+1$ shares $\mathsf{SS.Combine}(s'_1, \ldots, s'_{t+1}) \rightarrow s$, while no information on the secret is revealed otherwise.

Verifiable Secret Sharing (VSS). The basic (t, n) threshold secret sharing scheme of [46] is passively secure, meaning that it works as long as the participating parties run the protocol as specified. In the Byzantine setting, the dealer needs to ensure parties about the correctness of sharing and later parties ensure a reconstructor about the correctness of their released shares. Verifiable secret sharing (VSS) does that by having the dealer commit to the sharing and broadcast it to the parties. Starting from Feldman VSS [24], there have been considerable efforts in the literature to develop VSS schemes with better efficiency in various network models. These are particularly concerned with two aspects of VSS schemes including broadcasting a polynomial commitment [29] to enable share verification and having a complaint phase to deal with any faulty/missing share. In asynchrony, the situation becomes complicated as we need to cope with the *share recovery* in case the leader completes and goes offline without having all the honest parties end up with valid shares. So, the inherent necessity of $n-t$ acknowledgments for making progress could lead to having only $n - 2t$ honest parties receive valid shares. Asynchronous VSS (AVSS) protocols deal with this situation by going through a share recovery phase that enables a party eventually recover its valid share with the help of $t + 1$ honest parties.

Publicly Verifiable Secret Sharing (PVSS). To extend the scope of verifiability to the public, not only participating parties, PVSS schemes deploy cryptographic primitives such as encryption and non-interactive zero-knowledge proofs (NIZKs). This, in turn, enables anyone to verify the correctness of the distribution phase by the dealer and the reconstruction phase by the set of shareholders. The immediate consequence of public verifiability is getting rid of the complaint phase and therefore reducing latency in VSS schemes at the cost of incurring higher computational overhead to the protocol. A commonly used PVSS scheme in the literature is SCRAPE [45, 14] with the following abstract. To share a random secret s, the dealer runs $\mathsf{PVSS.Share}(s, \{pk_i\}_{i\in[n]})$ and outputs encrypted shares $\{\hat{s}_i\}_{i\in[n]}$ with a proof of correctness π_s. This proof enables anyone to verify the consistency of the shares (*i.e.*, they are evaluations of the same polynomial) and the validity of the ciphertexts (*i.e.*, they contain valid shares). Each shareholder can invoke $\mathsf{PVTSS.Decshare}(\hat{s}_i, sk_i)$ to output a decrypted share $\tilde{s}_i$ with a proof of correctness π_i (*i.e.*, showing correct decryption). Upon gathering $t + 1$ valid decrypted shares, anyone can reconstruct the secret s via $\mathsf{PVTSS.Combine}(\tilde{s}_1, \ldots, \tilde{s}_{t+1})$.

2.4 Threshold Cryptography

Distributed Key Generation (DKG). A DKG protocol shares a uniformly distributed secret sk among n parties such that each receives a partial secret key sk_i, a partial public key pk_i, and a common public key pk. DKG is needed to jointly perform cryptographic operations like decryption or signing. Once it is executed, the resulting keys can be used by parties polynomially many times.

Threshold Encryption. In a (t, n) threshold encryption scheme, one can run the algorithm $\mathsf{TE.Enc}(pk, m) \rightarrow c$ to encrypt a message m under a public key pk resulting from a DKG among a set of n parties. Each party then runs the algorithm $\mathsf{TE.Pardec}(sk_i, c)$ using its own secret key sk_i to obtain a partial decryption pd_i. Finally, c can be decrypted by any proper set of partial decryptions $\mathsf{TE.Dec}(pd_1, \ldots, pd_{t+1}, c)$. Note that it is also possible to verify partial decryption via an additional algorithm $\mathsf{TE.Verify}(pd_i, c)$.

Threshold Signature. In a (t, n) threshold signature scheme, any subset of n parties of size $t + 1$ can jointly sign a message m by having each run $\mathsf{TS.Parsign}(sk_i, m)$ to produce a partial signature ps_i, and then $\mathsf{TS.Sign}(ps_1, \ldots, ps_{t+1}, m)$ to produce the signature σ. Anyone can verify a partial signature via $\mathsf{TS.Verify_1}(pk_i, ps_i, m)$, and the signature via $\mathsf{TS.Verify_2}(pk, \sigma, m)$.

3 MEV Mitigation

MEV mitigation is a rather controversial subject. Some believe it is necessary not only because of the possibility of causing a considerable loss to the users but also the threat of making the system centralized due to the potential benefit of becoming a leader. On the flip side, some may argue about the positive side effects of MEV phenomena, *e.g.*, to equilibrate the price of assets across different exchanges deploying arbitrage opportunities. Such mindset makes practitioners mostly in favor of making MEV accessible by facilitating the process and exploring the ways to properly distribute benefits across the traders [12, 50, 28]. This paper follows the former approach and focuses on MEV mitigation. To do so, there are two generic methods at the consensus layer, namely timed-order fairness and blind-order fairness.

