

Forking the RANDAO: Manipulating Ethereum’s Distributed Randomness Beacon

Ábel Nagy
Eötvös Loránd University
Budapest, Hungary
nagyabi@gmail.com

István András Seres
Eötvös Loránd University
Budapest, Hungary
seresistvanandras@gmail.com

János Tapolcai
University of Technology and Economics
Budapest, Hungary
tapolcai@tmit.bme.hu

Bence Ladóczki
University of Technology and Economics
HUN-REN Information Systems Research Group
Budapest, Hungary
ladoczki.bence@vik.bme.hu

Abstract

Proof-of-stake consensus protocols often rely on distributed randomness beacons (DRBs) to generate randomness for leader selection. This work analyses the manipulability of Ethereum’s DRB implementation, *RANDAO*, in its current consensus mechanism. Even with its efficiency, *RANDAO* remains vulnerable to manipulation through the deliberate omission of blocks from the canonical chain. Previous research has shown that economically rational players can withhold blocks – known as a block withholding attack or selfish mixing – when the manipulated *RANDAO* outcome yields greater financial rewards.

We introduce and evaluate a new manipulation strategy, the *RANDAO* forking attack. Unlike block withholding, whereby validators opt to hide a block, this strategy relies on selectively forking out an honest proposer’s block to maximise transaction fee revenues and block rewards. In this paper, we draw attention to the fact that the forking attack is significantly more harmful than selfish mixing for two reasons. Firstly, it exacerbates the unfairness among validators. More importantly, it significantly undermines the reliability of the blockchain for the average user by frequently causing already published blocks to be forked out. By doing so, the attacker can fork the chain without losing slots, and we demonstrate that these are later fully compensated for. Our empirical measurements, investigating such manipulations on Ethereum mainnet, revealed no statistically significant traces of these attacks to date.

CCS Concepts

• Security and privacy → Distributed systems security.

Keywords

Ethereum, blockchain, proof-of-stake, unpredictability, manipulability, distributed randomness beacon, *RANDAO*

ACM Reference Format:

Ábel Nagy, János Tapolcai, István András Seres, and Bence Ladóczki. 2025. Forking the *RANDAO*: Manipulating Ethereum’s Distributed Randomness Beacon. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS ’25)*, October 13–17, 2025, Taipei, Taiwan. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3719027.3744852>

1 Introduction

Randomness is indispensable for (permissionless) consensus [17, 22]. Ethereum [36], the second-largest cryptocurrency by market capitalisation and the largest by transaction volume, pseudo-randomly selects block proposers using cryptographic algorithms and protocols. Back in the days, Ethereum utilised a purely *longest-chain protocol* to reach consensus, a similar construction to the current proof-of-work (PoW) consensus mechanism of Bitcoin [24]. Manipulative attacks against longest-chain protocols have been analysed extensively, and several problems related to predictability were pointed out by prior research efforts [3, 7, 30]. Then, on September 15, 2022, the Merge protocol upgrade took place at block 15537394, and since then, beacon committees voting on the beacon block at each slot via attestations are active. In this *committee-based* consensus protocol, new validators are chosen using a DRB called *RANDAO*, introduced in [38]. While the *RANDAO* is an efficient DRB protocol, it is not robust against strategic manipulations [1, 34].

In Ethereum Proof-of-Stake (PoS), blocks are published at fixed time intervals (*i.e.*, at every 12 seconds), referred to as slots, with 32 slots comprising one epoch. The *RANDAO* mechanism selects validators who publish the blocks in the subsequent epoch on the epoch boundaries (*i.e.*, at every 384 seconds). In each slot, there is only one uniquely determined validator who should publish a block. Should it fail to do so, the slot is marked as missed in the canonical chain. As the output of the randomness beacon depends on these missed slots, this creates an opportunity for an attack called **selfish mixing**. Validators colluding within staking pools might find it profitable to intentionally miss blocks if this influences the *RANDAO* outcome such that, in upcoming epochs, more slots are allocated to validators of the given staking pool. The effect of these missed slots is only predictable before the epoch boundaries at the so-called tail slots.

The first analysis on the manipulability of the current *RANDAO* protocol was conducted in a blog post by Wahrstätter [34]. He



This work is licensed under a Creative Commons Attribution 4.0 International License. *CCS ’25, Taipei, Taiwan*

© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1525-9/2025/10
<https://doi.org/10.1145/3719027.3744852>

examined the selfish mixing attack and demonstrated that major staking entities could have manipulated RANDAO dozens of times since the genesis of Ethereum PoS by strategically publishing or omitting tail slots. Alpturer and Weinberg, applying Markov decision processes (MDP), determined optimal selfish mixing strategies in [1] to maximise the number of blocks proposed. According to previous results, a staking pool controlling 20% of the total stake can gain an additional 0.7% of slots with selfish mixing [1] and when a strategic player mounts a selfish mixing attack roughly every 340th slot will be missed, causing a 0.3% reduction in transaction throughput. This can potentially increase transaction costs due to network congestion [10]. However, such issues alone do not threaten the operation of the blockchain.

We observe that strategic validators, besides selectively withholding blocks, might also fork out honest tail blocks to manipulate the RANDAO outcome. Malicious players can perform short-range forks when they have multiple adversarial slots surrounding honest blocks. The attack is initiated as follows: The adversary privately builds a fork and purposefully ignores the next few blocks proposed by the honest players. Finally, the adversary forks out the “surrounded” honest block(s) by building on the private fork. As a side effect, these forking attacks lead to numerous slots missing from the canonical chain. In general, blockchain consensus mechanisms might try to thwart forking attacks. However, it is usually not a punishable offense because it sometimes happens naturally due to network delay.

