BlindPerm: Efficient MEV Mitigation with an Encrypted Mempool and Permutation

Alireza Kavousi[1], Duc V. Le[2], Philipp Jovanovic[1], and George Danezis[1,3]

[1] University College London
[2] Visa Research
[3] Mysten Labs

Abstract. To mitigate the negative effects of the maximal extractable value (MEV), we propose techniques that utilize randomized permutation to shuffle the order of transactions in a committed block before execution. We argue that existing approaches based on encrypted mempools cannot provide sufficient mitigation, particularly against block producer, and can be extended by permutation-based techniques to provide multi-layer protection. With a focus on PoS committee-based consensus we then introduce BlindPerm, a framework enhancing an encrypted mempool with permutation and present various optimizations. Notably, we propose a protocol where this enhancement comes at essentially no overheads by piggybacking on the encrypted mempool. Further, we demonstrate how to extend our mitigation technique to support PoW longest-chain consensus. Finally, we illustrate the effectiveness of our solutions on arbitrage and sandwich attacks as the two main types of MEV extraction through running simulations using historical Ethereum data.

1 Introduction

Blockchain and in particular cryptocurrencies initially emerged with a focus on presenting a robust financial ecosystem, but the lack of attention to the issue of *ordering manipulation* turned out to be problematic. It became more serious with the introduction of decentralized finance (DeFi) and in particular decentralized exchange (DEX). The execution mechanism behind these platforms called smart contract has some level of transaction ordering dependence [30] that allows ordering manipulation to cause a major impact on the traded crypto asset by the actors [17]. So, a block producer can take advantage of such opportunities to achieve some benefit beyond the regular transaction fee and block reward. An illustrative scenario on a 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. The profits made via including, excluding, or re-ordering transactions within blocks are described as maximal extractable value (MEV) [17]. Not only the block producer (*i.e.*, miner, validator) can readily extract MEV due to their centralized role in preparing the block, but also any actor in the system known as *searcher* [1] can do so by observing

the state of the system and the knowledge of profitable transactions. This mostly 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 transactions relative to that of the victim; otherwise, it may not be profitable or even lead to a loss for the attacker. Adoption of services like Flashbots [1] makes the MEV extraction even easier due to the ability to offer front-running as a service [38].

Recent years have seen substantial efforts to combat the negative effects of MEV. Arguably the most notable ones are so-called *order fairness* protocols that consider providing a notion of fairness for the transactions that appear in a finalized block. This notion, however, is not universally agreed upon and may have various interpretations. Timed ordering [12,28,29] and blind ordering [15,33, 50] are two well-known notions in the literature that aim at providing mitigation at the consensus layer. In essence, the former fallows a *first come first serve* approach and 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.

Given that the existing proposals following timed ordering have expensive configuration costs and usually demand low fault tolerance and high latency due to enforcing extra constraints on consensus layer, realizing blind ordering in various settings has received plenty of attention [27]. However, there are two principal drawbacks to this approach that might severely affect its usefulness. First, the block producer can easily frontrun other transactions regardless of their contents, *e.g.*, upon observing an arbitrage opportunity triggered by the latest block [38]. Consider a scenario where a popular non-fungible token (NFT) is dropped and the block producer decides to buy some NFT. They can place their transaction in an early spot, front-running others and buying at a lower price. 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 [50].

The main intuition behind blind ordering is to make the ordering of transactions independent of their contents. We observe that a random permutation on the committed block also renders the final ordering independent of the committed one. So, 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. This technique is a useful strategy in mitigating the power of the block producer in imposing their desired ordering, that is shown to capture the majority of total MEV [34]. Deploying permutation is also helpful to protect against those types of MEV that negatively affect users, particularly sandwich attacks [2]. This is due to the fact that a random permutation can either make the attack non-profitable or 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.

Random ordering has recently received attention at an abstract level, analyzing it as an ex-ante fairness where the profit a validator obtains via MEV

extraction is measured relative to a random ordering as a base [3]. In this work, we show how to realize such random ordering efficiently using cryptographic techniques. We take a step further and argue about the importance of a combined solution, where blind ordering and permutation enhance each other in a complementary manner. More precisely, an encrypted mempool that is the core of blind ordering mitigates content-related attacks such as *censoring* or *congestion* and *spamming*, where an attacker may fill out the mempool in the hope of front-running/back-running a profitable transaction and thus diminishing the effectiveness of the permutation. Also, permutation on a committed block strengthens the effectiveness of an encrypted mempool by mitigating the role of metadata leakage and the advantage of the block producer to take the (top of the block) MEV solely.

Contributions. The contributions of this work are as follows.

- We introduce randomized permutation as a mechanism for MEV mitigation at the execution layer and propose an efficient construction for BFT style consensus. This technique reduces the power of block producer in imposing their desired ordering and, in particular, hampers sandwich attacks.
- We introduce BlindPerm, a framework that combines an encrypted mempool with permutation to provide enhanced MEV mitigation. Particularly noteworthy is that our permutation-based enhancement is *generic* and can come at *no additional cost* in comparison to regular blind ordering without shuffling.
- We present several optimizations 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. Also, we show how to extend the permutation-based approach to proof-of-work (PoW) longest-chain protocols.
- We demonstrate the effectiveness of our methods via simulating MEV extraction using the Ethereum historical data. We deploy the heuristics proposed in the literature [38] to detect arbitrage and sandwich attacks in the blocks over a one-year period and compare with the existing opportunities after applying our methods.