Timed-order Fairness. The consensus problem that is at the core of blockchain protocols known as SMR, traditionally does not aim at getting parties agree on a *specific ordering*, but a *total ordering* where all the honest parties are guaranteed to end up with the same sequence of transactions. One way to deal with this is to augment its requirements with an *order fairness* property, satisfying a specific ordering of transactions obtained by the honest parties as part of the consensus. A long line of works in the literature considers *time* as a promising measurement to reason about order fairness. Although different definitions have been proposed in this regard, the backbone of almost all proposals is to consider the ordering of transactions according to the time they arrive at a majority of the participating parties.

It was shown by two concurrent works of [34, 31] that arguably the most natural definition of fairness known as *receive/relative order fairness*, is impossible to achieve. This notion essentially states that for any two transactions tx and $\mathsf{tx'}$, if some majority of nodes receive the former sooner than the latter, tx should be ordered before $\mathsf{tx'}$. The impossibility result is due to the so-called Condorcet's cycle/paradox [26], preventing parties to agree on a fair ordering of transactions even when all behave honestly [31]. [2] The impossibility result necessitates the adoption of other variants of timed-order fairness. Kelkar et al. [31] relaxed their definition to capture *batch order fairness* by

[2] Such a cycle shows up intransitivity in the majoritarian relations, yielding a paradox in selecting a single winner.

making "before" to "no later", treating such transactions in batches with relative ordering.[3] In fact, the batch order fairness sidesteps the impossibility result by allowing output transactions in batches and ignoring the possible unfairness resulting from the cycles in each batch. Although timed-order fairness captures reality by enforcing the actual ordering and seems to be a solid countermeasure against MEV, it lacks enough protection against a searcher with a strong network connection, observing a transaction early and attempting to front-run a victim's transaction afterward.[4] Moreover, some proposals like [31, 30] demand a low fault tolerance and high latency to work properly which affect their applicability in practice.

Blind-order Fairness. Another approach towards MEV protection is achieved by making the ordering of transactions within a committed block independent of the content of transactions initially received, resulting in a *blind ordering*. The main way to realize this is via creating an *encrypted mempool*, containing the set of unordered encrypted transactions. In fact, each user simply encrypts their transaction before broadcasting it to the network of validators. The transactions remain encrypted until they get committed (*i.e.*, their inclusion is guaranteed) with a deterministic ordering. Then, any proper set of validators can decrypt each transaction included and then execute the committed block. One important consideration is to ensure a *guaranteed decryption* for each encrypted transaction before a commit by validators; otherwise, it may either lead to an encrypted transaction being buffered indefinitely [36], or the user being able to affect the ordering according to its view of the system [4]. There have been some attempts in the literate to implement such encrypted mempool in a distributed manner with majority of them following the traditional commit-reveal paradigm, making it robust (to ensure guaranteed decryption) either by relying on threshold security [36, 55, 4] or time-based cryptography [43, 21, 17]. It has been shown that blind ordering (with threshold security) is currently offering the most desirable qualities compared to the other proposals for MEV mitigation [28]. Low latency, wide coverage, and reasonable performance are the main features that have placed this approach at a promising position.

3.1 Encrypted Mempool

With a focus on threshold security, Malkhi and Szalachowski [36] present four approaches to build up an encrypted mempool, including threshold cryptography, VSS, secret sharing with post-verification, and Hybrid. Threshold cryptography (*e.g.*, threshold encryption) allows users to encrypt their transactions tx under the validator's common public key pk with $\mathsf{TE.Enc}$, where it can only be decrypted by threshold of $t+1$ validators with $\mathsf{TE.Dec}$. Secret sharing, however, requires each user to first pick a symmetric key $\mathsf{tx\text{-}key}$ to encrypt the transaction and then secret share (or confidentially disperse) the key to the validators. This procedure can also be applied with threshold cryptography by encrypting $\mathsf{tx\text{-}key}$ under the common public key. Thus, all the aforementioned approaches can be abstracted away with two functionalities of $\mathsf{Disperse(tx\text{-}key)}$ and $\mathsf{Retrieve(tx\text{-}key)}$. In this work, we present another approach using PVSS and elaborate on an optimized variant that is relevant for this purpose (we discuss this later in Section 6.1). When it comes to efficiency, PVSS may not be the best option as it includes non-trivial cryptography and aims at public verifiability, resulting in higher overhead. However, it provides lower latency (compared to VSS) and does not require running a DKG (compared to threshold cryptography). Communication-wise, the relatively small size of validators in a BFT style consensus (*e.g.*, 100 - 200) makes using PVSS a reasonable option compared to VSS. Moreover, removing the DKG helps accommodating dynamic participation of the

[3] This was originally presented as "block" order fairness [31]. To avoid misconception, the authors in the follow-up work of [30] changed it to "batch" order fairness.

[4] Such searcher may have many more TCP connections than the average users. So they become aware of the victim's tx early enough to lunch the attack.

validators. In what follows, we briefly investigate two new approaches proposed in [36] including secret sharing with post-verification and Hybrid.

