ISSN: 2769-545X

Research Article

Advances in Machine Learning & Artificial Intelligence

Strategic Mining in Proof-of-Stake with Practical Random Election

Zhuo Cai*

Hong Kong University of Science and Technology Hong Kong SAR, China

***Corresponding Author**
Zhuo Cai, Hong Kong University of Science and Technology Hong Kong SAR, China.

Submitted: 2025, Sep 26; **Accepted:** 2025, Oct 21; **Published:** 2025, Nov 07

Citation: Cai, Z. (2025). Strategic Mining in Proof-of-Stake with Practical Random Election. *Adv Mach Lear Art Inte, 6*(4), 01-10.

Abstract

The security of blockchain systems relies on the honest majority assumption. However, strategic mining threatens this assumption, because selfish miners can gain more block rewards than honest miners by attacks such as withholding blocks. Due to its significant implication, blockchain mining games have been studied in PoW and PoS under various settings using different methods. Nonetheless, this paper argues that the practical limitation of random beacons has not been exploited in strategic mining in PoS blockchains.

Current PoS blockchains use random beacons to randomly select validators for each slot. However, the randomness is usually fixed for multiple slots, due to the latency of distributed random beacon protocols. This indicates that validators actually know some information about the election result in the future, which contrasts with the Markov process models in previous analysis. Using this information, this paper presents a close to optimal mining strategy based on an optimal interval scheduling algorithm for each epoch. For proof-of-stake protocols with no propagation delay, we show that a validator with arbitrary proportion of stake can strictly benefit from strategic mining and get significantly higher block rewards than the previous strategies.

Keywords: Game Theory, Blockchain Mining Games, Miner Revenues

1. Introduction

The security of blockchain consensus protocols relies on a super-majority of honest miners or validators. For example, the Bitcoin consensus protocol, based on Proof-of-Work (PoW) and longest-chain fork selection rule, requires that more than 50% of the hash power should be controlled by honest miners. Proof-ofStake (PoS) protocols, such as Ouroboros, also assume that honest miners own more than 50% of the total stake [1]. If the honest majority assumption does not hold, disastrous attacks might happen, including double spending, censoring transactions and reverting the history. Even though it is reasonable to believe that malicious miners can never afford to become the super majority, blockchain security is further undermined by selfish mining attacks. Showed that malicious miners $\alpha < 1/2$ of hash power can publish more than α of blocks in the finalized chain, by withholding block proposals in attempt to exclude honest blocks from the blockchain, so that they can earn more block rewards than they should [2-4]. Gradually, malicious miners become relatively richer and will control more than 50% of hash power in the end to perform disastrous attacks.

Proof-of-Stake protocols have gained popularity in recent years due to its energy-efficiency compared to Proof-of-Work, as evidenced by Ethereum's update to switch from PoW to PoS. Selfish mining on PoS blockchains is also studied in the literature. For example, shows that a malicious miner with 32.8% of the total stake can strictly outperform a honest miner with the same stake, in proof-of-stake protocols with perfect randomness [5]. Perfect randomness means there is a perfect decentralized random beacon that emits a random number in each slot to select the miner for the slot. However, for security and efficiency reasons, in practice, decentralized random beacons in PoS emits fresh random numbers only once every epoch, where one epoch consists of multiple slots. For example, an epoch consists of 32 slots in Ethereum 2.0 and 432,000 slots in Cardano [6,7]. Therefore, miners know which

of the future slots in the current epoch or even in the next epoch belong to themselves. In contrast to PoW or PoS with perfect randomness where miner do not know whether they can publish blocks in the future, malicious miners can utilize this information to launch selfish mining attacks with arbitrarily small stake and earn more block rewards.

Our Contribution Our main contribution is to characterize the effect of the Knowledge about Future miner Election results (KFE) in proof-of-stake selfish mining. As far as we know, this is the first work to address the knowledge of future election results in PoS selfish mining. We extensively study selfish mining with KFE in different settings. In more detail, we have the following results:

- In the basic setting of Proof-of-Stake protocol with longest chain rule, perfect communication and the majority $(1 - \alpha)$ of stake controlled by honest miners, we show that selfish mining strategy strictly outperforms honest mining strategy for a malicious miner with α of stake for any $\alpha > 0$.
- We present a deterministic optimal algorithm for the malicious miner to maximize his relative mining reward for one epoch. Using the algorithm for one epoch, we present a mining strategy that achieves close to optimal block ratios.
- We run simulation experiments to show that relative mining reward is vastly increased by our selfish mining algorithm, compared with previous strategies.

In practice, mining rewards include block rewards and transaction fees. For simplicity, we only consider a fixed block reward.

2. Related Works

2.1. Selfish Mining

Selfish mining is one class of attacks on blockchain consensus protocols. A miner gets a chance to produce a block for a specific block with a probability that is proportional to her hash power in PoW and proportional to her deposited stake in PoS. Ideally, all valid blocks form a chain without forks, and miners receive rewards for contributing their blocks. Therefore, miners should receive block rewards in proportion to their hahs power or deposited stake. However, if there are forks, only one fork, for example the longest fork, can be selected to form the consensused chain. Blocks in discarded forks do not yield block rewards for their owners.