2 Background

2.1 Threat Model

We consider the setting of a Byzantine fault tolerance (BFT) system. In this setting, there are n parties at most f 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. The network model is partially synchronous [20], 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. We consider the optimal resilience of $f \geq 3f + 1$ in this setting [6].

2.2 Secret Sharing

A (t, n) Shamir secret sharing [43] 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 [43] is passively secure, meaning that it works as long as the participating parties run the protocol as specified. In a Byzantine setting parties might be malicious, so the dealer needs to convince parties about the correctness of the sharing and parties need to convince 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 [21], there have been numerous works 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 to enable share verification and having a complaint phase to deal with any faulty/missing share.

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 sharing phase by the dealer and the reconstruction phase by the set of shareholders. A popular PVSS scheme is SCRAPE [13] 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 $\mathsf{PVTSS.Combine}(\tilde{s}_1, \ldots, \tilde{s}_{t+1})$.

2.3 Threshold Cryptography

Distributed Key Generation (DKG). A DKG protocol [36] shares a uniformly distributed secret sk among n parties such that each party receives a partial secret key sk_i, a partial public key pk_i, and a common public key pk while no individual party learns sk. DKGs are commonly used as a trustless setup for threshold encryption and threshold signature schemes.

Threshold Encryption. In a (t, n) threshold encryption scheme TE, 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 $t+1$ threshold 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 TS, 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 $\mathsf{TS.Parverify}(pk_i, ps_i, m)$, and the signature via $\mathsf{TS.Verify}(pk, \sigma, m)$.

2.4 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 f of them. The two core properties of a consensus protocol are *safety* and *liveness*. The former ensures all the honest parties decide on the same (valid) value and the latter ensures an honest party eventually decides. Byzantine broadcast is an important variant of consensus that enables a sender to send its input to the other parties such that all the honest ones decide on the same value.

Committee-based Consensus. Among different formulations of consensus, state machine replication (SMR) [42] enables agreement on *ever-growing* inputs received from external users, making it a suitable option for blockchain. Committee-based consensus protocols have become increasingly popular in recent years due to their high throughput and low latency [18, 32, 44]. In these protocols, the committee runs a BFT style consensus where 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 driven by a leader (*i.e.*, block producer). Although the interest in committees-based blockchains has led to considerable innovations in the literature, the following protocol flow is a common paradigm for SMRs.[4] First, the leader prepares a block of transactions and sends the proposal to all the other parties. Second, each party votes on the proposal if it is properly formed and sends the vote back to the leader. Third, upon collecting $n-f$ votes the leader creates a quorum certificate (QC) and disseminates it to the parties. This process repeats more than once in each view of the protocol to commit. As a concrete instantiation, we deploy the recent work of HotStuff-2 [32] that is an improvement over the original HotStuff [49] following a two-phase commit process per view.

2.5 Memory Pool

In the context of blockchain, a memory pool or mempool refers to where the uncommitted transactions (i.e., pending transactions) exist. The notion of *encrypted mempool* is wildly known as a countermeasure against MEV, addressing

[4] DAG-based BFT systems allow parallel dissemination of proposals by multiple validators [18, 44].

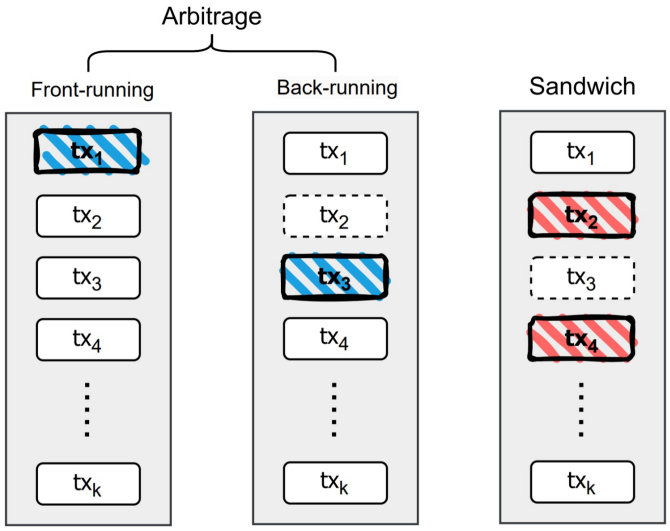

Fig. 1: A visual representation of the common MEV extraction strategies. Dashed boxes indicate the (potential) victim transactions which trigger the opportunities in a block.

the information asymmetry in the blockchain state between the user and the validator via providing privacy for transactions before they are committed [41]. Among different methods in realizing an encrypted mempool, the ones with threshold security are currently offering the most desirable qualities [27]. Low latency, wide coverage, and reasonable performance are the main features that have placed this approach in a promising position.