Secret Sharing with Post-verification. VSS aims at ensuring the uniqueness, meaning that invoking SS.Combine with any threshold number of shares results in the same outcome, and completeness, meaning that any honest party receives a (distinct) valid share from SS.Share. The recent work of [36] adapts a technique introduced in [52] to relax the requirements and only offer uniqueness. To do so, the dealer runs SS.Share, combines all shares in a Merkle tree, certifies the root, and sends with each share a proof of membership, *i.e.*, a Merkle tree path to the root. When a party receives a share, they should verify the Merkle tree proof against the certified root before acknowledging it. Moreover, after running SS.Combine, each party re-encodes the Merkle tree with the reconstructed secret and compares it with the data sent by the dealer. If the comparison fails, the dealer is faulty. Observe that here the signed Merkle root acts as a public commitment to somewhat relax the use of polynomial commitment. This protocol is the fastest as it uses the efficient and trivial primitives. Note that the sharing completes for each transaction when there are $n-t$ acknowledgments to ensure $t+1$ honest validators have received consistent shares, incurring latency. Another issue mentioned in [36] is the possible impact of some specific subset of $t+1$ validators on the latency of SS.Combine. More precisely, since there is no guarantee that all honest validators receive their shares, SS.Combine may not be run by the fastest $t+1$ validators and depend on a specific subset.

Hybrid. In order to address the dependency issue, the authors in [36] proposed using a hybrid design where the secret sharing with post-verification is augmented with threshold cryptography, enabling any subset of $t+1$ validators to perform the decryption. Moreover, to maintain safety the protocol requires the results recovered from the SS.Combine be equal to TE.Dec. To do so, each validator can make use of $t+1$ secret shares or partial decryptions to check both approaches have the same output. They just need to re-encrypt the key and re-encodes the Merkle tree and check with those originally sent by the dealer.

4 On the Use of Permutation

As mentioned, the prime goal of creating an encrypted mempool is taking away the possibility of MEV extraction either by the block proposer (validator) or any searcher. When looking closely, however, even leveraging a *perfect* encrypted mempool – where there is no leakage of data about the transaction and its sender – cannot prevent the block proposers from *inserting* some transaction at their desired spot in block, *e.g.*, the first entry of block. Observe that the success of such action does not necessarily rely on other transactions and stems from sole control over the block production. As a type of MEV [19], such insertion may result in the validator making an unnecessary profit, a profit besides the transaction fee and block reward. For example, consider a price slippage caused by some large trade on an exchange. The validator can simply place their transaction at the first entry to be able to take advantage of such an opportunity prior to others.[5] Although this strategy likely does not cause a loss to users, it reduces the level of fairness in the system due to having the validator exert their additional power to make a profit.

However, it is crucial to perform such permutation *safely* and in a *secure* way. To ensure the former, the honest validators must perform the same permutation on the same (committed) block. To ensure the latter, the seed (*i.e.*, randomness) for permutation should be unpredictable prior to the commit phase, and unbiasable by an adversary controlling the block proposer (and possibly a set of validators). In the following section, we propose a protocol that satisfies these properties.

[5]The validator may get to know about such opportunities irrespective of the transactions in the mempool, making the encryption useless.

4.1 Protocol Description

We present our protocol in four main steps as follows. The key insight is leveraging the finality of a BFT consensus to have each honest validator safely apply a permutation with secure shared randomness computed afterward.

Step 1 – Submission. Each user broadcasts their transaction tx to the network of $n = 3t + 1$ validators.

Step 2 – Committing to the total ordering. The protocol operates in views. Let r be the current view number where a designated validator acts as the leader to propose a block B_r. The block contains a list of submitted transactions in the mempool. Upon receiving $2t + 1$ votes, the proposed block B_r becomes committed by all the honest validators in the system. The protocol enters the next view upon committing the proposed block and having $t + 1$ valid partial signatures for deriving the seed or issuing $2t + 1$ complaints against the leader.

Step 3 – Deriving the seed. To generate secure shared randomness we assume validators already run a DKG protocol [40] to obtain the key materials. The randomness is securely derived by having validators jointly produce a threshold signature on the view number, using a unique signature like BLS [8]. Thus, when a validator observes the block B_r has become committed, they send a partial signature of form $H(r)^{sk_i}$ by invoking $\mathsf{TS.Parsign}(sk_i, r)$ to others. Let I be the set of indices of $t+1$ valid partial signatures. Anyone can run $\mathsf{TS.Sign}(\{ps_i\}_{i\in I}, r)$ to produce the signature $\sigma_r = H(r)^{sk}$ using Lagrange interpolation in the exponent. Finally, the seed for permuting block B_r is computed as $\mathsf{seed_r} = H(\sigma_r)$, where $H(\cdot)$ is a cryptographic hash function.

Step 4 – Execution. After computing the seed[6], each validator locally perform $\mathsf{Permute}(\mathsf{seed_r}, B_r)$ to randomly shuffle the ordering of the transactions in B_r, resulting in a permuted block B'_r. The permuted block is then added on-chain for execution. A standard permutation algorithm is given in Algorithm 1.