The direct goal of the selfish mining attack is to deliberately exclude blocks proposed by honest miners from the finalized blockchain so that an attacker with less than 50% of hash power or stake can receive more block rewards than they should. In PoW blockchains like Bitcoins, assume honest miners always publish their blocks immediately when they successfully mine new blocks, all messages are immediately delivered to all nodes in the peer-to-peer network without delay and honest miners always extend after the longest observed chain [2,3]. When the current block head is B0, if a malicious miner finds a block B_1, he might deviate from the prescribed protocol by withholding the block B_1 hoping that he can mine the next block B_2 before other honest miners. In one of the lucky situations, the malicious miner owns but withholds B_1 and B_2, while honest miners own B_3 pointed to B_0. Then the malicious miner can publish blocks B_1 and B_1 as a fork ($B_0 \leftarrow B_1 \leftarrow B_2$) longer than ($B_0 \leftarrow B_3$). Honest miners will extend after B_2 instead of B_3 according to the longest chain fork choice rule. In other cases, honest miners get the block B_2 and B_3 before the malicious miner, the malicious miner might give up on B_1 because the fork ($B_0 \leftarrow B_1$) is not likely grow longer than ($B_0 \leftarrow B_2 \leftarrow B_3$). Models this mining game as a Markov Decision Process (MDP) for the malicious miner [3,4]. They show that a miner with 33% of hash power can benefit from selfish mining, i.e., he owns more than 33% of blocks in the finalized chain and consequently receives more than 33% of the total block rewards. Assuming PoS blockchains use a perfect random beacon to select a miner randomly at each slot and, more specifically, miners do not know any additional information about the selection result prior to the slot, extends the MDP analysis of selfish mining to PoS and shows that malicious miners only need to deposit 30.8% ~ 32.5% of the total stake to benefit from selfish mining. In this work, based on the practical usage of random beacons that update the randomness only once per epoch, we show that selfish mining is much easier and more profitable than in PoS blockchains with perfect random beacons that update per slot [5].

Subsequent works on selfish mining address more realistic issues. Discusses the case where block rewards diminishes and transaction fees become the dominant part of mining rewards [8]. Besides making the mining rewards non-uniform among different blocks, shows that selfish miners might fork an existing block that contains many profitable transactions instead of extending after it [8]. Moreover, miners are incentivized to not include the entire remaining transactions to incentivize subsequent miners to extend after their blocks rather than fork their blocks, even if the block capacity is not the bottleneck. Deciding how many transactions increase the action space and state space from discrete to continuous.

Besides, miners with arbitrary stake are incentivized to do some form of strategic mining, so that it is more meaningful to study the equilibrium of all miners, rather than the optimal strategy of single malicious miner. As a result, the analysis becomes highly complicated so that uses simulation rather than rigorous mathematical MDP analysis [8]. A recent work even adopts deep reinforcement learning to study the strategic mining problem [9]. The complicated settings of these works are out of the scope of this work.

2.2. Random Election in PoS

Proof-of-Stake consensus protocols avoid wasting tremendous electricty in computing uesless puzzles in PoW by mimicking the distributed random election of block proposers based on random beacons, or distributed random number generation (RNG). Distributed random number generation should output random numbers that are agreed by the entire nodes, uniformly random, bias-resistant and even unpredictable against a collusion of a subset of nodes. Random beacons are non-trivial and attracted wide research interests in the community especially due to the adoption of Proof-of-Stake consensus protocols in blockchains, such as [10-17].

Most common distributed random beacons are instantiated by distributed protocols among a set of participants where each participant contributes some local randomness independently. In the simplest form, a set of n participants (P_1, P_2,...,P_n) jointly generate a random bit $v \in \{0,1\}$. Each participant Pi chooses her own local random number xi without knowing the choice of other participants. Define the random output to be $v = x_1 \oplus x_2 \oplus \bullet\bullet\bullet \oplus x_n$. It is easy to see that v is a uniformly random bit as long as at least one of the participants chooses their local bits uniformly at random. Moreover, any subset of $\leq n - 1$ participants cannot collaborate to bias the output from uniformly random distribution or guess the output before seeing the other participants' choices. The technical challenge is to implement the protocol by communication in distributed systems and prevent anyone from knowing others' choices before publishing her own choice. In the literature, various cryptographic techniques are adopted to achieve the simultaneous publishing, including commitment schemes, publicly verifiable secret sharing (PVSS) and verifiable delay functions (VDF) [10,11,18]. Commitment schemes and PVSS consist of at least 2 rounds and require significant time for each round to make sure the communication among participants is synchonized, while VDFs must be evaluated for more than synchronization time. In summary, current distributed random beacons require considerable time to generate a fresh random number, much longer than the duration of a slot in proof-of-stake blockchains. Therefore, existing PoS blockchains update the random seed only once per epoch, instead of once per slot.

In each slot t, miner election result is determined by the slot number t, the random seed r of the epoch and optionally metadata of miners. There are typically two cases for the metadata of miners. In the first case, one slot leader is elected uniformly from a known fixed committee of m miners. The solution is use a pseudo-random number generator *prng* and select the $prng(r, t) \pmod m$-th miner in the committee. In the second case, there is no fixed committee and every stake holder with address *addr* can propose a block if $prng(r,t,addr) < \rho$, where ρ is a difficulty parameter to control the expected number of leaders per slot. Since *r,t,addr* are known to any miner at the beginning of the epoch or earlier, the miner knows the election results in future slots of the epoch. Even if more advanced cryptographic primitives are adopted, such as verifiable random functions (VRF) or single secret leader election (SSLE) [4], to keep the election results as secrets to miners, each miner should at least know whether the leader of a slot is herself or not [19,20].

Existing blockchains suffer from low throughput, which results in high transaction fees and limits the widespread application of decentralized technologies. Since PoS blockchains aim at increasing the throughput, they typically use a shorter timeslot to generate a new block. On the other hand, distributed random beacons must have large enough committees to jointly generate the random numbers and use complicated communication protocols to be secure. Concerning the current situations such as the usage of RANDAO in Ethereum 2.0 and the future challenges, it is necessary for security researchers and PoS protocol designers to keep in mind that miners know (partial) election results in the future.

2.3. Mitigations of Selfish Mining

Due to the possibility of selfish mining attacks, especially attacks that can be successfully launched by arbitrarily small miners in the real world, the blockchain mining scenario might be significantly different from what blockchain designers expect and the security of blockchain is severely undermined. Therefore, the community has come up with solutions to mitigate the selfish mining attacks. For example, proposes a novel proof-of-work based solution to disentangle the relationship between the number of blocks in the chain and the amount of mining rewards, by associating mining rewards with fruit blocks that are referred by consensus blocks which form the blockchain [21]. Ethereum 2.0 claims to be immune to selfish mining, which they refer to as avalanche attacks, by requiring honest miners (validators) to ignore block of slot t_1 when they already agreed on a block of slot $t_2 > t_1$, according to Latest Message Driven (LMD) GHOST [6]. However, the security relies on stronger requirements on communication synchrony and user availability. Besides, since Ethereum 2.0 punishes late attestations, honest validators are more vulnerable to attacks against the peer-to-peer networks.