3 MEV Mitigation with Permutation

A block producer may take advantage of their full control over transaction ordering, which can lead to unfair advantages and unintended profits. For instance, they can simply place their transaction at the top of the block to benefit from a price slippage caused by some large trade on an exchange [26]. When looking closely, however, even leveraging a *perfect* encrypted mempool, *i.e.*, without any metadata leakage, cannot provide enough protection. This is due to the fact that such action does not necessarily rely on the content of existing transactions in the mempool but on the latest state of the blockchain. Figure 1 depicts the common MEV extraction strategies with the highest profit in practice [38].

A random permutation on the set of transactions in a committed block takes away such forced ordering. Moreover, it hampers sandwich attacks by raising the risk of loss for the attacker. A concrete scenario would be ordering the attacker's back-run before its front-run, causing a loss to the attacker. In another scenario, the attacker's front-run might execute but another searcher ends up

making a profit by having its back-run execute prior to that of the attacker. The permutation should occur on the committed block so that the attacker cannot nullify the outcome in case it happens not to be in its favor. It is also crucial to do it *safely* and in a *secure* way. To ensure the former, the honest validators must perform the same permutation on the same block. To ensure the latter, the randomness seeding the permutation should be unpredictable prior to the commit, and unbiasable by an adversary controlling the block producer and possibly a subset of validators. In the following section, we propose a protocol that satisfies these properties.

3.1 Protocol Description

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

Step 1 – Submission. Each user broadcasts their transaction tx to the network of $n \geq 3f + 1$ 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 set of transactions in the mempool. Upon receiving the second QC within the view, the proposed block B_r becomes committed by each honest validator.

Step 3 – Deriving the seed. To generate secure shared randomness we assume validators have already run a DKG protocol.[5] The randomness is securely derived by having validators jointly produce a (t, n) threshold signature on the view number using a unique threshold signature scheme like BLS [9], where $t = 2f$. Thus, when a validator observes that the block B_r has become committed, they send a partial signature of the form $H(r)^{sk_i}$ by invoking $\mathsf{TS.Parsign}(sk_i, r)$ to others. Let I be the set of indices of $2f + 1$ valid partial signatures. Anyone can then 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. A validator enters the next view after deriving the seed.

Step 4 – Execution. Upon computing the seed,[6] each validator locally performs $\mathsf{Permute}(\mathsf{seed_r}, B_r)$ to randomly shuffle the ordering of transactions in B_r, resulting in a permuted block B'_r. The permuted block is then executed. A standard permutation algorithm is given in Appendix A.

[5] This is a common assumption made in the state-of-the-art BFT consensus including HotStuff-2 [32].

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

3.2 Analysis

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

Proof. Safety and the liveness of the protocol follow directly from the underlying consensus as we treat it in a black-box manner. Thanks to the finality of the BFT consensus and the uniqueness of the threshold BLS signature, the honest validators apply the same permutation on a committed block they have already agreed on. The seed is pseudorandom and unpredictable to the validator proposing the block before it gets committed, guaranteeing a secure permutation.

4 BlindPerm

A broad scope of MEV comes from the availability of information about transactions, either those that are already submitted on the public mempool or the ones observed early by a capable searcher [10]. Such information could directly affect the users by facilitating the MEV for the validator or searcher through creating dependent transactions or even censoring an undesirable transaction. This essentially leads to reducing the effectiveness of the permutation as the chances of the victim's transaction getting front-runned/back-runned nevertheless increases.[7] Since a sole permutation-based solution cannot offer suitable protection in these situations, we propose BlindPerm, a framework that enhances an encrypted mempool with permutation. 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 potential negative effects, offering the best of both worlds. From the attacker's perspective, transaction permutation shifts sandwich attacks from being riskless to being risky, and transaction blinding shifts censorship from being optional to being all-or-nothing.

Our framework includes two categories depending on the way the permutation seed is generated. The first relies on the validators and the second relies on the users to contribute to the seed. Interestingly, the second approach allows obtaining the random seed essentially *for free* by piggybacking on the encrypted mempool. 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 [33], or the user being able to affect the ordering according to its view of the system [5].

4.1 Seed Contribution by Validators

In this section, we extend the protocol proposed in Section 3.1 to establish an encrypted mempool. We use threshold cryptography to let users encrypt their transactions and validators compute the seed.

[7] Any relative ordering of 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 the validators' common public key $\mathsf{TE.Enc}(pk, \mathsf{tx})$ and broadcasts the ciphertext c to the network of $n \geq 3f + 1$ 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 set of encrypted transactions in the mempool. Upon receiving the second QC within the view, the proposed block B_r becomes committed by each honest validator.

Step 3 – Decryption and deriving the seed. When a validator observes that the block B_r has been committed, they produce a decryption share $\mathsf{TE.Pardec}(sk_i, c)$ for each committed tx and a partial signature $\mathsf{TS.Parsign}(sk_i, r)$ as their contribution towards the seed.[8] They then send the partial decryptions together with partial signatures to others. Each validator can obtain the transaction tx and the seed $\mathsf{seed_r}$ by running $\mathsf{TE.Dec}$ and $\mathsf{TS.Sign}$ upon receiving $2f+1$ valid partial contributions and enter the next view afterwards.

Step 4 – Execution. Each validator locally performs $\mathsf{Permute}(\mathsf{seed_r}, B_r)$ to randomly shuffle the ordering of transactions in the committed block B_r, resulting in a permuted block B'_r which is then executed.