4.2 Analysis

Lemma 1. *The proposed protocol satisfies safety, liveness and a secure permutation.*

Proof. The safety and the liveness of the protocol directly follow those of underlying consensus as we treat it in a black-box manner. Thanks to the finality of the BFT consensus, the honest validators apply the permutation on a committed block they all already agree on. The seed is pseudorandom with hash function modeled as random oracle [27], and unpredictable to the validator proposing the block before it gets committed, guaranteeing a secure permutation. □

5 BlindPerm

In the previous section, we showed how to construct a protocol to securely permute the set of transactions in a committed block, removing the advantage of block proposer in imposing a desired ordering and possible sandwich attack. However, it has been argued that a broad scope of MEV comes from the availability of information about transactions, either those that are already submitted on public mempool or the ones observed early by a powerful searcher [10]. Such information could directly affect the users by facilitating the MEV for the validator or searcher through creating

[6] A pseudorandom generator may need to apply on the seed first to produce a long random string.

Algorithm 1 Permute [3]

Input: An array a with k elements
Output: A random permutation on the array a
for $i := k$ *downto* 2 *by* -1 **do**
  $j :=$ Knuth-Yao$(i) + 1$;
  swap(a_i, a_j);
end
Procedure: Knuth-Yao(k)
Input: A positive integer k
Output: Uniform$[0, k-1]$
$u := 1$; $x := 0$;
while *true* **do**
  while $u < k$ **do**
    $u := 2u$;
    $x := 2x +$ randbit;
  end
  $d := u - k$;
  if $x \geq d$ **then**
    return $x - d$;
  else
    $u := d$;
  end
end

dependent transactions or even censoring an undesirable transaction. Consider the scenario where a searcher detects a victim's transaction and submits a corresponding transaction to the mempool. Anyone observing the mempool may subsequently see such an opportunity and do the same. This essentially leads to reducing the effectiveness of permutation as the chances of the victim's transaction getting frontrun by one of the attackers' transactions nevertheless increases.[7] Since a sole permutation-based solution cannot offer suitable protection in these situations, we propose Blind-Perm, a framework augmenting the permutation on top of an encrypted mempool. Given that an encrypted mempool may still leak some metadata related to identity or content, this combination is complementary and has the additional benefit of reducing such negative effects, offering the best of both worlds. From the attacker's perspective, permutation moves sandwich attack from being riskless to risky, and blinding moves the censorship from being flexible to an all-or-nothing.

Our framework includes two categorizes depending on the way the permutation seed is generated. The first category relies on the validators, while the second one relies on the users to generate the seed. Interestingly, the latter allows obtaining the seed for free by piggybacking on the encrypted mempool.

5.1 Validators Generating the Seed

In this section, we extend the construction proposed in Section 4.1 to establish an encrypted mempool. We make use of threshold cryptography to both let validators compute the seed and users encrypt their transactions.

[7] Any relative ordering of the transactions is equally probable and having more dependent transactions from attackers increases the overall chance of frontrunning.

Step 1 – Submission. Each user encrypts a transaction tx under validators' common public key $\mathsf{TE.Enc}(pk, \mathsf{tx})$ and broadcasts the ciphertext c to the network of $n = 3t + 1$ validators.

Step 2 – Committing to the total ordering. The protocol operates in views. Let r be the current view number where a designated validator acts as the leader to propose a block B_r. The block contains a list of encrypted transactions in the mempool. Upon receiving $2t + 1$ votes, the proposed block B_r becomes committed by all the honest validators in the system. The protocol enters the next view upon committing the proposed block, having $t+1$ partial signatures for deriving the seed, and $t + 1$ partial decryptions for each committed transaction, or issuing $2t + 1$ complaints against the leader.

Step 3 – Decryption and deriving the seed. When a validator observes the block B_r has become committed, they produce a decryption share $\mathsf{TE.Pardec}(sk_i, c)$ for each committed tx and a partial signature $\mathsf{TS.Parsign}(sk_i, r)$ for their contributions towards seed. It then sends the partial decryptions together with partial signature to others. Each validator can obtain transaction tx and the seed $\mathsf{seed_r}$ by running $\mathsf{TE.Dec}$ and $\mathsf{TS.Sign}$ upon receiving $t + 1$ valid partial contributions, respectively.

Step 4 – Execution. After computing the seed, each validator locally performs $\mathsf{Permute}(\mathsf{seed_r}, B_r)$ to randomly shuffle the ordering of the transactions in the committed block B_r, resulting in a permuted block B'_r. The permuted block is then added on-chain for execution.

5.2 Users Generating the Seed

In this section, we build our $\mathsf{BlindPerm}$ protocol following $\mathsf{Disperse(tx\text{-}key)}$ and $\mathsf{Retrieve(tx\text{-}key)}$. Our main observation here is to generate the permutation seed as a function of the keys $\mathsf{tx\text{-}key}$ corresponding to the encrypted transactions in the committed block, *e.g.*, XOR of all. This allows computing the seed essentially *at no cost* as the validators no longer compute any threshold signature. More precisely, the users just need to share the random symmetric key $\mathsf{tx\text{-}key}$ they already used for the symmetric encryption as their contributions towards randomness. As a result, the randomness is uniformly distributed with no DKG needed. To implement $\mathsf{Disperse()/Retrieve()}$, we deploy PVSS for concreteness. However, we remark all the other options already presented in Section 3.1, including the fast secret sharing with post-verification, can also be used.