In this paper, we introduce a new class of RANDAO manipulation strategies that pose a much greater threat to the average user, called **forking attack**. Forking is already a serious issue, but becomes even more severe when linked to strategic RANDAO manipulations. This study investigates under what circumstances such an attack can be economically rational. First, this manipulation significantly increases the bias in the DRB, potentially doubling the attacker’s gain in extra slots when combined with selfish mixing. What further exacerbates the situation is that it undermines trust in the chain’s ecosystem. By forking blocks, transactions in the discarded blocks are removed, and the attacker can hijack the MEV (maximal extractable value) of that block. As we discuss later, in some epochs, the attacker is incentivised to fork out other validators’ blocks, as this provides short-term profits in addition to more MEV.

To understand the inner workings of these forking attacks, we dive into the Ethereum PoS consensus protocol [26]. The current Ethereum PoS protocol has evolved over the years: the consensus protocol has been revised multiple times as new practical attacks were discovered, challenging its presumed liveness and safety [31]. The LMD-GHOST (Latest Message Driven Greedy Heaviest Observed Sub-Tree) protocol was introduced in [9]. In LMD-GHOST, validators attest to a block in each slot, and the chain with the most accumulated attestations becomes the canonical chain, considering only the latest received messages. To counter block withholding attacks, *proposer boost* has been introduced to artificially increase the weight of a block that has been proposed on time.

We elucidate the details of a forking attack in Figure 1. Note that this attack is possible only when the adversary controls at least 20% of the total stake and when the tail slots are ordered as adversary-honest, with the next epoch starting with an adversarial

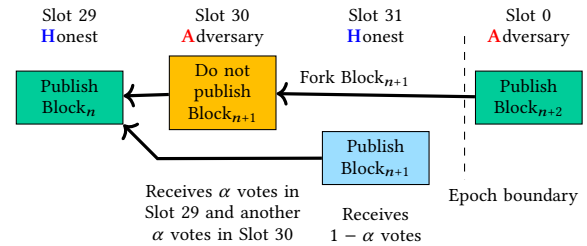


Figure 1: Illustration of an attack where the adversary (\mathcal{A}) is selected to propose blocks in Slot 30 of the current epoch and the first slot of the next epoch, with the final slot assigned to an honest validator. This scenario is called **AH·A forking attack. Assume $\alpha (\geq 0.2)$ denotes the adversary’s voting power, while $1 - \alpha$ denotes the honest voting power.**

The attack proceeds as follows: First, \mathcal{A} secretly builds and votes for its block in Slot 30. This block is kept hidden from honest validators. Hence, they think that the block in Slot 29 is the head of the chain and vote for it. Second, the honest validator publishes a block in Slot 31 and builds it on top of Slot 29. Meanwhile, the validators controlled by \mathcal{A} vote for the adversary’s secret block during Slot 31, pretending to be unaware of the honest block published in Slot 31. Finally, in Slot 0 of the next epoch, \mathcal{A} publishes its two-block fork (illustrated in the first row of the figure), and the honest block in Slot 31 is successfully forked out from the canonical chain [31], thus not contributing to the RANDAO output.

slot (denoted as **AH·A** in our shorthand notation). This is because, in the LMD-GHOST protocol, the adversary’s branch has a weight $2\alpha + p_{\text{boost}}$ (at the time of writing $p_{\text{boost}} = 0.4$) while the honest branch has a weight $1 - \alpha$. If $\alpha \geq 0.2$, the adversarial branch has more attestations, forcing honest validators to choose this branch and attest to the block in Slot 0 as the head of the chain. As a result, the honest block proposed in Slot 31 gets forked out.

Figure 1 highlights key differences between forking and selfish mixing. First, the adversary does not necessarily need to control the tail slot; controlling the beginning of the next epoch may be sufficient. This complicates the attack strategy as it depends on the validators’ actions across two epochs. In other words, when the attacker manipulates the RANDAO outcome to enable further attacks in future epochs, the necessary preparations for two further attacks are being made, accounting for two simultaneous attacks: one at the beginning and another at the end of the new epoch.

This work formalises these RANDAO forking attacks and analyses how much financial gain they lead to by applying these strategic manipulation techniques. Similar to [1], we use an MDP to compute the optimal strategy (or policy) that an attacker should follow to potentially manipulate the RANDAO outcome. This approach maximises the immediate reward and the opportunities for future attack-related rewards in upcoming epochs as well. In the case of forking attacks, the MDP expands considerably compared to scenarios involving only selfish mixing. This is partly due to the larger degrees of freedom with many possible strategies and the fact that many attacks extend across epoch boundaries. As a result,

the states of the MDP extend across two consecutive epochs, resulting in an enormous directed graph. Evaluating this MDP poses a significant computational challenge and proving its optimality remains unattainable.

Our contributions. We provide the following contributions.

- We extend the MDP of [1] for selfish mixing to accommodate forking attacks. Our MDP computes the extent (expected number of proposed blocks) to which a strategic player can bias the RANDAO output in its favour *employing both selfish mixing and forking strategies*. Our RANDAO manipulation results are currently the best known; see Table 2.
- We collect and process traces from the beacon chain to investigate whether RANDAO manipulation attacks occur in the wild. We search for both one- and multi-epoch RANDAO manipulations. We have not yet found statistically significant evidence for multi-epoch manipulations. However, since the protocol currently lacks cryptographic guarantees to prevent such attacks, they are likely to occur in the future, given that validators act as economically rational agents.
- For the sake of reproducibility, we release all our program codes in an open-source repository.¹

The rest of this paper is organised as follows. In Section 2, we describe the pertinent parts of Ethereum’s proof-of-stake protocol. Section 3 describes the RANDAO manipulation strategies we consider in this work. In Section 4, we formalize and compute the advantage of an adversary applying our RANDAO manipulation techniques. In Section 5, the results of our empirical tests on Ethereum mainnet are presented. We discuss countermeasures in Section 6 and review related work in Section 7. We conclude our paper with open problems and future directions in Section 8.