4.2 Seed Contribution by Users

We now build our $\mathsf{BlindPerm}$ protocol with each user choosing a random symmetric-key $\mathsf{tx\text{-}key}$ to encrypt a transaction and secret share the key to the validators. 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 produce any threshold signature and use the keys they already retrieved for decryption. As a result, the randomness is uniformly distributed. To implement the secret sharing, we deploy PVSS for concreteness. However, we stress that our protocol is *agnostic* to the type of secret sharing scheme and all the other options presented later in Appendix C could 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 $n \geq 3f+1$ 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 set of encrypted transactions in the mempool whose sharing has been completed at the validators. Upon receiving

[8] For ease of notation, we use the same key-pairs for both threshold encryption and signature. However, they could be different.

the second QC within the view, the proposed block B_r becomes committed by each honest validator.

Step 3 – Decryption and deriving the seed. When a validator observes the block B_r has been committed, they produce a decrypted share $\mathsf{PVTSS.Decshare}(\hat{s}_i, sk_i)$ for each committed transaction tx and send it to others. Upon gathering $2f+1$ valid decrypted shares, the 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 $\mathsf{seed}_r = \mathsf{tx\text{-}key}_1 \oplus \cdots \oplus \mathsf{tx\text{-}key}_k$ and enters the next view thereafter.

Step 4 – Execution. Each validator locally performs $\mathsf{Permute}(\mathsf{seed}_r, B_r)$ to randomly shuffle the ordering of transactions in the committed block B_r, resulting in a permuted block B'_r. The permuted block is then executed.

4.3 Analysis

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

Proof. The safety of the protocols directly follows from that of the underlying consensus. Due to the robustness of the underlying threshold cryptosystems, we are guaranteed to have enough shares for decryption and to derive the seed, ensuring liveness. It is guaranteed that the transactions are revealed once they are finalized, given the equality of the consensus threshold and decryption threshold. When users contribute to the seed, they secret share a random key $\mathsf{tx\text{-}key}$ 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.

5 Optimizations

Selective Encryption. Several works in the literature separate the issue of transaction censorship from the common types of MEV that suffer user experience [27,38,46]. Following this thread we can make some bandwidth optimization in our $\mathsf{BlindPerm}$ constructions, particularly the one with users' contributions towards the seed (Section 4.2). That is, 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 permutation seed 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 implies there is no MEV-potential transaction in the block and thus there is no permutation seed, paving the way for the block producer to insert their transaction at their desired spot (refer to Section 3).

Timelock Encryption. The concept of timelock encryption or timed encryption [40] allows encrypting a message that is decryptable only after passing some determined time. In other words, it features "encrypting to the future". To provide a guaranteed delay, traditional schemes rely on sequential computation that is unparallizable. Recently, Gaily et al. [22] 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 [8], anyone can encrypt a message to the 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 signatures in our BlindPerm construction thanks to the validators (Section 4.1), one may leverage it to enable users encrypt their transactions tx to any future view number of their choice. 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 the threshold network the privacy of transactions lasts 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 block producer, making it vulnerable to MEV extraction afterwards. Very recently, Choudhuri et al. [16] proposed the notion of *batched threshold encryption* that fixes the aforementioned issue by providing pending transaction privacy to only decrypt the batch of transactions made it to block while the partial decryption key still being independent of the batch size.

Communication-efficient PVSS. SCRAPE [13] is a state-of-the-art PVSS with the following sharing procedure 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. Cascudo et al. [14] introduce 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 [14]. They managed to achieve these efficiencies

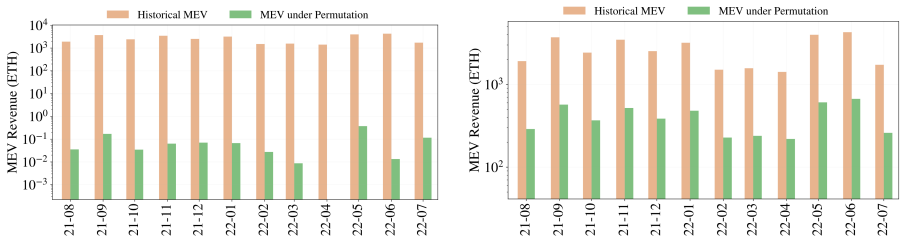

Fig. 2: Simulation results for MEV extraction due to perfect sandwich attacks (left) and imperfect sandwich attacks (right) with a random permutation on the block.

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 BlindPerm construction by accommodating both modifications as the users are equipped with such key-pairs[9] and symmetric key tx-key to share could be a random group element. We now briefly discuss the high-level idea behind the PVSS proposed in [14]. The authors initially observe that it is possible to check the correctness of sharing in SCRAPE 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 [14] for more details.

6 Simulation Results

We simulate the MEV extraction using real-world Ethereum data over a one-year period from August 2021 to July 2022.[10] We measure the historically extracted MEV according to the heuristics introduced in [38] and compare it with the expected amount of MEV extraction after applying our methods, namely permutation and BlindPerm.[11] We run our simulations 10 times to capture various relative ordering of transactions and report the average results. Note that we particularly focus on two popular strategies including arbitrage and sandwich attacks that are the most profitable [38]. For completeness, we provide the heuristics used to detect these opportunities in Appendix B.