In summary, proof-of-stake protocols are still developing and evolving rapidly. Selfish mining attacks may never be completely resolved because other desired properties might be sacrificed. Therefore, the findings of this work, notably random beacons that are not updated frequently enough leak information about future slot leader election results, should be taken into the account by PoS protocol designers.

3. PoS with KFE Model

In this section, we present a complete specification of a simplified model of PoS blockchain protocols with random beacons updated once per epoch. We call it PoS with KFE (knowledge of future election results).

Miners We assume there are two miners: miner M_1 is malicious, while miner M_2 is honest. M_1 deposited $\alpha (0 < \alpha < 1/2)$ of the total stake and M_2 deposited the rest $1 - \alpha$ of stake. While the blockchain grows, M_1 and M_2 might receive different mining rewards. However, we assume that both M_1 and M_2 do not change their deposited stake. This means α is constant throughout the lifespan of the blockchain. We will define the behavior of these two miners later.

Transactions We ignore transactions in our model, because we do not consider the effect of transaction fee rewards or the attack of censoring particular transactions.

Timing Blockchain is a dynamic system. Our model use a discrete time system and use slot as the basic time unit. Blockchain starts from slot 0 and extends infinitely. We also define epoch as T slots, so that slots $(k-1)T+1$ to kT form the k-th epoch for $k \in \{1,2,...\}$. In each slot, we select one of the miners to be the leader of the slot.

Leader of slot t can propose a block for the slot. Every epoch k uses a different random seed r_k for leader selection that is unpredictable by either miner in previous epochs and known to both miners from the beginning of the epoch k.

Blocks In each slot t, the slot leader can create and own a block B_t. A valid block should specify its predecessor, which is a previous valid block $B_{t'}$, $t' < t$. Once a block becomes part of the finalized blockchain, its owner receives a fixed amount of reward R. The honest miner M_2 always creates only one block for one slot when she is the leader and immediately publishes the block. The malicious miner M_1 might create multiple blocks for one slot, by pointing to different predecessors, and might withhold these blocks and publish one of them later. M_1 cannot publish ≥ 2 different blocks for one slot, because M_2 can detect this dishonesty and punish M_1 severely. There is a genesis block B_0 at slot 0 that does not belong to M_1 or M_2 but agreed by both miners as the first block.

Communication Since we exclude transactions, it suffices to consider that M_1 and M_2 have a communication channel between each other. We assume M_1 and M_2 can send arbitrarily many messages through the channel each other and the messages are delivered to the recipient immediately. This assumption is rather an oversimplification, especially because selfish mining such as maliciously withholding blocks can be detected by honest miners when the communication is perfect. We remark that if we consider the setting that the communication might be delayed longer than a slot, the total stake ratio of multiple honest miners is effectively discounted because they create forks.

Forks, Views and Blockchains Ideally, blocks form a chain ($B_0 \leftarrow B_1 \leftarrow B_2 \rightarrow \bullet\bullet\bullet \rightarrow B_t$) after slot t. However, since M1 might withhold his blocks and point to arbitrary predecessors, blocks might form forks and M_1 and M_2 might have different views of the forks. For example, if M_1 owns and withholds blocks $B_1(\rightarrow B_0)$ and $B_2(\rightarrow B_1)$, M_2 owns and publishes block $B_3(\rightarrow B_0)$, then the view of M_1 is two forks ($B_0 \leftarrow B_1 \leftarrow B_2$) and ($B_0 \leftarrow B_3$), while the view of M_2 is one fork (chain) ($B_0 \leftarrow B_3$). The consensus blockchain after slot t is defined as the longest fork, which might be different for M_1 and M_2. We can simplify the views. M_2 always extends after the longest fork, so she only keeps the longest fork in her view and ignores other forks. M_1's view can be compactly represented by M_2's view and slots of M_2 where M_2 has not published a block. In our example, M_2's view is ($B_0 \leftarrow B_3$,{1,2}). We define depth of a block as the length of the fork ending at the block.

Reward and Payoff The longest fork in M_2's view is considered as the consensus chain, denoted as $chain_t$. After slot t, the reward of M_i, $REW_{i,t}(chain_t)$, is defined as the number of blocks of M_i in the consensus chain $chain_t$. Since the goal of selfish mining is to maximize the ratio of REW_1 versus REW_2, we define the payoff of M_1 as $\rho_t(chain_t, \lambda) = REW_{1,t}(chain_t)\bullet(1-\lambda) - REW_{1,t}(chain_t)\bullet\lambda$. λ is introduced as inspired by [19,10] to facilitate aggregating the payoffs of different slot intervals. λ is closely related to the proportion of block rewards received by M_1. If M_1 owns λ of the blocks in chaint (excluding the genesis block), then $\rho_t = 0$.

Strategies M_2 always uses the simple honest strategy so it suffices to only consider the strategy of M_1. Suppose M_1 uses strategy π, that at the beginning of slot t, given his view at t − 1, (chain$'_t$, slots$_{t-1}$ = s_1, s_2,...), according to whether he is the leader of slot t and future slots in the current epoch, chooses his action. When M_2 is the slot leader, M_2 publishes her block B_t before M_1 chooses his action. chain$'_t$ refers to the longest chain after M_2 publishes her latest block B_t. If M_1 is the slot leader, he adds slot t to his state, updates slot$_{t-1}$ to $slot'_t = slot_{t_1} \cup \{t\}$.