Step 1 – Submission. Each user picks a key $\mathsf{tx\text{-}key}$ to encrypt a transaction tx and broadcasts it to the network of $n = 3t + 1$ validators. Moreover, the user runs $\mathsf{PVSS.Share}(\mathsf{tx\text{-}key}, \{pk_i\}_{i\in[n]})$ and broadcasts the encrypted shares $\{\hat{s}_i\}_{i\in[n]}$ and proof π_s to the validators.

Step 2 – Committing to the total ordering. The protocol operates in views. Let r be the current view number where a designated validator acts as the leader to propose a block B_r. The block contains a list of encrypted transactions in the mempool whose sharing has been completed at $n - t$ validators (*i.e.*, shares are held by at least $t + 1$ honest validators). Upon receiving $2t + 1$ votes, the proposed block B_r becomes committed by all the honest validators in the system. The protocol enters the next view upon committing the proposed block and having $t + 1$ valid shares for each committed transaction, or $2t + 1$ complaints against the leader.

Step 3 – Decryption and deriving the seed. When a validator observes the block B_r has become committed, it produces a decrypted share $\mathsf{PVTSS.Decshare}(\hat{s}_i, sk_i)$ for each committed transaction tx and sends it to others. Upon gathering $t + 1$ valid decrypted shares, each validator obtains $\mathsf{tx\text{-}key}$ using Lagrange interpolation and decrypts tx. Let $\mathsf{tx\text{-}key_1}, \ldots, \mathsf{tx\text{-}key_k}$ be the set of keys corresponding to the valid transactions in the committed block B_r. Each validator computes the permutation seed as $\mathsf{seed_r} = \mathsf{tx\text{-}key_1} \oplus \cdots \oplus \mathsf{tx\text{-}key_k}$.

Step 4 – Execution. After computing the seed, each validator locally performs $\mathsf{Permute}(\mathsf{seed}_r, B_r)$ to randomly shuffle the ordering of the transactions in the committed block B_r, resulting in a permuted block B'_r. The permuted block is then added on-chain for execution.

5.3 Analysis

Lemma 2. *The proposed protocols satisfies safety, liveness and a secure permutation.*

Proof. The safety and the liveness of the protocols directly follow those of underlying consensus. When users generate the seed, they secret share a random key tx-key to the validators. The key is recovered only after committing the block by the validators, guaranteeing the security of the permutation. Moreover, the existence of just one non-colluding user (with validators) enables the seed to be uniformly at random. □

6 Optimizations

6.1 Bandwidth

Selective Encryption. Several works in the literature separate the issue of transaction censorship from the common types of MEV that suffer user experience [41, 28, 41, 49, 48]. Following this threat we can introduce efficiency in our BlindPerm constructions, particularly the one with users' contributions towards the seed (Section 5.2), by having only those users owning an MEV-potential transaction encrypt and let others send their transactions in plaintext. This stems from doing the shuffling after the commit, providing protection against possible front-running, back-running, and sandwich attack against any encrypted transactions. Observe that this does not affect the security of the seed derived for the following reason. In order for the attacker to make a profit from a victim's transaction tx (which we assume is encrypted) via the aforementioned strategies, they need to ensure it is indeed *included* in the committed block. This consequently guarantees that the corresponding key tx-key will be considered in the computation of the seed seed, guaranteeing uniform randomness. In fact, even if the validator only includes one encrypted transaction (*i.e.*, victim's transaction) in the block it is sufficient to ensure the security of the permutation. However, one caveat arises when there is no encrypted transaction included in the committed block. It basically means there is no MEV-potential transaction in the block and thus there is no permutation seed, paving the way for the block proposer to insert their transaction at their desired spot (refer to Section 4).

Timelock Encryption. The concept of timelock encryption or timed encryption [43] allows encrypting a message that is decryptable only after passing some determined time. In other words, it features "encrypting to the future". To ensure a guaranteed delay, traditional schemes rely on sequential computation that is unparallizable. Recently, Gaily et al. [25] presented a construction that offers the same functionality without requiring any sequential computation. In fact, it relies on an existing committee (*i.e.*, threshold network) that produces BLS signatures on time intervals (*i.e.*, discrete view numbers). With the use of an identity-based encryption scheme [7], anyone can encrypt a message to future under the view number as the identity that can be decrypted only after the release of the corresponding threshold signature as the private key. Given that we already have such threshold network producing BLS signature in our BlindPerm construction thanks to the validators (Section 5.1), one may leverage it to enable users encrypt their transactions tx to any future view number of their choice. This can be thought of as an on-chain commit-reveal realization for the encrypted mempool [28]. Unlike existing solutions such as [32, 21, 18], this approach does

not incure latency or computational overhead to the system. Also, this can pose a considerable boost in communication overhead compared to the typical threshold cryptography paradigm, as the permutation seed and the decryption key for a given view number is only a single BLS signature. By separating the role of validators from threshold network the privacy of transactions last even against a *dishonest majority* of colluding validators. However, an immediate issue that arises with a naive implementation is the possibility of decrypting a transaction tx at view r without having it included in the committed block by the leader, making it vulnerable to MEV extraction afterwards. Fixing this issue without sacrificing the efficiency could be an interesting research question.