2 Preliminaries and System model

This section introduces the pertinent parts of the Ethereum PoS consensus protocol. We focus on randomness generation and leader selection and omit irrelevant protocol details. A comprehensive overview of the full protocol can be found in [26].

2.1 Notations

Arrays are written in lowercase bold, e.g., \mathbf{v} . We use a Python-like notation for array elements and slices, e.g., $\mathbf{v}[0]$ or $\mathbf{v}[i : j]$, which is non-inclusive on the right, i.e., $\mathbf{v}[j] \notin \mathbf{v}[i : j]$. We denote counterfactual values—those that are reorged or not part of the canonical chain—with a superscript star (*).

For instance, the RANDAO output at epoch e is denoted as R^e . However, due to adversarial bias, it could take on various other counterfactual values, denoted as $R^{e,*}$. Let $X \sim \text{Binom}(N, p)$ denote a binomial distribution with parameters N, p . Typically, we use $(N, p) = (32, \alpha)$. $x \leftarrow S$ denotes sampling x from distribution S . See Table 1 for a summary of the used notations.

2.2 Leader selection in Ethereum

Leader selection in Ethereum PoS is performed using the RANDAO mechanism, outputting a pseudorandom value $R^e \in \{0, 1\}^{256}$ at the end of each epoch e . An epoch consists of 32 slots, we number

the slots from 0 to 31, where slot 31 is called the *tail slot*. In each slot, a single validator, identified by a public key, can propose a block. Each valid proposed block contains a 96-byte value called `randao_reveal`, the BLS signature of the epoch number generated using the proposer’s private key. The contribution is validated against the proposer’s public key during block verification. This information cannot be determined until the block is published unless the proposer’s private key is compromised. Furthermore, note that the BLS signature scheme is deterministic [6]; this prevents the signer from being able to manipulate it (grinding attacks). The final random output is calculated as the bitwise XOR of the hashes of all `randao_reveal` values from each previous block, all the way back to the genesis block. Formally, the RANDAO beacon output at epoch e is calculated as $R^e := R^{e-1} \oplus \left(\bigoplus_{i=0}^{31} h(r_i^e) \right)$, where $h(r_i^e) \in \{0, 1\}^{256}$ is the i th `randao_reveal`’s hash in epoch e . The initial condition is $R^0 := 0^{256}$. Missed blocks are excluded from the calculation, that is, formally a missed block is taken as $h(r_i^e) := 0^{256}$. The RANDAO output R^e defines the proposers’ list² in epoch $e + 2$.³ Specifically, the proposer list is shuffled using a pseudorandom permutation [19], where the permutation seed s is further randomised with the epoch number, i.e., $s := h(e || R^e)$. This ensures that even if all proposers selected in the current epoch are offline (i.e., $R^e = R^{e-1}$), the proposer list differs in the next epoch.

Let \mathbf{v}_R^{e+2} denote the proposer list in epoch $e + 2$ if the RANDAO output in epoch e is R^e , formally, $\mathbf{v}_R^{e+2} = [F_s(\mathbf{v})[j]]_{j=0}^{31}$.

Let us write the *number of blocks* allocated to \mathcal{A} in a certain epoch e for a RANDAO output R^e as:

$$\mathcal{P}(e + 2, R^e, \mathcal{A}) . \quad (1)$$

Observe that $\mathcal{P}(e + 2, \cdot, \mathcal{A})$ is a discrete probability distribution in the RANDAO output R^e of epoch e , with range $0 \leq \mathcal{P}(e + 2, \cdot, \mathcal{A}) \leq 32$. Moreover, $\mathcal{P}(e + 2, \cdot, \mathcal{A}) \sim \text{Binom}(32, \alpha)$ assuming cryptographic building blocks are idealised.

Each slot has a duration of 12 seconds, during which a selected proposer can publish a new block. Due to stringent latency requirements, each block must be published no later than the 4th second of the slot. Otherwise, it is deemed as a “missing” block [18]. If a block is published on time, it receives virtual votes, accounted for by the proposer boost (p_{boost}). At the time of writing, $p_{\text{boost}} = 0.4$, i.e., a newly published block receives virtual votes that equal 40% of the total stake. Note that the proposer boost only exists for the current slot of the block. Blocks published late do not get a boost.

2.3 System model and manipulation objectives

In our model, we assume two parties, an adversarial entity \mathcal{A} with staking power α (i.e., owns α portion of all validators), and an honest entity \mathcal{H} with staking power $(1 - \alpha)$. Based on the RANDAO output R^{e-2} of epoch $e - 2$, \mathbf{v}_R^e defines the list of proposers for epoch e . The proposers controlled by the adversary \mathcal{A} (or strategic player) are denoted by **A**, and the slots assigned to the honest entity \mathcal{H} are denoted as **H**. Using this notation for the slots, we define a string called *chain string* (cs), which is a string that continuously grows

²The current protocol is somewhat more complex, but the details are immaterial to our discussions. Specifically, validator effective balances are also taken into account.

³We use the terms “future epoch” or “next epoch” loosely, even though the current e^{th} epoch’s RANDAO output determines the validators in epoch $e + 2$ in the protocol.

¹ https://github.com/nagyabi/forking_randao_manipulation.

Table 1: Summary of notations used throughout the paper.

Variable	Description
\mathcal{A}, \mathcal{H}	The set of adversarial and honest validators.
α	The adversary \mathcal{A} 's staking power ($0 \leq \alpha < \frac{1}{2}$)
e	Epoch number ($e \in \mathbb{N}$)
r_i^e	Randomness contribution in slot i of epoch e
R^e	The RANDAO beacon's output in epoch e
\mathbf{v}_R^e	The chosen validators in epoch e from R
$\mathbf{A}^t, \mathbf{H}^t$	t consecutive slots assigned to \mathcal{A}/\mathcal{H}
p_{boost}	The proposer boost votes ($p_{\text{boost}} = 0.4$)
$\mathcal{P}(e+2, R^e, \mathcal{A})$	Number of \mathcal{A} 's blocks in epoch $e+2$ for R^e
$\mathcal{S}(e, R^e, \mathcal{A})$	Number of \mathcal{A} 's sacrificed blocks in epoch e

over time. More precisely, at the end of epoch e , we extend cs using a $\{\mathbf{A}, \mathbf{H}\}^{32}$ string based on \mathbf{v}_R^{e+2} , called the $(e+2)^{\text{th}}$ epoch string. When the same character appears again, we typically indicate the number of repetitions in the exponent.