We call the set $slot_t$ as the available slots. The valid actions of M_1 is to choose a subset pub_t of the available slots to publish blocks. For each chosen slot, M_1 publishes one block and specifies its predecessor block. In the end of slot t, the set of available slots becomes $slot_t = slot'_t \,/\, pub_t$.

Runs and Randomness We define a run to be a particular execution path, consisting of the view of M_1 at every slot, determined by random seeds {rk} at every epoch and the strategy π of M_1. We define a variable et for every slot $t \geq 1$ to represent the selected leader for slot t. If et = 0, M_1 is the leader of slot t. Otherwise et = 1 and M_2 is the leader. The leader election result of epoch k is represented by bit sequence $e_k = e(k-1)T+1e(k-1)T+2 \bullet\bullet\bullet e_{kT}$, determined by random seed r_k. Assuming rk is a uniformly drawn integer from a large range, ek is uniformly distributed in $\{0,1\}^T$. r_k is independent from any $k' \neq k$. With a bit abuse of notation, we refer to the payoff $\rho_t(chain_t, \lambda)$ as $\rho_t(e_1 e_2 \bullet\bullet\bullet e_t, \pi, \lambda)$, because chaint is determined by the election results and strategy of M_1. Previous works uses MDP analysis and measures the expected payoff of a strategy π over all randomness used in slot leader election [4,5]. Our work also measures the expected payoff over leader election. If we consider the selfish mining game within an epoch, for simplicity the first epoch, and want to maximize the payoff at the end of the epoch, then the problem becomes an offline algorithm so that we can find a deterministic optimal algorithm.

4. Optimal Strategies

This section first presents an optimal strategy for M_1 to maximize his payoff in the first epoch. We recall the problem in subsection 4.1 present the strategy in subsection 4.2 and prove its optimality in subsection 4.3. If slots at the boundary serve as checkpoints, so that honest miners ignore blocks of epoch $k' < k$ received after epoch k starts and the malicious miner cannot withhold blocks across epochs, the optimal strategy for the first epoch can be repeated for every epoch and remains the optimal strategy.

4.1. Problem Formulation for the First Epoch

We use the model in section 3 and only consider the first epoch. For a run of the mining game, suppose the slot election result is the bit sequence $e_1 = e_1 e_2 \bullet\bullet\bullet e_T$.

Let $\eta = 1 - (\sum_{i=1}^{T} e_i)/T$ be the ratio of M_1's slots. The election result is known to both M_1 and M_2 at the beginning of slot 1. We will present an optimal strategy π of M_1 to maximize $\rho_T(e_1,\pi,\lambda) = (\#$ blocks of M_1 in $chain_T) \bullet (1-\lambda) - (\#$ blocks of M_2 in $chain_T) \bullet \lambda$. We reduce the maximization problem to a interval scheduling problem and present a $\Theta(T^2)$ time optimal algorithm.

Payoff of Honesty If M_1 uses the honest strategy πh, then the chainT is a chain of $T+1$ blocks ($B_0 \leftarrow B_1 \leftarrow B_2 \bullet\bullet\bullet \leftarrow B_T$), where B_0 is the genesis block and $B_i (1 \le i \le T)$ is owned by M_1 if $e_i = 0$, otherwise owned by M_2. The payoff is

$$\rho T(\mathbf{e}1, \pi h) = \left(\sum_{i=1}^{T}(1-e_i)\right).(1-\lambda) - \left(\sum_{i=1}^{T} e_i\right).\lambda = T\,(\eta - \lambda) \tag{1}$$

Using our optimal strategy π_s, M_1 always receives payoff no less than $T(\eta - \lambda)$.

4.2. Optimal Strategy for the First Epoch

Extremely Lucky Case ($\eta > 1/2$) In an extremely lucky case, M_1 owns more slots than M_2 in the first epoch. In this case, M_1 can simply withhold all of his blocks until near the end of slot T. M_2 does not know blocks of M_1 so she grows a chain consisting of her own slots. Before the slot T ends, M_1 publishes his fork using all of his slots. Since the fork of M_1 is longer than the view of M_2, M_2 gives up her old view and agrees on the new fork that only includes blocks of M_1. This simple strategy is obviously optimal and achieves payoff $T\eta(1-\lambda)$ for M_1. This lucky case happens with only a low probability, because $\alpha < 1/2$, and admits a simple optimal strategy. In the following discussion, we can focus on the more complicated case when $\eta \le 1/2$.

Fork Attack Firstly we discuss the condition for the selfish miner M_1 to win the fork competition and exclude one fork produced by M_2. An important observation is that when M_1 publishs a fork **fork** = $(\bullet\bullet\bullet B_c \leftarrow) B_{s1} \leftarrow B_{s2} \bullet\bullet\bullet B_{sg}$ that diverges from the view of M_2 $view_{old} = (\bullet\bullet\bullet B_c \leftarrow) B_{h1} \leftarrow B_{h2} \bullet\bullet\bullet B_{hf}$ after the common block B_c and longer than the view of M_2 by at least 1, M_2 will give up her old view. This is also the necessary condition for successfully excluding an honest fork. If M_1 publishes a fork that is the same long or shorter than M_2's view, M_2 will ignore the fork and continue extending after her own view. The two sets of slots $S = \{s_1, s_2, \ldots, s_g\}$ and $H = \{h_1, h_2, \ldots, h_f\}$ are disjoint because B_c is defined as the last common block. Under strategy π_s, slots in S are all controlled by M_1 and slots in H are all owned by M_2.

Interval Attack π_s only considers attacks such that $S \cup H$ forms an interval of slots $I = [t_1 = \min\{s_1, h_1\}, t^2 = \max\{s_g, h_f\}]$. Therefore, we call such a fork attack as an interval attack. When interval attack is successful in I, we call I a valid interval. Note that a valid interval consists of at least 3 slots, because the trivial case of 1 M_1's slot does not exclude any honest block. We imformally justify why this restriction does not lose optimality. Suppose there exists a slot t in the interval I but is not in $S \cup H$.

