MEVade: An MEV-Resistant Blockchain Design

Julien Piet
UC Berkeley
Berkeley, CA, USA
piet@berkeley.edu

Vivek Nair
UC Berkeley
Berkeley, CA, USA
vcn@berkeley.edu

Sanjay Subramanian
UC Berkeley
Berkeley, CA, USA
sanjayss@berkeley.edu

***Abstract*—Ethereum is a popular blockchain that facilitates the creation of decentralized applications (dApps) and enables digital transactions to be executed without the need for a central authority. However, as in traditional markets, information asymmetry and market inefficiencies are used to the detriment of ordinary users via trading strategies that exploit "Miner Extractable Value" (MEV). We propose two extensions of Ethereum, one for proof of work (PoW), and one for proof of stake (PoS), that eliminate most forms of MEV by randomizing the execution order of transactions and hiding the content of transactions until their inclusion in a block. We simulate attack scenarios for both settings and provide detailed security properties and proofs.**

I. Introduction

Ethereum is today the most popular blockchain for Decentralized Finance (DeFi), providing a set of on-chain currencies, exchanges, and services that emulate a conventional financial market without reliance on central authorities. Despite Ethereum's promise of a secure, transparent, and immutable digital asset exchange platform, a plethora of information asymmetries persist (just as they do in traditional financial markets), and are heavily exploited at the cost of all other users. As early as 2014, Ethereum miners were already conspiring to perform frontrunning [1], a type of market manipulation that would be prohibited in regulated exchanges.

The concept of extracting value from DeFi transactions is coined Miner Extractable Value (MEV). Measurement studies of MEV in Ethereum have conclusively demonstrated that MEV exploitation is not only practical but, in fact, widely prevalent [2]. This presents an increasingly worrisome trend, as most forms of MEV harm ordinary users by impacting currency exchange rates while adding friction and uncertainty to transaction execution. The net effect is increased cost and uncertainty for DeFi users, to the benefit of only a select few validators who reap the rewards. Moreover, in PoW chains, MEV encourages forking; as of in March 2022, multiple blocks per week contained enough extractable value to incentivize forking attacks from Ethereum miners [1].

MEV is possible thanks to validators' advanced knowledge of transactions, and their full control over the ordering of blocks they propose. Transactions in Ethereum, as in many chains, are public until added to a block, which allows curious participants to peek, and possibly propose transactions that take advantage of them. This is used in MEV, but also for generalized frontrunning (GF), in which malicious participants can steal from honest participants, giving Ethereum's transaction pool the "Dark Forest" nickname [3], [4].

Both academia and industry alike have proposed methods to address the MEV problem, such as improved decentralized exchange (DEX) contracts, enforcement of ordering consistency, and transaction encryption. However, none have truly resolved the issue in the proof-of-stake (PoS) setting.

In this paper, we propose two practical protocols, one for PoW and one for PoS, to mitigate MEV and GF, by (1) encrypting transactions until they are included in blocks and (2) randomizing the execution order of transactions. We analyzed the security properties of our PoS protocol, which preserves the safety and liveness guarantees of Ethereum, while supporting Ethereum's current transaction throughput.

The remainder of this paper is structured as follows: We first provide background on Ethereum, both in its PoW and PoS variants (§II). We then review previous efforts in this space (§III), and present our protocols for PoW (§V) and PoS (§VI). Finally, we discuss the limitations of our approach (§VII).

II. Background

A blockchain enables a network of nodes to maintain a decentralized ledger of transactions without necessitating the use of a trusted authority that is responsible for record-keeping. Ethereum [5] is presently one of the largest blockchains in the world by transaction volume, and is the most popular blockchain that supports smart contracts that can be arbitrarily complex in terms of Turing completeness [2]. A typical example of a smart contract that may be invoked by a transaction is a decentralized exchange (DEX) of one token for another.

Central to the design of a blockchain system is the process by which nodes in the network achieve consensus on a transaction history. Each block in a blockchain contains a sequence of transactions; once a block becomes part of a node's chain, the node will consider all of those transactions to be part of the canonical transaction history. While new blocks are always "gossiped" between nodes, a node looking for the newest version of the blockchain may see several different versions of the chain when looking at the blocks broadcast by other nodes. In Ethereum, to resolve this conflict, the node will refer to a forking rule to decide which chain is valid.

Nodes should not be able to tamper with the transaction history. Blockchains use consensus mechanisms for participants to agree on a common history and come up with new blocks.

978-8-3503-1019-1/23/$31.00 ©2023 IEEE

Ethereum used to rely on a proof of work consensus (PoW), before recently switching to proof of stake (PoS).

In PoW, which was introduced for the Bitcoin blockchain [6], each block includes a value called a *nonce* that establishes the validity of the block. Nodes that publish new blocks after finding nonces for them are called *miners*. The forking rule states the longest chain of blocks is valid. Crucially, given the contents of a block, it is computationally difficult to find a nonce that will be judged valid. Thus, rewriting the chain requires mining faster than the rest of the network, in order to produce the longest chain.

By contrast, PoS requires nodes to have a financial stake (i.e. capital) to participate in the process of establishing new blocks. These nodes are called *validators*. Ethereum's PoS splits time into epochs of 32 slots, each slot lasting 12 seconds. In each slot, a random committee of validators is chosen. One of these validators proposes a block, while the others cast votes to elect the valid chain. Finally, a third class of validators in each slot aggregate votes to be included in future blocks [7].

MEV. In both the PoW and PoS variants of Ethereum, miners and validators can take advantage of the public nature of the mempool to exploit information asymmetries to their own benefit. There are several recognized forms of MEV [1]:

- **Frontrunning** avoids the price slippage caused by a target transaction by inserting a transaction before the target.
- **Backrunning** exploits the price slippage caused by a target transaction by inserting a transaction after the target.
- **Sandwiching** combines front and back running to exploit the price differential before and after a target transaction.
- **Arbitrage** takes advantage of pricing gaps of the same asset in different markets, and is the most well-known but least prevalent form of MEV. Unlike the above attacks, Arbitrage is generally seen favorably in financial markets, as it does not rely on insider knowledge of transactions. Still, arbitrage transactions benefit from placement at the top of a block so that others cannot benefit from the opportunity.