Communication-efficient PVSS. SCRAPE [14] is a state-of-the-art PVSS protocol with the following sharing procedure $\mathsf{PVSS.Share}$. The dealer samples a uniform value $s \xleftarrow{\$} \mathbb{Z}_q$, sets the secret as a group element of form $S = h^s$, splits s into shares $\{s_i\}_{i\in[n]}$ using Shamir secret sharing, and computes the encrypted shares under parties' public keys $\{\hat{s}_i = {pk_i}^{s_i}\}_{i\in[n]}$. The dealer also publishes commitment to shares and $O(n)$-sized NIZK proofs π_s with individual shares as their witnesses, enabling anyone to check the correctness of sharing with a linear cost.

The recent work of [15] introduces efficiency optimizations over SCRAPE to reduce its communication and computation complexities. In particular, the dealer needs to send just $O(1)$-sized proof of correctness with no public commitments, making the overhead close to optimum [15]. They managed to achieve these efficiencies thanks to making two modifications in the usual model of PVSS, including assigning key pairs to the dealer and doing secret sharing in a group. Fortunately, we can use such PVSS in our $\mathsf{BlindPerm}$ construction by accommodating both modifications as the users are equipped with such key-pairs[8] and symmetric key $\mathsf{tx\text{-}key}$ to share could be a random group element. We now briefly discuss the high-level idea behind the PVSS of [15]. The authors initially observe that it is possible to check the correctness of sharing in SCRAPE PVSS without the involvement of the shareholders' key-pairs. To do so, each encrypted share should be of form $\hat{S}_i = S_i \cdot {pk_i}^{sk_D}$, establishing a shared Diffie-Hellman key between the dealer and each shareholder to communicate the share. This then turns out to be useful in allowing the dealer to produce one NIZK proof with its secret key sk_D being the witness (instead of individual shares as in SCRAPE) to ensure the correctness of sharing as a whole. We refer the reader to [15] for more details.

6.2 Latency

In all proposed constructions, the latency for commit is the same as for the underlying consensus protocol. One more latency is required for producing the seed and decrypting transactions where $t + 1$ valid contributions of each kind enable any validator to prepare the block for execution. It is possible to reduce the latency by having each validator include their partial signature (for seed) and decryption share (for each transaction) as part of their vote for the block. An honest validator then counts the vote if all the contributions are valid. So, upon committing a block the validators can compute the seed and also decrypt all the transactions. We remark that this process should be done carefully to ensure the confidentiality of the seed or transactions before block gets committed. The commit phase usually takes two or three rounds in the BFT SMR. Initiating the seed generation or key decryption in an early round may put the leader in a position to learn the secret before the block gets committed. A possible sidestep would be having validators start revealing the shares (together with the votes on proposal) after they become locked on the proposal. For instance, HotStuff [54] commits B_r when there exist three consecutive quorum certificates in the chain, QC_{r1}, QC_{r2}, QC_{r3}. The validators should start revealing the shares no sooner than getting the second QC_{r2} and becoming locked on the proposal. In such safety-offering protocols, the worst a byzantine

[8]Such key-pairs are nevertheless needed, either ephemeral (for wallets) or registered (for authentication).

leader can do is to delay the process or abort. Given that at least $t+1$ honest parties are locked on the proposal, they will not vote for incompatible blocks while the lock is in place. To put another way, since at least $t+1$ honest parties are required to produce any QC, this makes it impossible for any block incompatible with B_r to receive a QC in subsequent views.

7 Discussion

Extension to Longest-chain. The principle of applying a permutation on the list of transactions in a committed block before updating the state is not limited to the BFT style consensus and can also be deployed in the longest-chain setting. However, figuring out how to provide a random seed for shuffling is a challenging task without relying on threshold security. In a proof-of-work (PoW) blockchain, the miner needs to find a solution (*i.e.*, nonce) to a puzzle to be eligible as the block proposer. Our idea is to use this nonce together with the Merkle root of the transactions (and possibly some auxiliary data to increase the entropy) as the seed for the permutation. So, the state change occurs with regard to the permuted block. Should a miner decide to modify the ordering of transactions in the block after learning the seed, they face the threat of losing it due to the difficulty rule of the puzzle. Security-wise, this method enhances the recent efforts in leveraging trusted execution environments (TEEs), such as SGX [6], to provide privacy for transactions up to a point where their inclusion in the block is ensured. TEEs can be thought of as a replacement for committee to generate a key-pair, with the public key being used for encryption. The process is as follows. The encrypted transactions are received by the SGX and get decrypted and ordered to form a block. The miner receives the block header computed by the SGX to start solving the puzzle and obtains the body of transactions upon providing the SGX with a valid nonce. Afterward, any change in the ordering of transactions in the block demands solving a new puzzle.