- If the leader of t is the honest miner M_2, then M_2 should have mined a block B_t and included it in her old view before M_1 publishes his fork.
- If the leader of t is M_1, he can use another set of slots $S' = (S \cup \{t\}) \backslash \{s_g\}$ to form another fork of the same length but saves the slot s_g for future attacks. In contrary, if M_1 does not use slot t in the fork, he can never use the slot t in future attacks without the cost of excluding his own slot s_g.
- If $t > s_g$, this means M_1 has additional slots so that he can wait until M_2 catches up and exclude more blocks of M_2.

Multiple Interval Attacks Under strategy π_s, M_1 might perform multiple interval attacks that do not intersect each other. If M_1 chooses intervals $I_1, I_2, \ldots, I_d$ (the intervals are sorted so that the largest slot of I_i is smaller than the smallest slot of I_{i+1}), M_1's full mining strategy is the following

- M_1 acts honestly when the current slot t is not in any of these intervals, i.e., he does not withhold any block or form forks.
- During slots $[t_{i1}, t_{i2} - 1]$ in a slot interval $I_i = [t_{i1}, t_{i2}]$, M_1 withholds his slots and does not publish any block.
- At the last slot t_{i2} of a slot interval $I_i = [t_{i1}, t_{i2}]$, M_1 waits until M_2 publishes her block if she is the leader of t_{i2}, then M_1 publishes a fork consisting of all of his slots in the interval I_i and connects the fork to the last common block right before I_i.

Under π_s which specifies intervals $I_1, I_2, \text{dots}, I_d$ for M_1, the final chain $chain_T$ consists of slots

$$(\{1,2,\ldots,T\}(I_1 \cup I_2 \ldots I_d)) \cup \left(\cup_{i=1}^{d}\{t \in I_d | e_t = 0\}\right) \tag{2}$$

All slots of M_1 are included in $chain_T$. In each interval, all slots of M1 are included in $chain_T$ while slots of M_2 are all excluded. The goal of M_1 is to exclude as many M_2's slots as possible.

Interval Scheduling π_s chooses one subset of non-intersecting intervals from the set of all valid intervals that maximizes the number of honest blocks excluded in these intervals. This is a weighted interval scheduling algorithms and has an efficient algorithm of $\Theta(T + |I'|)$ time complexity, where T is the range of time units and I′ is the set of all valid intervals. Note that for valid intervals are slightly modified from $I = [t_1, t_2]$ to $I' = (t_1 - 1, t_2]$ before running an interval scheduling algorithm.

Generating Valid Intervals I′ For T slots, there are $O(T^2)$ valid intervals. We can record all the valid intervals in $\Theta(T^2)$ time using the following steps:

1. Construct an array Diff such that $\text{Diff}[i] = \sum_{j=1}^{i}((1-e_i) - e_i)$ for each $i \in \{1,2,\ldots,T\}$. Diff[0] = 0. Diff[i] indicates the difference between the number of M_1's slots and M_2's slots. Diff can be computed in $\Theta(T)$ time.
2. Initiate intervals to be an empty list. For each $j \in \{3,4,\ldots,T\}$, creates a list intervals[j] that includes all the intervals ending at j.

For each $i \in \{0,1,...,j-3$, if δ_{ij} = diff $[j]-$diff $[i] \geq 1$, add (i,w_{ij}) to the list intervals$[j]$. w_{ij} is the weight of the interval $(i, j]$, i.e., the number of M_2's slots within the interval $[i+1, j]$. w_{ij} is computed as $(j-i-\delta_{ij})/2$. This step takes $\Theta(T^2)$ total time.

Interval Scheduling Algorithm Given T and intervals as input, an interval scheduling algorithm finds the optimal selection of a subset of non-intersecting intervals that maximizes the total weight. The algorithm uses bottom-up dynamic programming. The optimal total weight for the range $(0,T]$, $W[T]$, is the maximum of different subcases:

– The slot T is not covered by any interval in the optimal solution. In this case, $W[T] = W[T-1]$, the optimal total weight for the subrange $(0,T-1]$.

– The slot T is covered by an interval $I' = (t_1,t_2]$ with weight w in the optimal solution. Firstly, t_2 must be equal to T. If $t_2 < T$, T is not covered by I'. If $t_2 > T$, the interval is not covered by the range $(0,T]$. In this case, the optimal total weight is $W[t_1] + w$.

Optimal Strategy The optimal solution Sol of interval scheduling can be reconstructed from $W[0...T]$ and $Temp[0...T]$ as illustrated in the pseudocode of algorithm 1. π_s use the intervals in *Sol* to instruct M_1 to perform the multiple interval attacks.

```
W[0, 1, ... T] ← 0;
Temp[3, 4, ... T] ← 0;
for j ∈ 3, 4, ..., T do
    W[j] ← W[j − 1];
    Temp[j] ← j;
    for (i, w) ∈ intervals[j] do
        if w + W[i] > W[j] then
            W[j] ← w + W[i];
            Temp[j] ← i;
        end
    end
end
Sol ← [];
j ← T;
while j ≥ 3 do
    if Temp[j] < j then
        Sol ← concat([(Temp[j], j)], Sol);
        j ← Temp[j];
    else
        j ← j − 1;
    end
end
```

Algorithm 1: Interval Scheduling Algorithm

4.3. Optimality Proof

At each slot, M1 can choose to release a subset of his unpublished slots. If the published blocks form a chain that extends after the longest chain in the honest view, both M_1 and M_2 acknowledge the new chain. Otherwise the published blocks form forks w.r.t. the honest view. These blocks might form multiple forks. If all forks are no longer than the honest view, M_2 ignores the new forks and continue with her old view. If some forks are longer than M_2's old view, M_2 changes to extend after the longest fork (breaking ties arbitrarily when there are multiple longest forks) and ignores other forks. Therefore, in the view of M_2, M_1 effectively only publishes exactly one fork in each fork attack. M_1 might try to extend after shorter forks, as long as he does not publish two different blocks for one slot.