From a RANDAO manipulation perspective, the slots immediately before and after the epoch boundary are the most important. The potential attacks will be defined using these string fragments, which we refer to as *attack string*:

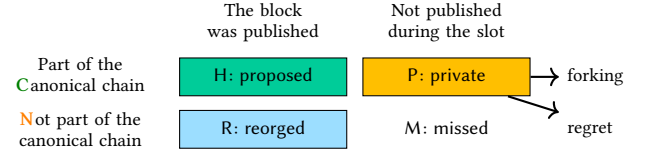
DEFINITION 1 (ATTACK STRING). Let $(m, n) \in \mathbb{N}^2$. The set of attack strings is denoted by $AS_\alpha(m, n)$, where each attack string $as \in \{\mathbf{A}, \mathbf{H}\}^{\leq m} + \dots + \{\mathbf{A}, \mathbf{H}\}^{\leq n}$, where “ \dots ” denotes the epoch boundary separating epoch e and $e+1$. We refer to the substring before/after the epoch boundary as tail(as)/head(as), respectively. $AST_\alpha(m)$ denotes the set of possible tail slots in $AS_\alpha(m, n)$. Additionally, upon reaching the first slot \mathcal{A} can already calculate at least one potential RANDAO outcome $R^{e,*}$. An empty string is denoted by ϵ , and $AS_\alpha(m, n)$ includes an attack string called the honest attack string \mathbf{H} .

Individual slots in an attack string are referred to by their slot number in the subscript, i.e., the tail slot would be \mathbf{H}_{31} in Figure 1. For illustrative examples of attack strings, see Figures 3 to 5.

We assume that the stake distribution remains constant throughout the attack. In our model, \mathcal{A} has exactly α fraction of the voting power in each slot. To be exact, \mathcal{A} has $\frac{\text{Binom}(N/32, \alpha)}{N/32}$ fraction of the votes in each slot but we do not model this variation of the voting power in this work. Considering the law of large numbers ($N \approx 10^6$), this simplification must make sense. An honest entity \mathcal{H} always broadcasts a block when selected. Moreover, honest blocks are always built upon the latest public head of the blockchain. In contrast, the adversary \mathcal{A} may refrain from proposing blocks in an allotted slot. Furthermore, the adversary might strategically fork the blockchain, i.e., propose blocks on blocks that are parents of the current head of the blockchain. We work in a synchronous network model and assume that the upper bound of message delivery is less than one-third of a slot's duration (4 seconds). As noted above, we assume the use of perfect cryptographic building blocks.

We aim to maximise the number of adversarial blocks. We denote the *number of missed tail blocks* in epoch e (sometimes referred to as sacrificed blocks) corresponding to a RANDAO output R^e as

$$\mathcal{S}(e, R^e, \mathcal{A}) . \quad (2)$$

**Figure 2: The four block states in a RANDAO manipulation with their colour encodings in the figures. Slot statuses, (non)canonical, are denoted as N, C.**

An economically rational validator \mathcal{A} applying policy π is incentivised to manipulate the RANDAO output R^e to maximise the expected number of blocks obtained in the future, i.e.:⁴

$$\Gamma_\pi := \lim_{N \rightarrow \infty} \mathbb{E} \left[\frac{1}{N} \sum_{e=1}^N \mathcal{P}(e+2, R^e, \mathcal{A}) - \mathcal{S}(e, R^e, \mathcal{A}) \right] . \quad (3)$$

Γ_π is the most natural RANDAO manipulation objective of a validator, or in other words a policy to maximise the expected number of proposed blocks without incurring significant losses to current rewards, e.g., missing to propose too many blocks. To simplify our reward function Γ_π , we assume a discount factor of $\gamma = 1$; slots in the distant future are as valuable as the ones in the imminent future.

3 RANDAO manipulation strategies

We distinguish between four possible states a block can take in the context of RANDAO manipulation. see Figure 2.

- H: proposed** The validator successfully proposed a valid block accepted by the supermajority of the validators (i.e., 2/3) into the canonical chain. Honest validators always exhibit this behaviour in our analysis, while adversarial validators may consider additional options, such as withholding blocks.
- R: reorged** The validator proposed a valid block; however, it ended up on a non-canonical branch of the blockchain. It is no longer considered canonical by the supermajority of the validators' stake, e.g., Slot 31 in Figure 1.
- M: missed** A validator fails to publish a block in its designated slot. For an honest validator, this is typically caused by connectivity or other operational issues (e.g., resource-constrained validator and large computational overhead of validating attestations).
- P: private** The validator builds a block but does not publish it on time during its allotted slot. An adversarial validator then sends the block only to the validators within its staking pool. Later, the private block either becomes part of the canonical chain by *forking* the next block (a strategy known as an ex-ante reorg attack) or, depending on the RANDAO outcome, the attacker might decide not to publish the block at all; an action we refer to as *regret*, see Figure 4.

Block statuses are denoted as $H_i^e, R_i^e, M_i^e, P_i^e$ indicating that the block in the i th slot in epoch e was proposed, reorged, missed, built privately, respectively. As indicated in Figure 2, reorged and missed blocks *do not contribute to the RANDAO output* R^e since they are

⁴We define the parameter of the reward functions, i.e., our MDP policy in Section 4.1.