III. Related work

There have been multiple works analyzing MEV and trying to understand the scale of the issue [1], [2], [4], [8], [9]. Although the incentives of stakeholders on Ethereum, the chain with most DeFi activity, are not aligned with reducing MEV, it is nonetheless an important topic of research given that it has been shown to create network congestion and hence increase transaction prices not just for DEX transactions but also for simple transfers affecting the system.

MEV-resistant DEX. One strategy is to change the model used to redefine how DEXs process transactions instead of using instant confirmation such as AMMs. For example, McMenamin et al. [10] provide game-theoretic guarantees against MEV extraction by using frequent batch auctions, while Heimbach et al. [11] do the same by setting the slippage tolerance in swaps to prevent sandwich attacks. Other approaches aim to use off-chain communication [12] or multi-party computation [13] to eliminate arbitrage opportunities. Finally, A2MM [14] advocates for a unified AMM for the blockchain to mitigate sandwich and back-running attacks. While these designs provide better guarantees than standard AMMs, they are significantly more complicated, sometimes with more trust assumptions. This also increases the surface area of attack for smart contract bugs. Finally, designing better individual DEXs does not limit MEV across different DEXs.

Order-fairness. An alternative direction to mitigating MEV is to have miners agree on the order in which transactions were submitted to the mempool, with the idea that limiting the reordering of transactions limits the ability of a miner to extract MEV. Byers [15] gives an overview of ordering consensus, while Themis [16] and Aequitas [17] achieve decentralized order fairness in a decentralized manner, but are only able to enforce a weak version of order-fairness, coined *batch order-fairness*, that does not eliminate MEV. Wendy [18] is a set of protocols for ensuring order fairness based on linearizability. Unfortunately, this too only provides coarse order-fairness and does not mitigate all reordering possibilities.

Threshold Encryption. Finally, some prior work suggests that clients first encrypt transactions that get decrypted only after the position of the transaction on the blockchain has been fixed. Helix [19] is a PBFT-based blockchain that uses threshold encryption to hide the content of transactions. Ferveo [20] extends this idea and proposes a protocol for Mempool Privacy on any BFT consensus-based blockchain. These solutions are elegant and provide strong privacy guarantees; however, they do not immediately extend to Ethereum, which is not based on BFT consensus and instead uses GASPER [7]. In contrast, this work proposes a practical encryption mechanism, inspired by Ferveo but supported by Ethereum, and crucially also introduces random transaction ordering to further deter MEV.

Alternatively, one may consider using "Timelock Encryption" via Verifiable Delay Functions [21]. However, this would require that a transaction always be included in a block within a certain period of time. We argue that this is not a reasonable model for Ethereum, given that transactions can potentially wait indefinitely in the mempool. Further, if a transaction is included early, then the updated state from this transaction is unavailable until much later, affecting the user experience. One could also use trusted execution environments (TEEs), but this approach crucially relies on the integrity of TEEs, which has been repeatedly broken [22]. For instance, a TEE-based blockchain named Secret Network was recently rendered completely vulnerable due to an attack on the underlying TEE [23], whereby researchers were able to extract the secret key.

IV. Techniques

In this section, we describe the primitives we use for both our protocols. We prevent MEV by randomizing the execution order of transactions in a block (so no validator can execute reordering MEV strategies) and by blinding transactions until they are committed to (to prevent frontrunning). Combined, these two techniques make most MEVs impractical. For instance, front and back running rely heavily on having advance knowledge of a transaction, which is made impossible by blinding. Further, the likelihood that the three transactions of a sandwich attack will be randomly ordered in the correct sequence is only $\frac{P(N,N-3)}{P(N,N)} = \frac{1}{6}$. Finally, while our schemes do not eliminate arbitrage, they ensure that all nodes have equal extraction odds rather than advantaging validators.

A. Random ordering

Our first primitive serves to randomize the execution order of transactions in a block. This execution must happen after the block has been committed to, but before the next block is submitted, in order to support DeFi applications without adding latency or uncertainty. We require a source of randomness to execute transactions in a random ordering, which must be agreed upon by all parties in the chain, needs to be refreshed for every block, and cannot be revealed before the block has been committed to (or else the block proposer could adaptively select the set of transactions in a block to influence extracted value). We abstract this concept by using a randomness beacon that interfaces with the blockchain to provide fresh randomness for each block. We also introduce $\rho(N, R)$, which returns a pseudorandom permutation of $[1, N]$.

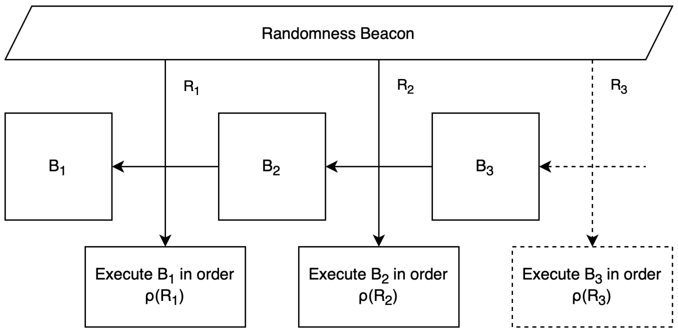

Randomness Beacon
R1
R2
R3
B1
B2
B3
Execute B1 in order ρ(R1)
Execute B2 in order ρ(R2)
Execute B3 in order ρ(R3)

Fig. 1. Use of randomness beacon for random transaction ordering.

Figure 1 shows a high-level view of random ordering. First, validators propose a block B, including a set of N transactions, ordered in a canonical way (e.g., by hash). Next, the randomness beacon emits a random value R, which is agreed upon by all. This value is used to generate a permutation $\rho(N, R)$ that determines the transaction execution order.

We now provide practical solutions for implementing this randomness beacon, both for PoW and PoS blockchains. In particular, we make sure (1) our randomness is unpredictable before block commitments, yet (2) agreed upon by all after the fact. We propose two main strategies for implementing such randomness beacons: ones derived from chain data, and others derived from participants in the chain.

a) Chain data randomness.: In this setting, random values are derived using chain data. Initially, this seems to violate property (1), as a single user can compute randomness independently from other users before submitting a block. However, we circumvent this problem with delay functions, which are difficult to compute, taking a non-compressible amount of time, but have an efficiently verifiable output. As long as the block time is smaller than the delay, miners are not able to adaptively select transactions to alter the final order.