We use the following propositions on fork attacks to show that an optimal strategy exists after we prune the strategy space and reduce the selfish mining problem to an interval scheduling problem.

Proposition 1. *M_2 never mines two different blocks (of different slots) with the same depth. Moreover, M_2 mines blocks with strictly increasing depth. Immediately after a successful fork attack, the next honest block is deeper than the previous honest block by at least 2 because the head of the new fork is deeper than the previous honest block by at least 1. In other cases, M_2 extends after the longest chain which is at least as deep as her previous block.*

Next we want to make a proposition that an optimal strategy will not jump between two conflicting forks back-and-forth. In other words, after M_1 successfully excludes some blocks in a fork attack, he will never extend after those discarded blocks. To formally analyze this proposition, we define conflicting forks precisely first. Recall that a fork is a chain from the genesis block to the most recent block and a slot has at most one block. A fork can be uniquely represented as **fork**(t) for the slot t. Two forks **fork**(t_1) and **fork**(t_2) are conflicting forks if neither fork is a prefix of the other.

Proposition 2. *In an optimal strategy, there cannot be three different views of* M_2, **fork**(t_1), **fork**(t_2) *and* **fork**(t_3), ($t_1 < t_2 < t_3$) *such that* **fork**(t_2) *conflicts with* **fork**(t_1) *and* **fork**(t_3) *conflicts with* **fork**(t_2), *but* **fork**(t_1) *is a prefix of* **fork**(t_3).

Proof of Proposition 2 Suppose proposition 2 does not hold. Choose one counterexample that firstly maximizes t_1 then maximizes t_3: (t_1,t_2,t_3). t_2 is ignored because there can be multiple conflicting forks between t_1 and t_3. Suppose the complete fork transitions are **fork**(t_1) → **fork**(u_1) → **fork**(u_2)••• → **fork**(u_{m-1}) → **fork**(t_3) → **fork**(v_1) → **fork**(v_2) → ••• → **fork**(v_{n-1}) → **fork**(T). M_1 performs one valid fork attack to realize each transition. We show that the $m \geq 1$ attacks between t_1 and t_3 should not exist in the optimal strategy. We observe that |**fork**(t_3)| − |**fork**(t_1)| ≥ m + m_h, where $m_h = \sum_{i=t_1+1}^{t_3} e_i$ is the number of honest blocks during the period (t_1,t_3]. Besides, **fork**(v_1) conflicts with both **fork**(t_1) and **fork**(t_3) because we choose the counterexample that maximizes t_3 given t_1. This means |**fork**(v_1)| > |**fork**(t_3)| and **fork**(v_1) does not use blocks in **fork**(t_3) \ **fork**(t_1). Besides, note that the forks **fork**(u_i) are never used after t_3, because we choose the counterexample of the maxmized t_1. This means forks after t_3 do not contain blocks published between (t_1,t_3]. If M_1 does not publish any block during (t_1,t_3], then the new **fork**(t_3)′ extends **fork**(t_1) with mh honest blocks and M_1 saves at least m blocks of himself for the future that improves his payoff. All fork transitions after t_3 are still valid. Therefore, the strategy that jumps back-and-forth is not optimal.

Now that M_1 will not extend after discarded forks, he can only extend after and attack the most recent fork. We further make one proposition about the fork attacks.

Proposition 3. *In an optimal strategy, after* **fork**(t_2) *beats a conflicting fork* **fork**(t_1) *in a fork attack at the common slot* t_{12} = *max* **fork**(t_1) ∩ **fork**(t_2)), *subsequent fork attacks with* **fork**(t_3) *should not intersect with* **fork**(t_2) *at a slot* t_{23} *earlier than* t_{12}. *Moreover,* t_{23} *should not be earlier than* t_2.

Proof of Proposition 3 If $t_{23} < t_{12}$, M_1 can improve his payoff by not publishing the fork **fork**(t_2) at all. M_2 might publish m new blocks duing (t_2,t_3]. Without **fork**(t_2), M_2 appends these m blocks after **fork**(t_1). Since |**fork**(t_3)| ≥ |**fork**(t_2)|+m+1 and |**fork**(t_2)| ≥ |**fork**(t_1)|+1, we know |**fork**(t_3)| ≥ |**fork**(t_1)|+ m + 2, which means M_1 can still successfully attack **fork**(t_1) appended with m honest blocks using **fork**(t_3). M_1 can even withhold (1+|**fork**(t_2)|− *depth*(t_{12})) more blocks at slot t_3. Moreover, if $t_{23} \in (t_{12},t_2)$, M_1 would better intersect **fork**(t_2) at t_2, to make sure slots in (t_{23},t_2] to be included in the final chain and use fewer blocks to form **fork**(t_3).

Fork Attacks and Intervals Proposition 2 and 3 show that the slots effected (published by M_1 or excluded from the old forks) in two slot attacks have no intersection. It is also easy to see that, in each slot attack, the set of excluded honest slots form a continuous interval (ignoring M_1' slots in between). We can also show that M_1's slots affected by a slot attack form a continuous interval (ignoring M_2' slots in between), by requiring M_1 to always use the earliest available slots first. Now we want to show that M_1's slots and M_1's slots affected by one fork attack form a continuous slot interval, i.e., $H \cup S$ is a slot interval.

Proposition 4. *In an optimal strategy, if* M_1 *performs a fork attack such that the honest view changes from* **fork**(t_1) *to* **fork**(t_2). *Let* C *be the set of common slots of* **fork**(t_1) *and* **fork**(t_2), H *be the set of slots in* **fork**(t_1) *but not in* **fork**(t_2), S *be the set of slots in* **fork**(t_2) *but not in* **fork**(t_1). *Then* $S \cup H = \{min(S \cup H),min(S \cup H) + 1,...,max(S \cup H)\}$. *In other words,* $S \cup H$ *is an interval of slots.*

Proof of Proposition 4 Suppose $S \cup H \neq I(= \{\min(S \cup H),\min(S \cup H) + 1,...,\max(S \cup H)\})$. There exists a slot $t \in I \setminus (S \cup H)$.