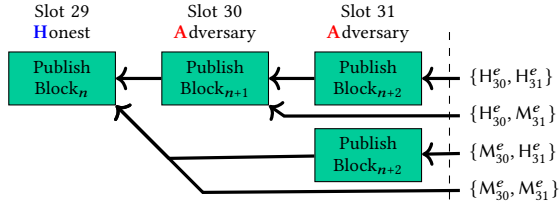


Figure 3: Decision tree for the selfish-mixing with attack string A^2 . \mathcal{A} computes R^e for all four scenarios and selects the most lucrative one.

not part of the canonical chain. In general, RANDAO manipulation strategies can take advantage of this fact. This study examines two strategies (selfish mixing and forking) giving power to the adversary to manipulate the RANDAO output.

3.1 Selfish mixing

As well known, the adversary can selectively propose or miss blocks to manipulate the RANDAO output [34]. Assume that \mathcal{A} is assigned with t consecutive tail blocks, formally A^t , of epoch e , then \mathcal{A} can choose arbitrarily between 2^t RANDAO outputs by missing or proposing each tail block. Thus it is trivial, that $A^t \in AS_\alpha(m, n)$ for $0 \leq t \leq m$, as \mathcal{A} can compute R^e corresponding to C^t . The manipulative power for $t = 2$ is the decision tree shown in Figure 3, e.g., the adversary chooses option $\{H_{30}^e, M_{31}^e\}$ if the calculated R^e eventually leads to the highest number of blocks. In this case, sacrificing Slot 31 is worthwhile, as it results in a significantly higher number of blocks in epoch $e + 2$. When evaluating selfish mixing, the adversary has all the necessary information to make decisions. This is not the case for forking, as we will see next, leading to more complex decision trees and evaluations.

3.2 Ex-ante reorging honest blocks

We argue that a RANDAO manipulator can selectively reorg out honest blocks from the canonical chain to influence the output of the randomness beacon. The attack strategy we propose here may involve more sophisticated behaviour than merely missing tail slots. The simplest forking manipulation is illustrated in Figure 4 when \mathcal{A} observes the attack string $AH \cdot A$. Note that this is the same attack described in Figure 1. In this scenario, \mathcal{A} can ex-ante fork out the tail slot H if $\alpha \geq 0.2$. \mathcal{A} must decide to fork out H before seeing its `randao_reveal` r_{31}^e . After seeing r_{31}^e , \mathcal{A} can reconsider its forking plans. Specifically, \mathcal{A} can decide in the first slot of the next epoch to build A_0 on top of either A_{30} (i.e., finalise the ex-ante reorg), or on top of H_{31} incurring the sacrifice of A_{30} . Note that \mathcal{A} has no incentive to withhold A_0 as it has got no manipulative power.

\mathcal{A} must make a decision at Slot 30, relying solely on the RANDAO output corresponding to $\{P_{30}^e, R_{31}^e\}$. The adversary’s strategy involves using stochastic methods to approximate the unknown RANDAO outcome and comparing these approximates with known values. In other words, if $\{P_{30}^e, R_{31}^e\}$ results in a sufficiently large revenue, \mathcal{A} will privately build the block in its allotted adversarial slot (Slot 30); otherwise, \mathcal{A} should opt to propose the block. The

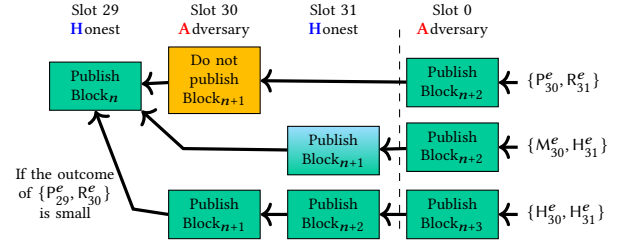


Figure 4: Decision tree for the forking RANDAO manipulation, see Section 3.2 with attack string $AH \cdot A$. If \mathcal{A} has $\alpha > 0.2$, it can choose from three scenarios, each with different RANDAO outputs. Note, \mathcal{A} needs to make the decision before seeing the honest party’s RANDAO contribution r_{31}^e . Consequently, \mathcal{A} may regret its decision.

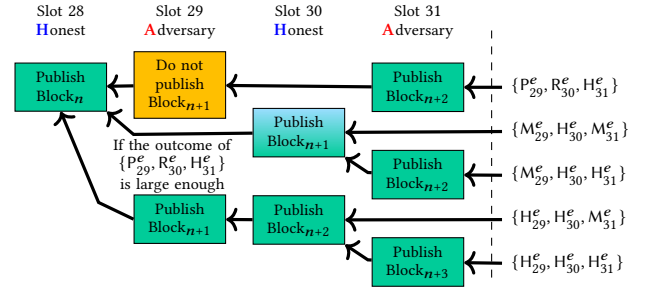


Figure 5: Decision tree for the forking and selfish mixing RANDAO manipulation with attack string $AHA \cdot$ and $\alpha > 0.2$. \mathcal{A} needs to make two decisions: first, before seeing the honest party’s RANDAO contribution r_{30}^e , and afterwards.

exact value of r_{31}^e will only be known after Slot 31, and it might be that case that \mathcal{A} regrets the forking decision made at Slot 30.

The forking attack becomes very effective when one combines it with selfish mixing. This combination occurs recursively, e.g., consider a variation of the previous attack with the epoch boundary shifted by one, resulting in $AHA \cdot$ and $\alpha \geq 0.2$. In this case, as shown in Figure 5, each branch of the decision tree offers an opportunity for an additional selfish mixing attack. This increases the number of possible RANDAO outcomes from three to five as follows:

- $\{H_{29}^e, H_{30}^e, H_{31}^e\}$: Parties publish blocks in their allotted slots.
- $\{H_{29}^e, H_{30}^e, M_{31}^e\}$: \mathcal{A} misses its tail slot to selfish mix.
- $\{P_{29}^e, R_{30}^e, H_{31}^e\}$: \mathcal{A} can reorg the honest block if $\alpha \geq 0.2$. This attack realisation is shown in Figure 1.
- $\{M_{29}^e, H_{30}^e, H_{31}^e\}$: \mathcal{A} foregoes forking after the 30th slot, i.e., \mathcal{A} misses to propose in the 29th slot but publishes in the last slot. Colloquially, we refer to this action as *regret*.
- $\{M_{29}^e, H_{30}^e, M_{31}^e\}$: \mathcal{A} may forego forking after the 30th slot and miss proposing blocks in both of its allotted slots. We refer to this branch (and the previous one) as “regretted forking”.