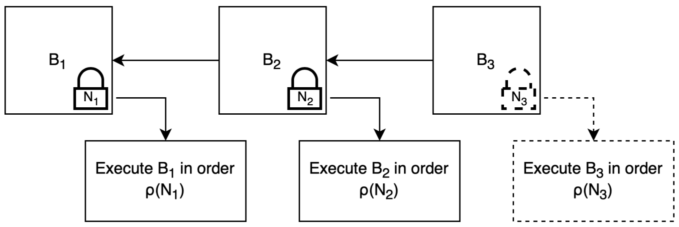

B1
N1
B2
N2
B3
N3
Execute B1 in order ρ(N1)
Execute B2 in order ρ(N2)
Execute B3 in order ρ(N3)

Fig. 2. Using mined nonces as a randomness source for proof of work.

In a proof-of-work setting, this can be implemented without any additional tools. Miners compete to find a nonce producing a low enough hash value. This nonce is difficult to find yet easy to verify by design, so we use it to seed our pseudorandom permutation function ρ, as illustrated in Figure 2. A powerful enough miner could potentially solve the PoW puzzle multiple times, and publish the block with the highest extracted value. However, as we show in Section V-A, non-majority miners have a negligible likelihood of mining multiple blocks before any other miner can solve a single PoW puzzle.

In a proof of stake setting, we cannot use the same trick: blocks are signed by validators instantly, so there is no natural delay. Verifiable delay functions (VDF) are a candidate solution. However, their correct implementation is challenging. Instead, in PoS, we rely on a second form of randomness, derived from other nodes in the network. In particular, we can bootstrap randomness directly from blinded transactions.

b) Consensus-derived randomness: Alternatively, instead of deriving random values from chain data, here we use inputs from participants to generate a random value. By combining user inputs, we derive a value that no single node has influence over, yet every participant agrees upon. The advantage of this type of solution is no validator can determine the ordering of a block alone. However, in its simplest form, it also comes with a significant weakness: the last user to provide a random value can adaptively choose their value and thus have total control over the random output.

We can solve this using verifiable user contributions, meaning inputs must be random but not arbitrary. The node cannot choose its contribution as it could be used to its own advantage. A standard way of achieving this uses digital signatures: inputs must be the digital signature of a known piece of information using a node's secret key, so every other party can verify the signature is valid. Ethereum's own RANDAO algorithm, used to randomly select validator committees for block slots, uses this primitive. In this setting, the last node cannot fully influence the random value, but can still have one bit of influence over the final result, by choosing or not to withhold its value. In particular, a set of k conspiring nodes can exert k bits of influence over the random value, and thus can potentially have a significant impact on the final permutation.

Thankfully, we can combine random input sharing and transaction blinding to prevent this last problem: each blinded transaction contains a random input (e.g., the signature of the transaction nonce), which is only revealed when the transaction is decrypted. As long as decryption is atomic, meaning either all transactions in a block are revealed or none are, as long as at least two transactions in the block are not conspiring, the resulting transaction ordering is truly random to all nodes. We now describe our primitives for blinding.

B. Transaction Blinding

Hiding transactions prevents MEV by making them unreadable until execution. We are only concerned with hiding the recipient of the transaction, its data contents, and its value. Other parts of the transaction need to remain in the clear for miners to ensure they are profitable, and that the sender has enough funds. Moreover, other fields do not provide useful information for frontrunners (except for correlation attacks). Fully encrypting the content of a transaction poses significant hurdles without providing additional MEV protection, thus we chose to only hide the transaction recipient, content, and value.

As for random ordering, it is important that the transaction be uncovered before the next block in the chain is proposed; waiting any longer would slow DeFi trades and thus add uncertainty to asset prices. This necessarily implies we cannot guarantee transaction privacy in the event of forks; all blocks on a dead branch will have revealed their transactions, yet these transactions might not be included in the main chain. This is mostly an issue in PoW chains, in which occasional forks will inevitably leak the content of some transactions. There are two main ways to hide transactions: "Commit-reveal" and "Threshold decryption." We chose to use a commit-reveal structure for PoW and threshold decryption for PoS, which already uses committees.

a) Commit-reveal: In this setting, nodes commit to a transaction and only reveal it once it's been included in a block. In practice, this is done using a symmetric cipher. The sensitive part of the transaction is encrypted using a secret one-time key. Once the sender node sees a block containing the transaction, it reveals the key, so everyone can decipher the transaction. Nodes should only reveal keys once they trust the block will be included in the chain; otherwise, they might be tricked into revealing the contents of a transaction. As discussed earlier, this is necessary in order to avoid latency.

The advantage of this solution is its simplicity, and the fact it does not rely on a committee of nodes to decrypt transactions. However, these properties come at a cost. Nodes must stay online anytime they have a pending transaction, and the node must be aware of the key, and thus knows the content of the transaction. This is a potential issue: most transactions are submitted through third-party nodes, since most users do not have their own private Ethereum node. Consensus is also more difficult to reach, since all nodes in the network must agree on the set of keys that have been released for a specific block.

b) Threshold Public-Key Encryption: Threshold Public-Key Encryption (TPKE), albeit more complex than a commit-reveal scheme, has several advantages. It allows users to encrypt a transaction and go offline. In fact, as long as a user knows the public key for the block slot it is targeting, they can encrypt the transaction themselves, such that no node is aware of its contents (including the node submitting the transaction). TPKE also supports a degree of node failure. Previous work, such as Ferveo [20], uses TPKE to hide transactions. We use a simulation-based definition of adaptive CCA (chosen-ciphertext attack) secure TPKE as defined by Canetti and Goldwasser [24], which was shown to imply the strong notion of IND-CCA2. This strong notion of security ensures, in particular, the scheme will not be malleable, thus attackers cannot change the contents of an encrypted transaction. We restrict protocols Π_{TPKE} for TPKE to consist five PPT algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Verify}, \mathsf{Combine})$ as defined below:

- $\mathsf{Setup}(1^\kappa, n, t) \to \{\mathsf{pk}, \mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\}$: Takes as input a security parameter and positive integers n, t and outputs a public, verification key and secret keys with threshold $t+1$.
- $\mathsf{Enc}(\mathsf{pk}, m; \rho) \to \mathsf{ct}$: Takes as input the public key pk, a message m and randomness ρ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{ct}, sk_i) \to m_i$: Takes a secret key sk_i and ciphertext ct and outputs a partial decryption of message m_i.
- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, m_i) \to \{0, 1\}$: Takes as input the public key, verification key, and partial decryption of message and outputs 1 if and only if the share is a valid decryption.
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \{m_i\}_{i \in S \subseteq [n]}) \to m$ takes as input $t+1$ partial decryptions of the message and reconstructs m.