The above procedure cannot be directly translated to proof-of-stake (PoS) blockchain, like Ethereum, as the authentication is no longer based on a PoW puzzle. Given that a block proposer is chosen (pseudo) randomly, one may think of utilizing such high entropy randomness as the seed for the permutation. An astute reader can realize a crucial problem with this approach. As the leader already learns the seed and it is independent of the block content, they may neutralize the effect of permutation accordingly. Inclusion of the block content (*i.e.*, its Merkle root) in the leader election similar to PoW-based blockchain does not work as there is no computational difficulty involved and the leader can work through many candidates to find the best fit. We leave this as an interesting open problem. Moreover, one issue with the longest-chain setting is the lack of guaranteed finality compared to that of BFT, allowing a powerful adversary to revert the chain in an attempt to change the order of transactions in a block.

Pre-ordered Bundles. Private ordering approaches refer to the process of sending the transactions by the user directly to some trusted services known as relays, like Flashbot [1], rather than publicly broadcasting them to the peer-to-peer network. These services then sequence transactions in bundles and forward them to the validators. Although these approaches can provide MEV protection for users by hiding their transactions from public, their primary goal is to facilitate the MEV extraction by anyone who wishes. That is, the searcher can simply use such services to send a *pre-ordered* bundle of transactions (including its own and that of the victim) to relays without taking the risk of sending over a peer-to-peer network which is quite risky. It is clear that shuffling the order of transactions can be an effective method to get rid of such bundles. On the other hand, treating them as an object (*i.e.*, atomic unit) while shuffling allows appreciating the interior ordering if needed. This might protect the searcher against a front-running attack by the validator [2]. Note that the notion of proposer/builder separation [12] is now wildly adopted on the Ethereum blockchain and

aims at reducing the trust on relays by decoupling the role of creating and proposing the block by validators.

8 Related Work

After introducing the MEV problem by Daian et al. [19], a great deal of effort has appeared in the literature to propose countermeasures in various flavors. The authors in [53, 28] categorize these efforts and we here only highlight some of the relevant works that aim at providing solutions at the consensus level. The work of [31] introduces Aequitas protocols that order a transaction tx no later than tx' if some fraction γ of parties receive tx before tx', known as γ-batch order fairness. The protocol abstractly operates in three stages, requiring all transactions to go through them before finally getting delivered. This includes (1) Gossip, broadcasting transactions to all parties as they are received; (2) Agreement, agreeing on a set of local logs of transactions for ordering; (3) Finalization, locally producing the output based on the agreed set of local orderings. Apart from necessitating a relaxed definition of order fairness, it turns out Condorcet cycles may become larger arbitrarily and also negatively affect the liveness of [31], motivating the design of a follow-up protocol called Themis [30] with a similar spirit. In this work, the authors attempt to address the two challenges of weak-liveness (due to arbitrary long chained cycles), and cubic communication complexity (due to the all-to-all communication). To handle the former, they make the time required for ensuring liveness dependent on the network delay Δ and not the completeness of cycles. To handle the latter, they deploy the folklore method of all-to-one/one-to-all communication pattern to reduce the communication overhead from cubic to quadratic.[9] Cachin et al. [13] revisits the notion of order fairness by changing the relative measure of batch order fairness to *differential order fairness*, taking into account the difference between the number of correct parties that receive a tx before tx' compared to that of vice versa. They argue about the usefulness of such modification to tolerate higher fault-tolerance compared to that of batch order fairness [31, 30] with a reasonable value for parameter γ, where in their treatment only counts the honest parties. The work of [34] presents several protocols that can be added to the underlying blockchain protocol as a fairness toolkit. To get a better liveness guarantee, it also proposes timed relative fairness as a weaker fairness definition that requires parties to maintain a local clock to decide on the ordering of the incoming transactions according to some point in time.

The requirement for maintaining causality in SMR systems was first put forth by [42]. They showed the importance of preserving the casual order of users/clients' requests and proposed adding a confidentiality layer to the underlying atomic broadcast (*i.e.*, SMR) to establish a secure causal atomic broadcast [22]. The recent efforts in literature for blind-order fairness are essentially an extension of this approach, realizing the confidentiality layer with a range of new cryptographic tools and techniques. In [55], validators just carry out the consensus to commit a block of encrypted transactions where a separate secret-management committee runs the decryption per transaction. Such separation could provide optimum fault tolerance of $t < n/2$ for the committee. Fino [36] integrates the blind-order fairness into DAG-based BFT systems that allow parallel dissemination of proposals by multiple validators, achieving high throughput [20, 47]. The proposed blind-order fairness has a hybrid structure, where the key for decryption is either obtained via a fast path using secret-sharing with post-verification or a slow path using threshold decryption. The authors in [37] develop a blind-order fairness system with minimal communication overhead, allowing users to encrypt their transactions to some future time (*i.e.*, view number) with the corresponding private key being released by a committee then. FairPoS [17] introduces a similar notion to blind-order

[9] The communication complexity can be further reduced to (asymptotically) linear by making use of SNARKs.

fairness for a longest-chain style consensus called input fairness. They rely on time-based cryptography [11] to hide the content of transactions under a single unknown key until block finalization, which consequently leads to achieving adaptive security. This is implied by the non-parallelizable sequential computation needed for decryption, preventing the leakage of sensitive information (*i.e.*, key material) upon corrupting an honest party. Note that our proposed optimization using timelock encryption share the same rationale with [37, 17] in the sense that a single key (*i.e.*, BLS signature) is enough for decrypting all the encrypted transactions in a committed block.