Although there are five possible outcomes, the attacker can choose from at most three, depending on a prior decision. In the full version of this paper [23], we provide a general description of how to construct decision trees recursively.

DEFINITION 2 (SLOT STATUS). *The slot status can be either:*

- **C** - Canonical, the slot is proposed or private.
- **N** - Non-canonical, the slot is reorged or missed.

We refer to the strings $c \in \{\mathbf{C}, \mathbf{N}\}^{\leq 32}$ as realisation strings corresponding to the e^{th} epoch's tail slots' statuses producing a different RANDAO outcome $R^{e,*}$.

For instance, for \mathbf{A}^2 , the four possible realisation strings are $\{\mathbf{C}^2, \mathbf{N}^2, \mathbf{CN}, \mathbf{NC}\}$ and for $\mathbf{as} = \mathbf{AH}\mathbf{A}$; $c = \{\mathbf{C}^3, \mathbf{CNC}, \mathbf{NCC}\}$.

3.2.1 *Ex-ante reorgs in a stronger adversarial model.* A wider range of attack strings is shown to permit ex-ante reorgs, as analysed in this paper. However, they require a stronger adversarial model that we refrained from incorporating into our system model. Specifically, the following ex-ante reorg is possible if we assume that network delay is under adversarial control. Evaluating these types of ex-ante reorgs could provide valuable insights for future work. If \mathcal{A} is given tail slots $\mathbf{AAH}\mathbf{H}$, \mathcal{A} can fork out the \mathbf{H} tail slot with $\alpha \geq 0.2$ stakes as follows. First, \mathcal{A} secretly builds in both its allocated slots but does not publish the blocks. Naturally, \mathbf{A} votes for both blocks; thus, this fork accumulates 2α votes. The honest players do not see this fork; hence, they vote for the first block preceding \mathbf{AA} as the head of the chain. Moreover, they build \mathbf{H} on top of the block proposed in Slot 28. When the block \mathbf{H}_{31} is published, it gets the proposer boost, $p_{\text{boost}} = 0.4$. \mathcal{A} immediately publishes its own secret fork obtaining 2α votes. Assuming that \mathcal{A} has control over the network delay and its fork reaches honest validators faster, they vote for the adversarial fork as the heaviest subtree despite the virtual proposer boost aiding honest validators. Such a robust adversarial model is frequently encountered in the PoS consensus literature [31].

3.2.2 *Ex-ante reorg attack strings.* As stated in Definition 1, a necessary condition for an attack string is that \mathcal{A} can determine at least one valid $R^{e,*}$ upon reaching its first slot. This is only possible if the future RANDAO reveals corresponding to honest blocks do not contribute to $R^{e,*}$ and \mathcal{A} can fork them out. The following theorem describes a condition under which \mathcal{A} can perform ex-ante reorgs in our model. The proof is deferred to Appendix A.1.

THEOREM 1 (CONDITION FOR FORKING). *Given \mathbf{A}^{a_1} slots followed by \mathbf{HXA} , (where $\mathbf{X} \in \{\mathbf{A}, \mathbf{H}\}^{h-1}$, $a_1, h > 0$) \mathcal{A} can perform an ex-ante reorg with $0 < \alpha < 0.5$ stakes forking out \mathbf{HX} if*

$$a_1 \geq \frac{h(1 - 2\alpha) - p_{\text{boost}}}{\alpha}. \quad (4)$$

Strings in the form $\mathbf{as} \in \mathbf{A}^{a_1} \mathbf{H}^{h_1} \cdot \mathbf{H}^{h_2} \mathbf{A}$ ($h_1 > 0$) are attack strings if Equation (4) is true for $h = h_1 + h_2$. If \mathcal{A} forks out $\mathbf{H}^{h_1+h_2}$, the RANDAO reveals of \mathbf{H}^{h_1} do not contribute to $R^{e,*}$.

THEOREM 2 (RECURSIVE ATTACK STRINGS). *Given any $S \in \mathcal{AS}_\alpha$, for a forking string $\mathbf{A}^{a_1} \mathbf{H}^h \mathbf{A}$ where Equation (4) holds, $\mathbf{A}^{a_1} \mathbf{H}^h \mathbf{AS}$ is also an attack string, i.e., $\mathbf{A}^{a_1} \mathbf{H}^h \mathbf{AS} \in \mathcal{AS}_\alpha$.*

Due to space constraints, the proof is deferred to Appendix A.1.

3.3 Ex-post reorging honest blocks

In contrast to ex-ante reorgs, ex-post reorgs allow \mathcal{A} to fork honest blocks out *after they were proposed*. A player can be motivated to

fork out blocks a posteriori if they contain unusually high transaction fees [10] for example. However, at the time of writing, no reliable technique is known for non-majority entities (i.e., $\alpha < 0.5$) to perform ex-post reorgs [31]. Therefore, we do not consider this forking manipulation in our theoretical model. Consequently, strings starting with \mathbf{H} are not attack strings, as the `randao_reveal` corresponding to \mathbf{H} is computationally hard to derive given that BLS signatures are existentially unforgeable. For convenience, we slightly abuse notation and refer to “ \mathbf{H} ” as a valid attack string. Nevertheless, ex-post reorgs can occur in practice, see Section 5.2.2 and could be used for RANDAO manipulation. If a validator fails to propose a block on time, the block might receive a small fraction ϵ of the attestations. If the subsequent block's proposer sees this and $\epsilon < p_{\text{boost}}$, then \mathcal{A} can fork out the preceding block for any α . At the time of writing, similar mechanisms i.e., *honest reorgs* [32] are implemented in the consensus protocol, though following them is optional. Reorgs typically occur when the previous block was proposed late and received few attestations — that is, when $\epsilon < 0.2$ — with the exception of certain edge cases. We do not model honest reorgs in our MDP because a strategic player cannot force honest validators to propose blocks late in our adversarial model.