For a protocol Π_{TPKE} to be t-secure, it must securely emulate the following ideal functionality [25][1].

$\mathcal{F}_{\mathsf{TPKE}}$

Parties: Encrypting user E and Servers $(S_1, \ldots, S_n)$.
Parameters: Space of receipts $\mathcal{C}$, number of servers n and threshold $0 < t \leq n$.

- **Setup.** Adversary specifies a distribution Γ over $\mathcal{C}$.
- **Encryption.** When E sends (Enc, m), sample a receipt $c \leftarrow \Gamma$ and store (c, m). Send c to E.
- **Decryption.** When $t+1$ servers send (Dec, c), if a tuple (c, m) has been stored, send m to the servers. Else, send $\perp$ to the servers.

Fig. 3. Ideal Functionality for Threshold Public-Key Encryption scheme.

We rely on Ferveo's TPKE as a practical implementation of such a scheme. We refer the reader to the appendix of the original paper [20] for proof of semantic security. However, Ferveo itself is not well suited for Ethereum, as its bandwidth and computational requirements do not scale well for the 12,000 validators per slot in Ethereum.

Our solution closely adapts Ferveo's algorithm to Ethereum, but only uses aggregators instead of all validators to hold secret shares, making it more efficient. Section VI provides a detailed description of our protocol.

V. Proof of Work protocol

We describe in this section our proposed mechanism for committee-less transaction encryption in proof of work. Our system is designed as an extension of Ethereum but can be applied to other PoW chains. We blind transactions using a

[1]In the original paper, the authors actually define and construct protocols satisfying the stronger Universally Composable notion of security [26]

commit-reveal structure, which leverages two types of blocks, as shown in Figure 4. The first block, called the Transaction Block, is analogous to blocks in traditional blockchains, except that the transactions in this block are encrypted. The second type of block, called Key Blocks, contains the secret keys for deciphering the transactions in transaction blocks.

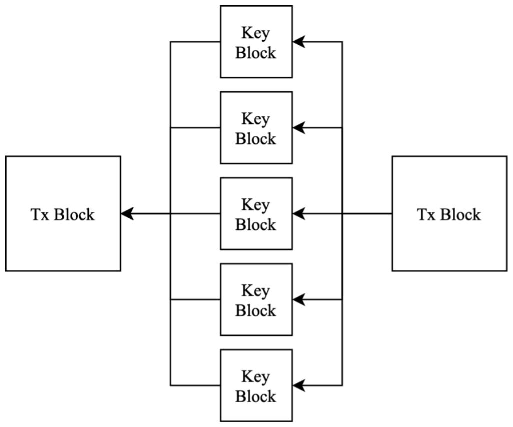


Fig. 4. Ordering of transaction and key blocks

When a node submits a transaction to the network, it generates a secret key and uses it to encrypt the data, recipient, and value. Miners then commit to a set of encrypted transactions in a transaction block. Nodes whose transactions are included in the blocks release their keys to the network, and miners can then build a key block containing these keys.

In order to provide censorship resistance, the network expects a fixed number of key blocks k to be submitted to the network before it can move on to the next transaction block. These blocks can be submitted in parallel by different miners, and only need to point toward the original transaction block. Once k key blocks have been created, nodes can use these to decipher the original transactions. They then use the mined Transaction Block nonce to shuffle the transaction order before execution and update the state machine accordingly before proceeding to the next transaction block. This next transaction block must include a pointer to the k valid key blocks.

One potential pitfall of this mechanism is that miners can choose not to include some keys in key blocks, censoring some transactions. Requiring a number of distributed key blocks helps mitigate this issue, as many miners would need to collude in order to censor a particular transaction since any block can contain the key. Increasing k leads to better censorship resistance, at the cost of a larger overhead for the network, especially for proof of work, since nodes will have to wait for k miners to mine key blocks, instead of a single miner. If a node submits a transaction but never releases the key, this transaction will be skipped when others cannot find the key in a key block, and the user will be charged max gas.

A. Simulation

We implemented the random ordering modification to the Ethereum protocol in the Proof-of-work setting. Our implementation is a modification of goethereum (Geth) [27]. In Geth, the code for the miner's processing of transactions in a block is separate from the code that nodes use when processing transactions received from a miner. We modified both of these sections so that they use the random ordering scheme.

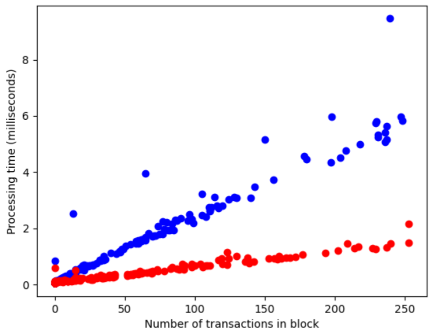
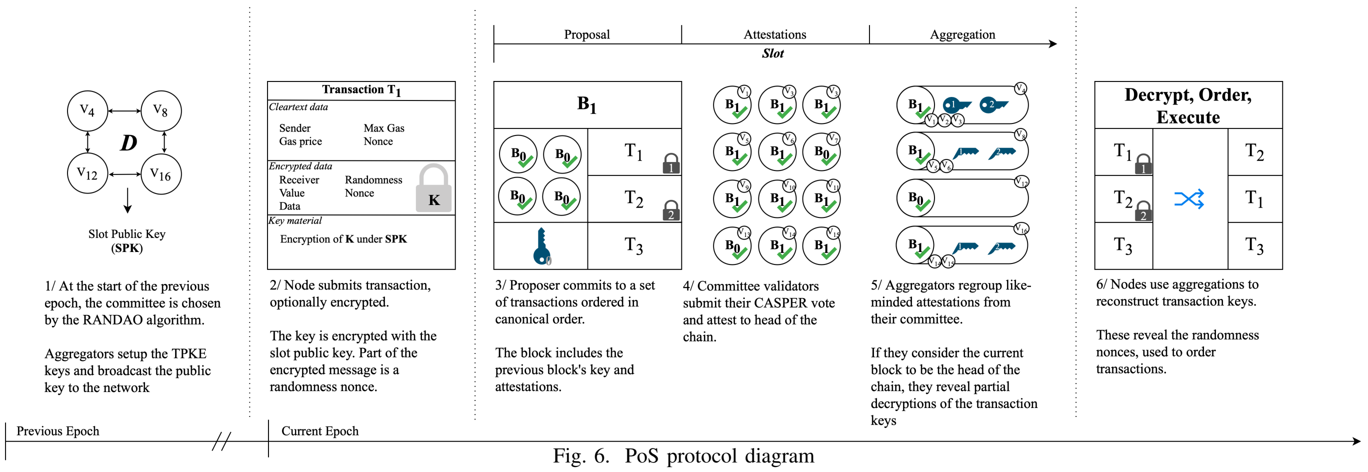