- t is owned by M_2: (1) $\min(S) < t < \min(H)$. If slot t is excluded by a previous fork attack, this means M_1 had at least one extra available slot $\min(S)$ when making the previous attack. We will explain how M_1 can extend the previous attack soon. If slot t is not excluded, M_1 can shift the range of the fork attack to exclude $(H \cup \{t\}) \setminus \{\max(H)\}$. (2) $\min(H) < t < \max(H)$. **fork**(t) conflicts with **fork**(min(H)) and **fork**(max(H)) conflicts with **fork**(t). This violates the proposition 2. (3) $\max(H) < t < \max(S)$. The fork attack happens after slot max(S), but it does not exclude M_1's block Bt at slot t in the attack. If B_t was excluded in previous attacks, according to proposition 3, M_1 should not exclude slots in H which are earlier than slot t. If B_t was not excluded, the success of attack means **fork**(t) conflicts with **fork**(max(H)) and M_1 has made a previous attack to make M_2 switch to **fork**(t). By proposition 3, M_1 will not try to exclude H again.
- t is owned by M_1: (1) $t < \max(S)$. M_1 can change the strategy without losing optimality by using $(S \cup \{t\}) \setminus \{\max(S)\}$ to perform the attack. (2) $\max(S) < t < \max(H)$. M_1 can postpone the current attack according to interval extension to be introduced soon after.

Interval Extension The case (a) t is owned by M_2, $t < \min(H)$ and t is excluded by previous attack, or (b) t is owned by M_1 and $t > \max(H)$, can be avoided by interval extension. Case-a is a result of case-b in the previous attack, so we only discuss how case-b can be avoided without losing optimality. Suppose in the next attack, slots in H' are excluded, replaced by slots in S'. We observe that the next attack can be firstly shifted so that there are no honest slots between H and H', then shifted again so that there are no M_1's slots between S and S'. Therefore, these two attacks can be merged into one fork attack denoted by $(\bar{S} = S \cup S', \bar{H} = H \cup H')$. Moreover, since $|S| \geq |H| + 1$ and $|S'| \geq |H'| + 1$, $|\bar{S}|$ has at least two more slots than $|\bar{H}|$. M_1 can withhold the slot $\max(S')$ to exclude

one more honest block in future attacks.

With proposition 4, a slot attack can be represented by an interval in an optimal strategy. Moreover, M_1 cannot perform two attacks on two intervals with non-empty intersection, because he will not exclude an honest slot twice according to proposition 3 and cannot use his own slot twice. So far we have proved that an optimal strategy exists such that M_1 performs attacks in a few non-intersecting intervals. Besides these intervals, M_1 publish blocks honestly.

4.4. Global Strategy for Multiple Epochs

Since M_1 does not know the future miner election result of the next epoch, he cannot extend the above optimal strategy for the first epoch to future epochs directly. Now we present how should miner M_1 act at the intersection of two epochs, according to three different cases for the first epoch:

• In the first epoch, M_1 owns more slots than M_2, so that M_1 can exclude all blocks of M_2 by presenting a fork of length exactly large than the number of M_2's slots by one. If M_1 has s remaining slots, he can withhold these slots until the next epoch starts. At the beginning of the next epoch, the state of the game is equivalent to the case that the next epoch has $T + s$ slots and the first s slots are all controlled by miner M_1. Then M_1 can use the optimal strategy for single epoch.

• In the first epoch, M_1 cannot exclude all blocks of M_2. If the first epoch ends with a winning interval for M_1, he should not have any remaining hidden slots and he wins the interval with exactly one more block. This means he wastes at most one block compared to the case that he knows all information of both epochs at the beginning of the first epoch.

• M_1 cannot exclude all blocks of M_2 and the first epoch does not end with a winning interval. In this case M_1 should have already published all his blocks as a honest player by the end of the first epoch. If M_1 is extremely lucky to own more slots in the second epoch, then he could have used the extra slots from the second epch to exclude M_2's blocks in the first epoch if M_1 knows all miner election results at the beginning of the first epoch. In this unlikely case, M_1 may waste up to 1 slot in the optimal strategy. Otherwise, M_1 does not waste any slot compared to the global optimal strategy.

Overall, in the worst case, for a period of n epochs (nT slots in total), the above global strategy is worse than the global optimal strategy by excluding at most n slots of M_2. In most cases, the gap is even much smaller. This global strategy ensures that every block of M_1 is included in the chain, while the strategies proposed by previous works risk losing some blocks of M_1.

5. Experiments

In this section, we use random samples to estimate how much payoff gain is achieved by the optimal strategy in section 4. We also compare it with the MDP strategy in that does not use the information of future leader election [5].

MDP Strategy The MDP strategy is the following:

– The original state is (0,0), when M_1 has no withholding block and agrees to extend after the same fork as M_2. From state (0,0), if M_1 gets the next slot, M_1 withholds it and the state becomes (1,0). Otherwise M_2 publishes a block and M_1 should accept it, so the state remains (0,0).

– At state (1,0), if M_1 mines the next block, it transits to state (2,0). Otherwise it transits to (1,1). M_1 waits in both cases.

– At state (1,1), if M_1 mines the next block, he publishes two blocks to exclude M_2's block and the state returns to (0,0). Otherwise M_2 publishes a block and the state goes to (1,2).

– At state (1,2), if M_1 mines the next block, it goes to state (2,2). Otherwise M_2 mines one more block. M_1 gives up his block and accepts the three blocks of M_2. The state returns to (0,0).

– At state (2,2), if M_1 mines the next block, he publishes all 3 blocks and successfully exclude 2 blocks of M_2. The state returns to (0,0). Otherwise M_2 gets the next block. Now the latest 5 slots belong to (M_1,M_2,M_2,M_1,M_2) respectively. M_1 gives up the first slot but still holds the fourth slot. The state transits to (1,1).