3.4 Detectability of RANDAO manipulations

Not all RANDAO manipulations are created equal from a detection point of view. In some cases, \mathcal{A} can even plausibly deny that it has performed manipulative attacks.

3.4.1 *Detecting the selfish mixing manipulation.* If the adversary misses one of its tail blocks, say \mathbf{A} , then one cannot recompute the counterfactual RANDAO output $R^{e,*}$ that also contains the contribution r_{31}^e . This is because r_{31}^e is not publicly available and is hard to compute. Therefore, one cannot decide whether the adversary has missed a block on purpose (i.e., to manipulate the RANDAO for its own gains) or by accident, e.g., network outage. More generally, if the adversary owns k tail slots and misses $n \leq k$ of them, then one can only recompute 2^{k-n} RANDAO counterfactual outputs $R^{*,e}$ out of the 2^k counterfactual outputs that the adversary sees. This greatly limits the detectability of the selfish mixing manipulation strategy. We note, however, that if a validator misses tail slots enough times, one could apply a binomial statistical test (e.g., Student's t -test, Z -test) to check whether the validator manipulates the RANDAO output for its own gains, see Section 5.1.3.

3.4.2 *Detecting manipulations for reorged blocks.* Since reorged blocks had been proposed, they contain valid RANDAO contributions. For the sake of concreteness, consider the attack string $\mathbf{AH}\mathbf{A}$ and suppose the adversary reorgs block \mathbf{H}_{31} in the tail slot with RANDAO contribution r_{31}^e by building its block \mathbf{A} on top of each other. Now, one could recompute how many blocks the adversary could have obtained with ($R^e \oplus h(r_{31}^e)$) and without (R^e) the honest tail block. Therefore, these RANDAO manipulations are publicly verifiable. Our measurement results of these manipulations on the Ethereum mainnet can be found in Section 5.2.1.

4 RANDAO manipulation profitability

Moving further, we describe how to devise a stochastic strategy in our RANDAO manipulation model, which accommodates both

selfish mixing *and forking strategies*. Finally, we evaluate its efficacy and other properties of this policy.

4.1 RANDAO manipulations strategies

A detailed analysis of RANDAO manipulations, *focusing solely* on the selfish mixing strategy, can be found in [1]. The authors show that RANDAO manipulation is profitable not only because of

immediate rewards: it leads to more **A** slots in epoch $e + 2$, but also because of

future rewards: it enhances the attacker’s ability to further manipulate RANDAO in epoch $e + 2$, potentially yielding additional **A** slots in epoch $e + 4$, and so on.

Should one disregard future rewards, the attacker’s strategy is straightforward: given the opportunity for RANDAO manipulation, simply calculate the possible epoch strings and choose the one that yields the most **A** slots in epoch $e + 2$, subtracting the missed slots sacrificed in epoch e . To evaluate this strategy one can explicitly calculate the strategy’s reward, similar to how selfish mining was evaluated for Bitcoin PoW [14]. However, in order to account for future rewards, the attacker must compute an appropriate strategy. And MDP can be utilised to solve this problem. Alpturer and Weinberg [1] observe that in the case of selfish mixing, the optimal strategy admits a simple description due to a significant reduction in the Markov chain’s state space. The future reward is an additive value, represented by a utility function for each attack string, which also depends on the stake α . The attacker’s best strategy (policy) is to choose the RANDAO outcome that maximises the sum of the immediate rewards and the corresponding attack strings’ utility.

The RANDAO manipulation MDP for selfish-mixing is defined as a tuple consisting of *states*, *action space*, *policy*, *transition probabilities*, and *rewards*. The states represent selfish-mixing attack strings, such as A^t , for $t \in \{0, \dots, 32\}$. The action space allows the adversary, for each slot **A** in the attack string, to choose between proposing (H) or withholding (M) the adversarial slot.

As discussed in [1], the determination of the policy reduces to finding a utility function over the attack strings. The transition probabilities are determined by a game-like process, modelled using off-the-shelf stochastic methods whereby the attacker selects the RANDAO outcome (the observation) that maximises the sum of immediate rewards and the utility of the corresponding attack string. Specifically, the RANDAO outcome is sampled by assuming that each slot in epoch $e + 2$ is drawn independently, proportional to the stake α . The transition reward is given by the immediate reward. Finally, policy iteration is employed to compute a strategy that satisfies the Bellman equations, thereby proving its optimality.

Next, we overview how the MDP of selfish mixing can be generalised to account for *forking strategies*. The situation is significantly more complex in the case of forking for the following two reasons:

- (1) The attack strings may span the epoch boundary *cf.* Figure 4.
- (2) Some attacks require decisions before seeing all the possible RANDAO outputs *cf.* Figures 4 and 5.

To address point (1), we define *extended* attack strings, *i.e.*, strings that also take into account the beginning of the next epoch.

DEFINITION 3 (EXTENDED ATTACK STRING). For $(m, n) \in \mathbb{N}^2$, the set of extended attack strings $EAS_\alpha(m, n)$ consists of all possible strings

$$\{as + * + as_t \mid (as, as_t) \in AS_\alpha(m, n) \times AST_\alpha(m)\},$$

where $*$ denotes an arbitrary number of wildcard characters.