Fig. 5. Time for a client node to process the transactions in a received block in the PoW setting with (blue) and without (red) the random ordering scheme.

We conducted a timing experiment to compare the amount of processing time for transactions required by a client node with and without the random ordering scheme. We simulate a client node by running the same process as the mining node, but without activating the mining functionality, such that it only receives blocks mined by the mining node. In this experiment, we have a single miner and a single client. Each transaction sends a fixed amount of ETH from a fixed sender account to a fixed receiver account. We insert 10,000 transactions; between each pair of transactions, we sample an amount of delay time uniformly at random between 0 and 0.2 seconds. Figure 5 shows the amount of time that a client node takes to process the transactions in a block when using the random ordering scheme vs. the time taken with the baseline protocol. The results show that there is a modest overhead in the processing time for the client that is proportional to the number of transactions in the block. In practice, the number of transactions per block is likely to be less when using the random ordering scheme because transactions are added to the block based on the maximum gas of the transaction rather than the actual gas consumed. For the miner, we expect that the overhead would be low relative to the cost of mining.

A miner able to mine N blocks before anyone else can mine a single block will be able to choose the best MEV option amongst N blocks. We compute the probability that one miner with hash rate a can achieve this. We can model the time taken for the miner to mine his ith block as $X_i \sim \mathrm{Exp}(a/N)$. Since the combined hash rate of all of the other miners is $1-a$, we can model the time taken for any other miner to mine one block as $Y \sim \mathrm{Exp}(1-a)$. We aim to find $\Pr[\max(X_1, ..., X_N) \leq Y]$, where X_is are independent. Thus:

$$\begin{aligned}\Pr[X_i \leq t] &= 1 - e^{-at/N} \\ \Pr[\max(X_i) \leq Y] &= \int_0^\infty (1-a)e^{-(1-a)t}(1-e^{-at/N})^N dt \\ &= \frac{\Gamma\left(1+N\right)\Gamma\left(1+N\left(\frac{1}{a}-1\right)\right)}{\Gamma(1+N/a)}\end{aligned}$$



Proposal
Attestations
Aggregation
Slot
V4
V8
D
V12
V16
Slot Public Key
(SPK)
1/ At the start of the previous epoch, the committee is chosen by the RANDAO algorithm.
Aggregators setup the TPKE keys and broadcast the public key to the network
Transaction T1
Cleartext data
Sender
Gas price
Max Gas
Nonce
Encrypted data
Receiver
Value
Data
Randomness
Nonce
K
Key material
Encryption of K under SPK
2/ Node submits transaction, optionally encrypted.
The key is encrypted with the slot public key. Part of the encrypted message is a randomness nonce.
B1
B0
T1
T2
T3
3/ Proposer commits to a set of transactions ordered in canonical order.
The block includes the previous block's key and attestations.
4/ Committee validators submit their CASPER vote and attest to head of the chain.
5/ Aggregators regroup like-minded attestations from their committee.
If they consider the current block to be the head of the chain, they reveal partial decryptions of the transaction keys
Decrypt, Order, Execute
6/ Nodes use aggregations to reconstruct transaction keys.
These reveal the randomness nonces, used to order transactions.
Previous Epoch
Current Epoch

Fig. 6. PoS protocol diagram

Figure 7 plots this probability for different values of a and N: Even a miner with a 50% hash rate has a very low likelihood of successfully mining more than 3 blocks before anyone else. This analysis also applies to the number of key blocks. A miner with 50% of the hash rate only has a 0.4% likelihood of mining 5 key blocks before anyone else.

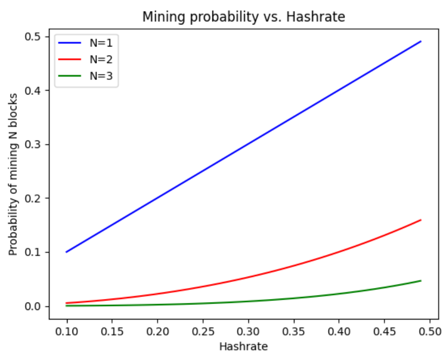


Fig. 7. Probability that in PoW a miner with hashrate a (x-axis) can mine N blocks before any other miner mines a single block, AKA the probability they can manipulate transaction ordering under our random ordering scheme.

However, using more key blocks increases the time to process a block. Assuming we keep the same mining difficulty for key blocks as for transaction blocks, a single key block doubles the time it takes to process a block since both the key block and transaction block must be mined sequentially. Requiring 5 key blocks triples the expected time. Thus, setting the number of key blocks allows one to select the best compromise between speed and security.

VI. Proof of Stake protocol

Our proof of stake protocol closely resembles Ferveo but adapts the algorithm to run efficiently with Ethereum's consensus layer. We build on top of the GASPER [7] Ethereum protocol. We first introduce 3 primitives used to interface the TPKE scheme with Ethereum.