References

[1] Flashbots, 2022.

[2] Ethereum Bot Gets Attacked for $20M as Validator Strikes Back., 2023.

[3] A. Bacher, O. Bodini, H.-K. Hwang, and T.-H. Tsai. Generating random permutations by coin tossing: Classical algorithms, new analysis, and modern implementation. *ACM Trans. Algorithms*, 13(2):24–1, 2017.

[4] J. Bebel and D. Ojha. Ferveo: Threshold decryption for mempool privacy in bft networks. *Cryptology ePrint Archive*, 2022.

[5] M. Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.

[6] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1521–1538, 2019.

[7] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[8] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

[9] G. Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[10] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz, et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. *Chainlink Labs*, 1, 2021.

[11] J. Burdges and L. De Feo. Delay encryption. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I*, pages 302–326. Springer, 2021.

[12] V. Buterin. Proposer/block builder separation-friendly fee market designs, 2021.

[13] C. Cachin, J. Mićić, N. Steinhauer, and L. Zanolini. Quick order fairness. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 316–333. Springer, 2022.

[14] I. Cascudo and B. David. Scrape: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*, pages 537–556. Springer, 2017.

[15] I. Cascudo, B. David, L. Garms, and A. Konring. Yolo yoso: fast and simple encryption and secret sharing in the yoso model. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part I*, pages 651–680. Springer, 2023.

[16] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[17] J. H.-y. Chiang, B. David, I. Eyal, and T. Gong. Fairpos: Input fairness in proof-of-stake with adaptive security. *Cryptology ePrint Archive*, 2022.

[18] D. Cline, T. Dryja, and N. Narula. Clockwork: An exchange protocol for proofs of non front-running.

[19] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[20] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[21] Y. Doweck and I. Eyal. Multi-party timed commitments. *arXiv preprint arXiv:2005.04883*, 2020.

[22] 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)*, pages 61–72. IEEE, 2017.

[23] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[24] P. Feldman and S. Micali. Byzantine agreement in constant expected time. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 267–276. IEEE, 1985.

[25] N. Gailly, K. Melissaris, and Y. Romailler. tlock: practical timelock encryption from threshold bls. *Cryptology ePrint Archive*, 2023.

[26] W. V. Gehrlein. Condorcet's paradox. *Theory and Decision*, 15(2):161–197, 1983.

[27] T. Hanke, M. Movahedi, and D. Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[28] L. Heimbach and R. Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. In *4th ACM Conference on Advances in Financial Technologies (AFT)*, 2022.

[29] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

[30] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *Cryptology ePrint Archive*, 2021.

[31] 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*, pages 451–480. Springer, 2020.

[32] R. Khalil, A. Gervais, and G. Felley. Tex-a securely scalable trustless exchange. *Cryptology ePrint Archive*, 2019.

[33] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[34] 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*, pages 25–36, 2020.

[35] L. LAMPORT, R. SHOSTAK, and M. PEASE. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[36] D. Malkhi and P. Szalachowski. Maximal extractable value (mev) protection on a dag. *arXiv preprint arXiv:2208.00940*, 2022.

[37] P. Momeni, S. Gorbunov, and B. Zhang. Fairblock: Preventing blockchain front-running with minimal overheads. In *International Conference on Security and Privacy in Communication Systems*, pages 250–271. Springer, 2022.

[38] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[39] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[40] T. P. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, pages 522–526. Springer, 1991.

[41] 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)*, pages 198–214. IEEE, 2022.

[42] M. K. Reiter and K. P. Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):986–1009, 1994.

[43] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. 1996.

[44] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[45] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*, pages 148–164. Springer, 1999.

[46] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[47] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[48] A. Wahrstätter, J. Ernstberger, A. Yaish, L. Zhou, K. Qin, T. Tsuchiya, S. Steinhorst, D. Svetinovic, N. Christin, M. Barczentewicz, et al. Blockchain censorship. *arXiv preprint arXiv:2305.18545*, 2023.

[49] Y. Wang, P. Zuest, Y. Yao, Z. Lu, and R. Wattenhofer. Impact and user perception of sandwich attacks in the defi ecosystem. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2022.

[50] B. Weintraub, C. F. Torres, C. Nita-Rotaru, and R. State. A flash (bot) in the pan: measuring maximal extractable value in private pools. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 458–471, 2022.

[51] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[52] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse. {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, 2022.

[53] 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.

[54] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[55] H. Zhang, L.-H. Merino, V. Estrada-Galinanes, and B. Ford. Flash freezing flash boys: Countering blockchain front-running. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 90–95. IEEE, 2022.