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

[10] The data is crawled through Ethereum archive Erigon node `https://github.com/ledgerwatch/erigon`.

[11] We only perform a local simulation to measure the effectiveness of our permutation-based enhancement and do not actually broadcast the transactions. We remark that the possible side-effects like congestion is handled via encrypted mempool.

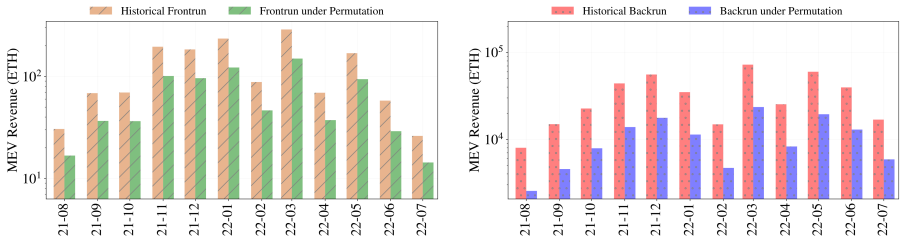

Fig. 3: Simulation results for MEV extraction due to front-running arbitrage (left) and back-running arbitrage (right) with a random permutation on the block.

Sandwich Attacks. We consider two scenarios to conduct our simulations. The first scenario follows the heuristics used to detect *perfect* sandwich attacks where front-running, victim, and backrunning transactions are ordered immediately with no intermediaries. We measure how much revenue could have been extracted if we applied a random permutation on the list of transactions in a block before execution. We find that the revenue is reduced from a total of $31,749.59$ ETH for the historical data to 1.74 ETH, showing almost a complete mitigation success as the attacks are no longer profitable.

Further, we relax the constraint and consider a (worst-case) scenario where the relative ordering of the involved transactions after permutation still holds but they are separated by multiple transactions, leading to *imperfect* profitable sandwich attacks.[12] To measure the expected amount of MEV extraction we find the indices of the front-running and back-running transactions, say i and j respectively in the historical block. If $i \leq j$ holds after the permutation, we consider win to be the event that the attack still is profitable, *i.e.*, the victim transaction appears somewhere in the middle. So, due to the uniform distribution we have $\Pr[\mathsf{win}] = \frac{|i-j|}{|\mathsf{B}|}$, where $|\mathsf{B}|$ denotes the number of transactions in the block.[13] Thus, we find that the revenue is reduced from a total of $31,749.59$ ETH for the historical data to $4,837.54$ ETH after doing the permutation, showing almost 85% of mitigation success. The numerical results are presented in Figure 2.

Arbitrage. We separately run our simulations for front-running and back-running arbitrage to give a better insight on the potential impact of a random permutation in both cases. We remark that arbitrage is not inherently harmful and is usually considered benign as it allows for market sustainability, *e.g.*, by synchronizing the price of assets across different exchanges [45]. The goal here is to reduce the sole power of the block producer (*i.e.*, sequencer or validator)

[12] It is shown that over 80% of profitable sandwich attacks are perfect and also no intermediate transaction is found in almost 99.6% of privately relayed sandwich attacks [38].

[13] Concretely, we noticed that the average number of transactions in a block is 231 with the minimum and maximum being 3 and 1386, respectively.

in MEV extraction and provide a rather fair chance for others to do so. Therefore, in Figure 3 we report the total MEV extraction (possibly by the block producer [37]) with and without doing permutation. We proceed with a brief description of our analysis that leads to the reported results.

- Front-running: A front-running arbitrage transaction should be profitable if it gets executed at the top of the block (*i.e.*, the state change is due to the previous block). So, we find the index of the (front-running) arbitrage transaction in the historical block i and then check if its index after the permutation is lower than or equal.[14] If not, the arbitrage transaction could still be profitable with a weight of at most $\Pr[\mathsf{win}] = 1 - \frac{|i-j|}{|\mathsf{B}|}$, where j is the index of the original arbitrage transaction after the permutation. Note that we consider a (worst-case) scenario where there is only one competitive arbitrage transaction in the block. However, given that arbitrage can be safely performed by anyone observing the latest state [45], the probability of win event may be considerably lower in practice. We find the revenue is reduced from a total of $1,483.18$ ETH for the historical data to 780.74 ETH after doing the permutation, showing almost 50% of mitigation success. This actually is in line with our expectation that the chance of a sole MEV extraction reduces by half.
- Back-running: Using the heuristics we are only able to find the (back-running) arbitrage transaction and do not infer the relative position of the arbitrage transaction with the victim/opportunity transaction in the historical block. So, we rely on a common assumption that the (back-running) arbitrage transaction is ordered immediately after the victim transaction. Let i, and j be the indices of the victim and arbitrage transactions in the original block. If $i \leq j$ holds after the permutation, the arbitrage transaction could still be profitable with a weight of at most $\Pr[\mathsf{win}] = 1 - \frac{|i-j|}{|\mathsf{B}|}$. We find that the revenue is reduced from a total of $410,112.07$ ETH for the historical data to $133,613.02$ ETH after doing the permutation, showing almost 67% of mitigation success.