- $\mathcal{D}$, a distributed key generation algorithm, in which N parties agree on a public key broadcast to the entire network.
- $\mathcal{B}$, a blinding procedure, that takes a transaction and a set of public keys and returns a blinded transaction.
- $\mathcal{R}$, a reveal algorithm, which uses partial decryptions to reveal and order blinded transactions.

As in GASPER, time is divided into epochs and slots, and each slot has a randomly chosen set of validators that can perform one of three tasks: proposing a block, attesting to a block, or aggregating attestations. There is a single proposer for each block; however, we fix the number of aggregators to a constant value C_A. More specifically, each slot gets assigned a proposer and a set of attestors at the start of the previous epoch, at least 32 slots in advance. As in GASPER, these are separated into committees. We choose a constant number of aggregators per committee. Once these roles are determined, aggregators run algorithm $\mathcal{D}$ to generate a public key pk. This public key is signed by each aggregator, broadcast to the entire network, and advertised in a subsequent block.

Nodes submit transactions to the network as in Ethereum, except that they now have the option of crafting blinded transactions. This is done using $\mathcal{B}$, which uses the public keys of target blocks to blind the transaction for inclusion in specific time slots. Blinded transactions contain a signature of the transaction nonce, used as a randomness source.

In GASPER, single slots are divided into three subparts: the proposer submits a block during the first third, validators submit their attestations during the second third, and aggregators summarize these votes in the last third. We keep this structure but slightly change what occurs in a slot, as illustrated in Figure 6. During the first part of the slot, the proposer submits a block, that can contain both blinded and clear-text transactions. However, blocks have the following modifications: (A) transactions must be ordered in canonical order, such as by hash value, (B) the block includes the keys for the transactions of its parent block, and (C) blocks can optionally contain advertised public keys for future slots.

In GASPER, during the second step of a slot, validators submit attestations containing two votes: One pointing to the current head of the chain, and a finality layer vote. Validators will vote on the block proposed in the first step if it is valid, and points to the correct parent block. The only modification to this setup is that we require an additional check for a block to be valid: it needs to include the correct keys to the transactions in the previous block. By "correct", we mean these keys must have been derived from the reveal algorithm $\mathcal{R}$: every honest node in the network should agree on the same keys.

Our most important change comes in the last part of the slot. Aggregators, now known ahead of time, will still aggregate like-minded votes from validators in their committee using BLS signatures. However, if they believe the proposed block to be the head of the chain, they reveal a partial decryption of the blinding keys for the transactions in the block, using Enc_{TPKE}. Finally, if at least two-thirds of aggregators have revealed partial decryptions, any node can use $\mathcal{R}$ to interpolate the blinding keys for each transaction and recover the content of transactions. In particular, $\mathcal{R}$ applies a bitwise XOR to the blinded signatures to seed a random permutation ρ, which is used to randomly reorder the transactions before execution.

Algorithms. We now describe our four primitives, using Ferveo's TPKE as a starting point.

a) $\mathcal{D}$: Our distributed key generation algorithm is built around the setup phase of Ferveo's TPKE, replacing all validators by the subset of aggregators. However, this protocol itself is not adapted to Ethereum: nodes can only derive and verify the public key if they have followed the transcript of the DKG. Instead, we propose aggregators sign the public key using BLS signatures and the existing PKI, aggregate their signatures, and release them to the blockchain to be included in a future block, so nodes can rely on on-chain information to determine the public key of future slots.

Nodes must verify that at least two-thirds of aggregators signed a slot public key. We use a bitmap to track the list of signatories, as described in Algorithm 1. Proposers include the slot public keys with the most signatures in their block.

DKG algorithms are expensive. Ferveo's implementation requires $\mathcal{O}(N^2)$ communications, and $\mathcal{O}(N^2)$ computation per aggregator. Thankfully, we only need a small number of aggregators (around $C_A = 128$); according to Ferveo's simulation, this should take less than a second for computation and require under 200kB total communication. A committee is elected about 6 minutes before the slot it supervises.

b) $\mathcal{B}$: Our blinding procedure is almost identical to Ferveo's encryption process, but transactions are extended to include a randomness seed and support multi-slot encryption.

c) $\mathcal{R}$: Our reveal algorithm first verifies the partial decryptions of aggregators, then uses them to reveal all blinded transactions in a block, and finally randomly permutes them before execution. If not enough valid partial decryptions are available, it aborts and the block is not executed. If a transaction cannot be decrypted, the sender is charged the maximum gas fee, and the transaction is not executed.

Algorithm 1: $\mathcal{D}$, the DKG algorithm, for aggregator A_i, $i \leq N$ with BLS key pair $\left(p_i^{BLS}, s_i^{BLS}\right)$

```
$pk, vk, sk_i \leftarrow \mathsf{Setup}_{TPKE}(1^\kappa, N, 2N/3)$
$M_i \leftarrow \left(\mathbf{e}_i, \mathtt{Sign}_{BLS}\left((pk, vk), s_i^{BLS}\right)\right)$
**Gossip** $M_i$ with other aggregators
**for** $M$ *received* **do**
    **if** $M^0 == \mathbf{e}_j$ $\mathtt{and}$ $\mathtt{Verify}_{BLS}\left(M^1, p_j^{BLS}\right)$ **then**
        $M_i \leftarrow \left(M_i^0 + \mathbf{e}_j, \mathtt{Agg}_{BLS}\left(M_i^1, M^1\right)\right)$
**if** $M_i$ *includes at least* $2/3^{rd}$ *of aggregators* **then**
    Broadcast $M_i$ to network
```

Algorithm 2: $\mathcal{B}$, the blinding algorithm, for a transaction T with sender S and key pair $\left(pk^S, sk^S\right)$.

```
Sample a random AES key $k$;
Separate $T$ into $T_{clear}, T_{blind}$
$R \leftarrow \mathtt{Sign}_{BLS}(T_{clear}\mathtt{.nonce}, sk^S)$
$C \leftarrow \mathtt{Enc}_{\mathtt{AES\text{-}GCM}}((T_{blind}, R), k)$
$K \leftarrow []$
**for** *Target slot s with TPKE public key pk* **do**
    $K\mathtt{.append}(s, \mathsf{Enc}_{TPKE}(pk, k))$
Sign and Broadcast $T_{final} := (T_{clear}, C, K)$
```