– At state (2,0), M_1 waits until M_2 catches up. When M_1 has exactly 1 more block than M_2, M_1 publishes all of his blocks and successfully exclude all blocks of M_2. The state returns to (0,0).

Experiment Parameters We consider a game with 10 epochs, where each epoch consists of 100 slots. We test the performance of our optimal strategy and the MDP strategy for M_1 with different stake ratios. The column η represents the stake ratio of M_1. The second, third and fourth columns represent the ratios of blocks of M_1 included in the blockchain if M_1 uses our optimal strategy for each epoch, the global optimal strategy and the MDP strategy introduced above, respectively.

Result The experiments show that our strategy significantly outperforms the MDP strategy. Our strategy is also very close to the global optimal strategy, especially in the usual case when a miner controls much smaller than 50% of the stake.

η	Ours	Optimal	MDP
0.05	0.0503	0.0503	0.0105
0.10	0.1019	0.1020	0.0360
0.15	0.1566	0.1566	0.0606
0.20	0.2174	0.2179	0.1311
0.25	0.2838	0.2864	0.1929
0.30	0.3736	0.3736	0.2783
0.32	0.4066	0.4092	0.3140

0.34	0.4450	0.4480	0.3329
0.36	0.4952	0.4986	0.4029
0.38	0.5413	0.5452	0.4385
0.40	0.5988	0.6061	0.4868
0.43	0.6891	0.6969	0.5508
0.46	0.7850	0.8214	0.6959
0.48	0.8759	0.9006	0.7613

Note that the MDP strategy only allows a miner with more than 32% of the total stake to benefit from strategic mining. To increase the relative revenue by 10%, approximately 36% of stake should be hold by the miner. In practice almost no miner or mining pool has such amount of stake. Using our strategy, a miner with 22% stake can already increase his revenue by 10% via strategic mining.

6. Conclusion

This paper is the first to discuss the effect of real-world distributed random beacons on the blockchain mining games. Previous works usually assume that the random seed to select validators is refreshed in every slot, but this paper points out that this is not the case in real-world PoS blockchains such as Ethereum and Cardano.

Fixing the randomness for an epoch of multiple slots allow blockchain nodes to reach consensus on the randomness, but also allows selfish miners to take advantage of the knowledge about future validator election result. Specifically, this paper presents a close-to-optimal block mining strategy that allows any miner to gain more block rewards in proportion, through an efficient interval scheduling algorithm. We mathematically show the optimality of our strategy for a single epoch and also evaluate its concrete performance using simulation experiments.

The findings of this work urge the community to improve distributed random beacon protocols and especially reduce the latency and round complexity. As for future directions, researchers should take into consideration the current random beacons in blockchain protocol design and security analysis.

References

1. Kiayias, A., Russell, A., David, B., & Oliynykov, R. (2017, July). Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference* (pp. 357-388). Cham: Springer International Publishing.
2. Eyal, I., & Sirer, E. G. (2018). Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM, 61*(7), 95-102.
3. Kiayias, A., Koutsoupias, E., Kyropoulou, M., & Tselekounis, Y. (2016, July). Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation* (pp. 365-382).
4. Sapirshtein, A., Sompolinsky, Y., & Zohar, A. (2016, February). Optimal selfish mining strategies in bitcoin. In *International conference on financial cryptography and data security* (pp. 515-532). Berlin, Heidelberg: Springer Berlin Heidelberg.
5. Ferreira, M. V., & Weinberg, S. M. (2021, July). Proof-of-stake mining games with perfect randomness. In *Proceedings of the 22nd ACM Conference on Economics and Computation* (pp. 433-453).
6. Proof-of-stake (PoS). (2025). Ethereum.
7. (re)introduction to Cardano. (2024).
8. Carlsten, M., Kalodner, H., Weinberg, S. M., & Narayanan, A. (2016, October). On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (pp. 154-167).
9. Bar-Zur, R., Abu-Hanna, A., Eyal, I., & Tamar, A. (2022, June). WeRLman: to tackle whale (transactions), go deep (RL). In *Proceedings of the 15th ACM International Conference on Systems and Storage* (pp. 148-148).
10. RANDAO. (2019).
11. Syta, E., Jovanovic, P., Kogias, E. K., Gailly, N., Gasser, L., Khoffi, I., ... & Ford, B. (2017, May). Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)* (pp. 444-460). Ieee.
12. Raikwar, M., & Gligoroski, D. (2022, November). Sok: Decentralized randomness beacon protocols. In *Australasian Conference on Information Security and Privacy* (pp. 420-446). Cham: Springer International Publishing.
13. Choi, K., Manoj, A., & Bonneau, J. (2023, May). Sok: Distributed randomness beacons. In *2023 IEEE Symposium on Security and Privacy (SP)* (pp. 75-92). IEEE.
14. Cai, Z., & Goharshady, A. K. (2023, May). Trustless and bias-resistant game-theoretic distributed randomness. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 1-3). IEEE.
15. Cai, Z., & Goharshady, A. (2023, July). Game-theoretic randomness for proof-of-stake. In *The International Conference on Mathematical Research for Blockchain Economy* (pp. 28-47). Cham: Springer Nature Switzerland.
16. Boneh, D., Eskandarian, S., Hanzlik, L., & Greco, N. (2020, October). Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies* (pp. 12-24).
17. Ballweg, J., Cai, Z., & Goharshady, A. K. (2023, December). PureLottery: Fair leader election without decentralized random number generation. In *2023 IEEE International Conference on Blockchain (Blockchain)* (pp. 273-280). IEEE.
18. Lenstra, A. K., & Wesolowski, B. (2015). A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive.

19. Micali, S., Rabin, M., & Vadhan, S. (1999, October). Verifiable random functions. In *40th annual symposium on foundations of computer science* (cat. No. 99CB37039) (pp. 120-130). IEEE.
20. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. (2017, October). Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles* (pp. 51-68).
21. Pass, R., & Shi, E. (2017, July). Fruitchains: A fair blockchain. In *Proceedings of the ACM symposium on principles of distributed computing* (pp. 315-324).