Finally, we present the last simulation result in Figure 4 where we report the cumulative MEV revenue including arbitrage and sandwich attacks for the historical data and BlindPerm.[15] An Encrypted mempool helps with mitigating the strategies that mainly depend on the content of the (victim) transactions such as back-running arbitrage and sandwich attacks. We find that the revenue is reduced from a total of $443,866.13$ ETH for the historical data to 634.88 ETH for BlindPerm.

[14] Note that top of the block does not necessarily refer to the position with the lowest index as we detected an arbitrage transaction that is positioned at index 7 while being profitable. See `https://etherscan.io/tx/0xcfd6f698657dd2851bea4b748723ed6ffb5503641f5ffbd8122ae120f148b034`.

[15] Given that we proposed a solution where the permutation enhancement comes at no cost than building up an encrypted mempool, we refrain from a direct comparison between MEV extraction with only encryption and BlindPerm.

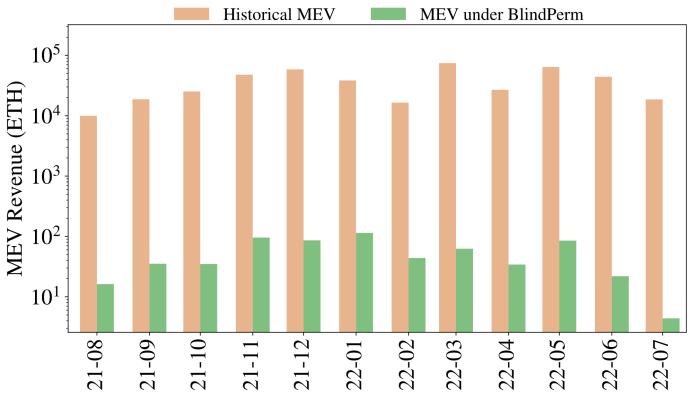

Fig. 4: Simulation results for MEV extraction due to arbitrage and sandwich attacks under BlindPerm.

7 Discussion

Extension to PoW Longest-chain. Permuting the list of transactions in a committed block before execution can also be realized in the longest-chain setting. 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 producer. Our idea is to use this nonce together with the Merkle root of the transactions (and possibly some auxiliary data) 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 loss due to the difficulty rule of the puzzle. Therefore, this method enhances the recent efforts in leveraging trusted execution environments (TEEs), such as SGX [7], 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 the committee to generate a key-pair, with the public key being used for encryption. In a recent work, Alpos et al. [2] presented a construction that utilizes permutation to prevent sandwich attacks in the (PoW) longest-chain setting. At a high level, a set of previous leaders contribute towards permutation seed using a commit-reveal mechanism. They use slashing techniques to protect against biasing and splitting the transactions into chunks to increase the permutation space and thus protect against a possible collusion of leaders. Consequently, the system incurs a considerable latency and is also limited in applicability.

Limitations. In this paper, we proposed the use of shuffling as a mitigation technique at the execution layer against MEV, which is a topic of concern within the Web3 community. Although shuffling is a powerful technique, a block producer may try to get around it. In an extreme scenario, they can just put one transaction in the block to surely capture an opportunity, say a front-running arbitrage. They can also reduce the effect of permutation by congestion, *e.g.*,

including too many of their own transactions in the block. We argue that they should give up on a large portion of the transaction fees in both cases. It is interesting to see how to sidestep such an ability, with an intuitive solution being to make the block preparation decentralized [48]. Our constructions rely on threshold cryptosystems that might be a hurdle to implement in settings with a large population of parties and dynamic participation. Recent attempts in the literature [23, 24] proposed constructions for realizing threshold cryptosystems without an interactive setup phase (known as silent setup) that could provide flexibility for supporting dynamic fault-tolerance and participation.

8 Related Work

After introducing the MEV problem in [17], a great body of research has been conducted to propose countermeasures in various flavors. Here we only focus on the solutions at the consensus layer using timed ordering and blind ordering.

Timed Ordering. The consensus problem 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. It was shown by two concurrent works of [29, 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 tx', if some majority of nodes receive the former sooner than the latter, tx should be ordered before tx'. The impossibility result is due to the so-called Condorcet's cycle/paradox [25], preventing parties to agree on a fair ordering of transactions even when all behave honestly [29].[16] The impossibility result necessitates the adoption of other variants of timed-order fairness. Kelkar et al. [29] relaxed their definition to capture *batch order fairness* by making "before" to "no later", treating such transactions in batches with relative ordering. 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. Kelkar et al. [29] introduced 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. 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 [29], motivating the design of a follow-up protocol called Themis [28] with a similar spirit. Cachin et al. [12] 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

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

to that of batch order fairness [28, 29] with a reasonable value for parameter γ, where in their treatment only counts the honest parties. The work of [31] 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 incoming transactions according to some point in time.