Algorithm 3: $\mathcal{R}$, the reveal algorithm, for block $B = (T_1, \ldots, T_b)$, with TPKE keys (pk, sk), and partial decryptions m_i^j for aggregator j and transaction i.

```
**for** *Aggregator j with partial decryptions* **do**
    **if** $\exists i, \mathsf{Verify}_{TPKE}\left(pk, vk, m_i^j\right) == \perp$ **then**
        Ignore decryptions from aggregator $j$;
$R \leftarrow 0$;
**for** $T_i \in B$ **do**
    $k_i \leftarrow \mathsf{Combine}_{TPKE}\left(pk, vk, \{m_i^j\}_j\right)$; **if**
    $\mathsf{Dec}_{AES\text{-}GCM}(T_i, k) \neq \perp$ **then**
        $T_{blind_i}, R_i \leftarrow \mathsf{Dec}_{\mathtt{AES\text{-}GCM}}(T_i, k)$;
        $R \leftarrow R \oplus R_i$;
        $T_i \leftarrow (T_{clear_i}, T_{blind_i})$;
    **else**
        Discard $T_i$ and charge sender max gas
$B \leftarrow \rho(R)(B)$;            /* Reorder block */
Execute $B$;
```

Properties. Our proposal keeps the same structure as Ethereum's GASPER protocol, with a small update to the LMD-GHOST forking rule. In addition to the current criteria, each link in the chain is only considered valid if the block was successfully revealed in procedure $\mathcal{R}$.

- We achieve the same safety and plausible liveness guarantees as GASPER. As long as at least $2N/3$ validators are honest, the protocol is safe and can make progress, even in the asynchronous model.
- As long as $2N/3$ validators are honest, in the (1/3)—synchronous model defined in [7], our protocol achieves probabilistic liveness.

Our protocol achieves the following properties:

- **Blinding.** As long as $2N/3$ validators are honest, in a (1/3)—synchronous model, an adversary has a negligible chance of revealing blinded transactions in the mempool, and transactions are only revealed once included in a block.
- **Random ordering.** If (1) $2N/3$ validators are honest, (2) the network is (1/3)—synchronous, and (3) a block includes two blinded transactions with fresh nonces from non-conspiring entities, then no node can predict the execution order of the block until it has been committed to.

We sketch proofs of these properties in Appendix A.

VII. Discussion

Transaction choice. Blinding and randomizing execution ordering prevent MEV, but also make it more difficult for a proposer to reach its gas objectives. The gas price and max gas of a transaction is still revealed, so proposers can choose profitable transactions from senders with enough funds. However, proposers can no longer predict the gas usage of each transaction. One solution is to use the max gas as an indicator of gas usage for a transaction. This makes sure no block will ever go above the gas limit, however, it introduces a risk of denial of service. Users can simply specify a high max gas, but then only use a small amount, leading to under-utilized blocks and lower throughput. We can fix this by imposing that a percentage of unspent gas in a transaction is paid to the proposers. This eliminates the DoS attack but makes transactions costlier as a result.

Another approach is to add a min gas field to transactions. Proposers can select transactions based on this field, and will always get paid at least min gas for each transaction, regardless of the execution outcome. This introduces another issue: blocks can now exceed their max gas allotment, which might create a computational bottleneck.

Economic incentives. Not all transactions are subject to MEV, and thus do not require blinding. However, we want to ensure proposers are incentivized to include blinded transactions in their blocks. We do this by adding a fixed gas cost for the blinding procedure, to be paid by the sender. We have not set a specific value for this parameter — this requires further economic analysis of this new protocol, to account for the additional computational and network cost of blinded transactions. Transaction pricing will be affected by randomized execution ordering, since users will no longer pay to be first, but will pay to be included. We leave this to future work.

PoW limitations. Our PoW protocol adds significant latency to the chain, because of the delay incurred by key blocks. Throughput can be recovered by increasing the gas limit of each block to compensate for the added delay. PoW Ethereum also had significantly more orphaned blocks than PoS. In our current PoW protocol, any orphaned block will leak the content of the transaction that was included, meaning we cannot protect transaction privacy in the event of forks.

PoS limitations. While transaction size does not increase substantially, the partial decryption process adds network overhead, since we require a total of $C_A \times |B|$ partial decryption messages for C_A aggregators and $|B|$ blinded transactions.

The reveal operation adds overhead to each node in the network because of the $\mathsf{Combine}$ primitive. Using results from Ferveo, a $\mathsf{Combine}$ takes about 20ms per transaction for an aggregation pool of size 128. This means we can realistically include up to 200 blinded transactions per block before the delay becomes larger than the aggregation slot time of 4 seconds. This is enough to support the encryption of all current DeFi trades, however, we need a more efficient combined algorithm to increase the throughput of blinded transactions.

Since the sender's address is still public, proposers can censor some addresses. Hiding the full content of a transaction would remedy this issue, however, this creates additional hurdles for selecting transactions.

Users wanting to protect their trades can do so by blinding their transactions: this will provide them with the guarantee of both privacy and random order execution, protecting them from frontrunning. However, we would like to also randomize the execution order of transactions even if none of them are blinded. A potential solution to this problem is to use Verifiable Delay Functions (VDF) to generate pseudo-random values at some fixed delay after the proposal of a block.

Transactions can only be encrypted to a specific set of slots. If the transaction does not get included in any of these slots, the encrypted transactions will be dropped from the memory pool, and the user will need to resubmit a transaction encrypted to future slots. Our system is subject to brute-force attacks. Attackers can submit multiple transactions so one has a higher chance of being executed first, adding congestion to the network. However, this is costly for the attacker, since only one of the transactions will potentially succeed, while all will be charged a gas price. Users can mount a privacy attack, in which they duplicate the encrypted data and key material of a target transaction, but replace the sender's address and signature with their own. If this newly crafted transaction gets included in a block before the original, it can be decrypted, revealing the original transaction, but not executed, since the sender address will not correspond to the encrypted data, thus violating the original transaction's privacy. This can be fixed using a zero-knowledge proof that the sender in the cleartext data corresponds to the sender in the encrypted data.

VIII. Conclusion

MEV exploitation severely impacts the trust users put in Ethereum transactions. In this paper, we mitigate MEV by encrypting transactions and randomizing their order in a block. We propose two practical protocols, one for Ethereum's PoW protocol and another for the newer PoS variant. We simulate the probability of various attack scenarios and show our PoS protocol provides the same safety and liveness guarantees as Ethereum, while supporting the same transaction throughput and latency, and only slightly block and transaction sizes.

Availability

Our Ethereum random ordering implementation is available at https://github.com/sanjayss34/geth-random-order.

References

[1] J. Piet, J. Fairoze, and N. Weaver, "Extracting godl [sic] from the salt mines: Ethereum miners extracting value," 2022.

[2] 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 *IEEE Symposium on Security and Privacy*, 2020.

[3] D. Robinson, "Ethereum is a dark forest." https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest, 2020. Accessed: 2022-02-16.

[4] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?," *arXiv preprint*, 2021.

[5] G. Wood, "A secure decentralised generalised transaction ledger."

[6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[7] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," 2020.

[8] L. Zhou, K. Qin, A. Cully, B. Livshits, and A. Gervais, "On the just-in-time discovery of profit-generating transactions in defi protocols," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[9] Y. Wang, Y. Chen, S. Deng, and R. Wattenhofer, "Cyclic arbitrage in decentralized exchange markets," *Available at SSRN 3834535*, 2021.

[10] C. McMenamin, V. Daza, and M. Fitzi, "Fairtradex: A decentralised exchange preventing value extraction," *arXiv preprint*, 2022.

[11] L. Heimbach and R. Wattenhofer, "Eliminating sandwich attacks with the help of game theory," *arXiv preprint*, 2022.

[12] M. Ciampi, M. Ishaq, M. Magdon-Ismail, R. Ostrovsky, and V. Zikas, "Fairmm: A fast and frontrunning-resistant crypto market-maker," *IACR Cryptology ePrint Archive*, 2021.

[13] C. Baum, B. David, and T. K. Frederiksen, "P2DEX: privacy-preserving decentralized cryptocurrency exchange," in *Applied Cryptography and Network Security - 19th International Conference* (K. Sako and N. O. Tippenhauer, eds.), 2021.

[14] L. Zhou, K. Qin, and A. Gervais, "A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges," *arXiv preprint*, 2021.

[15] J. Byers, *Combating Front-Running in the Blockchain Ecosystem*. PhD thesis, Lehigh University, 2022.

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

[17] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, "Order-fairness for byzantine consensus," in *International Cryptology Conference*, 2020.

[18] K. Kursawe, "Wendy, the good little fairness widget: Achieving order fairness for blockchains," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020.

[19] D. Yakira, A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, and R. Tamari, "Helix: A fair blockchain consensus protocol resistant to ordering manipulation," *IEEE Transactions on Network and Service Management*, 2021.

[20] J. Bebel and D. Ojha, "Ferveo: Threshold decryption for mempool privacy in bft networks." Cryptology ePrint Archive, Paper 2022/898, 2022. https://eprint.iacr.org/2022/898.

[21] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions." Cryptology ePrint Archive, Paper 2018/601, 2018. https://eprint.iacr.org/2018/601.

[22] "Sgx.fail." https://sgx.fail.

[23] S. Network, "Successful resolution of xapic vulnerability."

[24] R. Canetti and S. Goldwasser, "An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack," in *EUROCRYPT*, Springer, 1999.

[25] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[26] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS*, IEEE, 2001.

[27] "Go-Ethereum." https://github.com/ethereum/go-ethereum.

Acknowledgments

We appreciate the advice and feedback of Guru Vamsi Policharla, Dawn Song, David Wagner, Vern Paxson, Raluca Popa, Mayank Rathee, and Emma Dauterman. This work was supported in part by the National Science Foundation, the National Physical Science Consortium, the Fannie and John Hertz Foundation, and the Berkeley Center for Responsible, Decentralized Intelligence.

Appendix

A. Proof of Stake properties

a) Safety and liveness proof sketch: The proof of the safety guarantee from the GASPER paper still holds, since we do not change the Ethereum finality layer. The proof for plausible liveness is almost identical to the original (Theroem 6.3 of [7]). We simply need to add, in addition to the plausibility of an honest proposer submitting the first block B in epoch j, that it is also plausible for 2/3rds of the aggregators to be honest and to decrypt transactions in B in time. Then, honest participants in the epoch will vote for B as the next supermajority link, keeping the chain live. Probabilistic liveness is trickier to prove. Section 7 of the original paper provides detailed proof, that we can adapt for our purposes. We need to change the equivocation game, so that we win if O_2 gets at least $2/3^{rd}$ of the validator votes for the slot, or if O_1 gets at least $2/3^{rd}$ of the validator votes for the slot **and** gets $2/3^{rd}$ of the aggregator votes and partial decryptions for the slot. This means the current block is chosen only if it gets enough votes and it gets decrypted. For committees with 128 aggregators or more, the lower bounds on the success rate of the equivocation game still hold. The rest of the proof works the same as the original paper since our proposal does not change justification votes.

b) Blinding proof sketch: We now sketch the proof of the blinding property. First, we show that the probability of randomly picking 2/3rds malicious aggregators when $2N/3$ validators are honest is negligible. We model this using a Binomial distribution and find the probability of this event for 128 aggregators to be around $10^{-}15$, happening once every 60 million years on average. Now, we know transactions are only revealed if they are part of a valid block. The only way for transactions to be revealed without being executed is for the block to be orphaned. Assuming (1/3)—synchronicity, this is not possible. For a slot starting at T and a slot duration s, if a valid block is proposed, aggregators will release partial decryptions at $T + 2/3s$, which will be received by the rest of the network by the start of the next slot at $T + s$. In particular, the next honest proposer will receive the transaction decryptions in time and the block will not be orphaned.

c) Random ordering proof sketch: The random ordering property is a consequence of the blinding property. As long as one transaction per block is unknown to an adversary, and contains a fresh nonce, that adversary will be unable to guess the transaction's randomness, and will not be able to predict the execution order until all transactions are revealed. Since transactions are only revealed after the block has been committed, the adversary will be unable to predict the execution order until after commitment. Thus, if any two transactions are from non-conspiring entities, no adversary can know the contents of both transactions, and thus the above property applies.