Blind Ordering. The requirement for maintaining causality in SMR systems was first put forth by [39]. 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 [19]. 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 [50], 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 [33] integrates the blind ordering into DAG-based BFT systems that allow parallel dissemination of proposals by multiple validators, achieving high throughput [18, 44]. The proposed blind ordering 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 [35] develop a 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 [15] introduces a similar notion to blind-order 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 [15, 35] 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. Alpos, O., Amores-Sesar, I., Cachin, C., Yeo, M.: Eating sandwiches: Modular and lightweight elimination of transaction reordering attacks. In: 27th International Conference on Principles of Distributed Systems (2024)
3. Angeris, G., Chitra, T., Diamonds, T., Kulkarni, K.: The specter (and spectra) of miner extractable value. arXiv preprint arXiv:2310.07865 (2023)
4. Bacher, A., Bodini, O., Hwang, H.K., Tsai, T.H.: Generating random permutations by coin tossing: Classical algorithms, new analysis, and modern implementation. ACM Trans. Algorithms **13**(2), 24–1 (2017)

5. Bebel, J., Ojha, D.: Ferveo: Threshold decryption for mempool privacy in bft networks. Cryptology ePrint Archive (2022)
6. Ben-Or, M.: Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In: Proceedings of the second annual ACM symposium on Principles of distributed computing. pp. 27–30 (1983)
7. Bentov, I., Ji, Y., Zhang, F., Breidenbach, L., Daian, P., Juels, A.: Tesseract: Real-time cryptocurrency exchange using trusted hardware. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1521–1538 (2019)
8. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Annual international cryptology conference. pp. 213–229. Springer (2001)
9. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)
10. Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., et al.: Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. Chainlink Labs **1** (2021)
11. Burdges, J., De Feo, L.: 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. pp. 302–326. Springer (2021)
12. Cachin, C., Mićić, J., Steinhauer, N., Zanolini, L.: Quick order fairness. In: Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers. pp. 316–333. Springer (2022)
13. Cascudo, I., David, B.: Scrape: Scalable randomness attested by public entities. In: International Conference on Applied Cryptography and Network Security. pp. 537–556. Springer (2017)
14. Cascudo, I., David, B., Garms, L., Konring, A.: 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. pp. 651–680. Springer (2023)
15. Chiang, J.H.y., David, B., Eyal, I., Gong, T.: Fairpos: Input fairness in proof-of-stake with adaptive security. Cryptology ePrint Archive (2022)
16. Choudhuri, A.R., Garg, S., Piet, J., Policharla, G.V.: Mempool privacy via batched threshold encryption: Attacks and defenses. Cryptology ePrint Archive (2024)
17. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 910–927. IEEE (2020)
18. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and tusk: a dag-based mempool and efficient bft consensus. In: Proceedings of the Seventeenth European Conference on Computer Systems. pp. 34–50 (2022)
19. Duan, S., Reiter, M.K., Zhang, H.: Secure causal atomic broadcast, revisited. In: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 61–72. IEEE (2017)
20. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM (JACM) **35**(2), 288–323 (1988)
21. Feldman, P., Micali, S.: Byzantine agreement in constant expected time. In: 26th Annual Symposium on Foundations of Computer Science (sfcs 1985). pp. 267–276. IEEE (1985)

22. Gailly, N., Melissaris, K., Romailler, Y.: tlock: practical timelock encryption from threshold bls. Cryptology ePrint Archive (2023)
23. Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: hints: Threshold signatures with silent setup. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 57–57. IEEE Computer Society (2023)
24. Garg, S., Kolonelos, D., Policharla, G.V., Wang, M.: Threshold encryption with silent setup. Cryptology ePrint Archive (2024)
25. Gehrlein, W.V.: Condorcet's paradox. Theory and Decision **15**(2), 161–197 (1983)
26. Gupta, T., Pai, M.M., Resnick, M.: The centralizing effects of private order flow on proposer-builder separation. In: 5th Conference on Advances in Financial Technologies (AFT 2023). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2023)
27. Heimbach, L., Wattenhofer, R.: Sok: Preventing transaction reordering manipulations in decentralized finance. In: 4th ACM Conference on Advances in Financial Technologies (AFT) (2022)
28. Kelkar, M., Deb, S., Long, S., Juels, A., Kannan, S.: Themis: Fast, strong order-fairness in byzantine consensus. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 475–489 (2023)
29. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: 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. pp. 451–480. Springer (2020)
30. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on security and privacy (SP). pp. 839–858. IEEE (2016)
31. Kursawe, K.: Wendy, the good little fairness widget: Achieving order fairness for blockchains. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 25–36 (2020)
32. Malkhi, D., Nayak, K.: Hotstuff-2: Optimal two-phase responsive bft. Cryptology ePrint Archive (2023)
33. Malkhi, D., Szalachowski, P.: Maximal extractable value (mev) protection on a dag. In: 4th International Conference on Blockchain Economics, Security and Protocols. p. 1 (2023)
34. Mamageishvili, A., Schlegel, C., Sudakov, B., Sui, D.: Searcher competition in block building. arXiv preprint arXiv:2407.07474 (2024)
35. Momeni, P., Gorbunov, S., Zhang, B.: Fairblock: Preventing blockchain front-running with minimal overheads. In: International Conference on Security and Privacy in Communication Systems. pp. 250–271. Springer (2022)
36. Pedersen, T.P.: 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. pp. 522–526. Springer (1991)
37. Piet, J., Fairoze, J., Weaver, N.: Extracting godl [sic] from the salt mines: Ethereum miners extracting value. arXiv preprint arXiv:2203.15930 (2022)
38. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 198–214. IEEE (2022)
39. Reiter, M.K., Birman, K.P.: How to securely replicate services. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(3), 986–1009 (1994)
40. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)

41. Rondelet, A., Kilbourn, Q.: Threshold encrypted mempools: Limitations and considerations. arXiv preprint arXiv:2307.10878 (2023)
42. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys (CSUR) **22**(4), 299–319 (1990)
43. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
44. Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: Dag bft protocols made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2705–2718 (2022)
45. Torres, C.F., Mamuti, A., Weintraub, B., Nita-Rotaru, C., Shinde, S.: Rolling in the shadows: Analyzing the extraction of mev across layer-2 rollups. arXiv preprint arXiv:2405.00138 (2024)
46. Wahrstätter, A., Ernstberger, J., Yaish, A., Zhou, L., Qin, K., Tsuchiya, T., Steinhorst, S., Svetinovic, D., Christin, N., Barczentewicz, M., et al.: Blockchain censorship. arXiv preprint arXiv:2305.18545 (2023)
47. Yang, L., Park, S.J., Alizadeh, M., Kannan, S., Tse, D.: {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). pp. 493–512 (2022)
48. Yang, S., Nayak, K., Zhang, F.: Decentralization of ethereum's builder market. arXiv preprint arXiv:2405.01329 (2024)
49. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 347–356 (2019)
50. Zhang, H., Merino, L.H., Qu, Z., Bastankhah, M., Estrada-Galiñanes, V., Ford, B.: F3b: A low-overhead blockchain architecture with per-transaction front-running protection. In: 5th Conference on Advances in Financial Technologies (2023)

A Permutation Algorithm

Algorithm 1 Permute [4]

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

B MEV Extraction Heuristics

Sandwich Attacks. It is a well-known strategy, where the attacker (*e.g.*, searcher or validator) observes the network to capture victim transactions creating a considerable rise/fall in the market price of an asset. The attack is performed by having the attacker place a front-running purchase/sell transaction tx_{a1} prior to the victim transaction tx_v and a back-running transaction tx_{a2} after to end the trade. Here we present the heuristics proposed in [38] to identify potentially successful sandwich attacks.

- Heuristic 1: The transactions tx_{a1}, tx_v and tx_{a2} must be included in the same block and in this exact order.
- Heuristic 2: Every front-running transaction tx_{a1} maps to one and only one back-running transaction tx_{a2}. This heuristic is necessary to avoid double counting revenues.
- Heuristic 3: Both tx_{a1} and tx_v transact from asset X to Y. tx_{a2} transacts in the reverse direction from asset Y to X.

- Heuristic 4: Either the same user address sends transactions tx_{a1} and tx_{a2}, or two different user addresses send tx_{a1} and tx_{a2} to the same smart contract
- Heuristic 5: The amount of asset sold in tx_{a2} must be within 90% to 110% of the amount bought in tx_{a1}. If the sandwich attack is perfectly executed without interference from other market participants, the amount sold in tx_{a2} should be precisely equal to the amount purchased in tx_{a1}.

Arbitrage. It enables earning revenue by taking advantage of the price slippage in simultaneous selling and purchasing assets across different markets. The arbitrage is performed by having the searcher (*e.g.*, searcher or validator) monitor the blockchain state change and grab a profitable opportunity in two typical ways, front-running and back-running. In the former, the searcher follows the blockchain state and tries to frontrun others at block B_{i+1} upon the receipt of block B_i. In the latter, the searcher observes the network to detect a transaction triggering a large change to asset price in an exchange and thus attempts to back-run this transaction. Here we present the heuristics proposed in [38] to identify potentially successful arbitrage extractions.

- Heuristic 1: All swap actions of an arbitrage must be included in a single transaction, implicitly assuming that the arbitrageur minimizes its risk through atomic arbitrage. We use s to denote a swap action which sells $\mathsf{in}(s)$ amount of the input asset $\mathsf{IN}(s)$ to purchase $\mathsf{out}(s)$ amount of the output asset $\mathsf{OUT}(s)$.
- Heuristic 2: Arbitrage must have more than one swap action.
- Heuristic 3: The n swap actions $s_1, \ldots, s_n$ of an arbitrage must form a loop. The input asset of any swap action must be the output asset of the previous action, *i.e.*, $\mathsf{IN}(s_i) = \mathsf{OUT}(s_{i-1})$. The first swap's input asset must be the same as the last swap action's output asset, *i.e.*, $\mathsf{IN}(s_0) = \mathsf{OUT}(s_n)$.
- Heuristic 4: The input amount of any swap action must be less than or equal to the output amount of the previous action, *i.e.*, $\mathsf{in}(s_i) \leq \mathsf{out}(s_{i-1})$.

C Threshold Encrypted Mempool

Malkhi and Szalachowski [33] present four approaches to building up an encrypted mempool with threshold security, including threshold cryptography, VSS, secret sharing with post-verification, and hybrid. We present another approach using PVSS and elaborate on an optimized variant that is relevant for this purpose. In what follows, we briefly describe *secret sharing with post-verification* and *hybrid* that could be used in BlindPerm depending on the application.

- 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 authors in [33] adapt a technique introduced in [47] 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 (that is already broadcast by the dealer) 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 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 [33] 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, a hybrid design is proposed where 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.