The number of extended attack strings is

$$|EAS_\alpha(m, n)| = |AS_\alpha(m, n)| \cdot |AST_\alpha(m)| \leq 2^{2m+n}.$$

For example an extended attack string can be $eas = \mathbf{AH} \cdot \mathbf{A} * \mathbf{A} \in EAS_\alpha(2, 1)$. Notice that this eas extends the attack string example of Figure 4. The end of the first epoch in eas allows the adversary to ex-ante fork the chain, while the end of the second epoch provides a selfish mixing opportunity. Intuitively, an $eas \in EAS_\alpha(m, n)$ string captures the fact that only the beginning and the end of an epoch string can contribute to the manipulative power of a strategic player. Looking ahead, our MDP will have eas strings as its states. We chose $m = 6$, $n = 2$ to keep our MDP state manageable. We let $\|es\|_{\mathbf{A}}$ denote the number of adversarial slots in an epoch string $es \in \{\mathbf{A}, \mathbf{H}\}$ ³².

Addressing point (2) is more involved; therefore, we focus for now on evaluating the case illustrated in Figure 4. For this, it is sufficient to build an MDP for the $EAS(2, 1)$ attack strings. In the following subsection, we examine this in detail. This also serves as a small illustrative example involving only 16 states. We can solve the MDP to obtain an optimal strategy, *i.e.*, the value iteration converges. As a result, we derive an optimal strategy that essentially consists of only four attack strings (**H**, **A**, **AA**, **AH****A**). Surprisingly, for $\alpha \geq 0.2$ this strategy outperforms (by ~ 0.05 more slots) the one based on selfish mixing [1]⁵ including 33 attack strings (A^t , for $t \in [0, 32]$).

4.2 A generalised MDP model for $EAS(2, 1)$

To illustrate our model of \mathcal{A} ’s behaviour, we introduce the main concepts using the smallest possible extended attack string space that makes forking attacks possible, *i.e.*, $EAS(2, 1)$.

Recall that an MDP is a tuple $M_\alpha = (S, A, \Pi, \{P_\pi\}_\pi, \{R_\pi\}_\pi)$ of states S , action space A , policy $\Pi : S \rightarrow A$, transition probabilities $\{P_\pi : S \times S \rightarrow \mathbb{R}\}_\pi$ and rewards $\{R_\pi : S \times S \rightarrow \mathbb{R}\}_\pi$. If we were to model all possible attack scenarios by an MDP, one would need to create a separate state for each possible point in time where a decision or an observation can be made, and the states should contain information about the observables. This leads to a massive increase in the state space, making standard policy iteration techniques infeasible.

Such an explosion of space was also observed in [1], where the following insight is used. Let the state space consist of extended attack strings (number of tail slots), and consider the observables (the possible RANDAO outcomes from various selfish mixing decisions) during state transitions only. Informally, this means that an action (*i.e.*, which slots to miss), the corresponding transition probabilities, and the collected immediate rewards depend not only on the policy and the current state but also on the outcome of an experiment performed in that state. Strictly speaking, the model is no longer an MDP, as identical actions and state transitions may result in different rewards. Nevertheless, as observed there, this

⁵The term "optimal" refers to a specific set of attack strings.

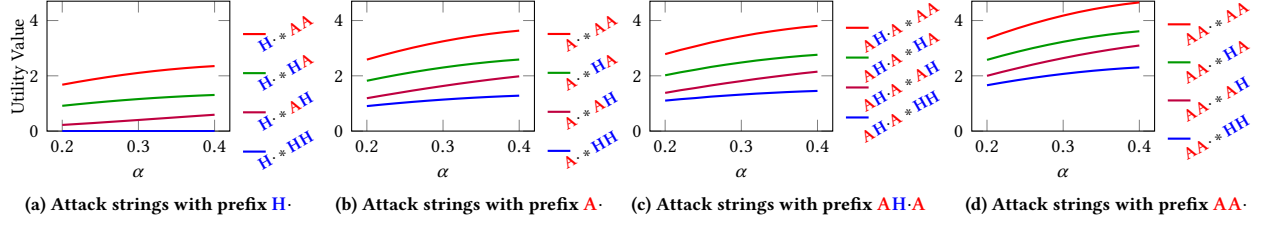


Figure 6: The utility function for each of the 16 extended attack string of $EAS(2, 1)$.

model is still suitable for policy iteration, which yields utility values for states, leading to a profitable policy.

We adopt the above-described approach to keep the state space as small as possible at the price of making state transitions more complicated. Our model consists of a state space of extended attack strings (cf. Section 4.2.1), an action space (cf. Section 4.2.2) and a policy (cf. Section 4.2.3). Note that our policies, as opposed to classical MDPs, yield randomised state transitions, as they result from a “virtual decision tree” corresponding to each state. We apply a utility function assigning a real number to each state accounting for the long-term benefit of being in the given state. Immediate rewards and transition probabilities are output as the result of actions. In the presentation below, we assume that a utility function $U(s)$ is already available for the extended attack strings s and formalize how one can derive them to describe the states of the decision trees. In this subsection, we assume that $0.2 \leq \alpha$ to ensure that the forking attack shown in Figure 4 is feasible.

4.2.1 The simplest MDP’s states. In our simplest case we have four attack strings, i.e., $AS_\alpha(2, 1) := \{H, A, AA, AH\}$. We refer to this as an honest attack string, two selfish mixing attack strings, and a forking attack string, respectively. The set of possible tail slots is defined as $AST_\alpha(2) := \{H, A, AA, AH\}$. Next, we generate $EAS(2, 1)$ as in Definition 3 to obtain $4 \cdot 4 = 16$ different extended attack strings, see Figure 6. Let $S := EAS(2, 1)$.

4.2.2 The MDP’s action space.

DEFINITION 4 (ATTACK ACTIONS). An attack action denotes a basic action \mathcal{A} can do. Let $\mathcal{B} := \{\text{Prop}, \text{Miss}, \text{Hide}, \text{Fork}, \text{Regret}\}$, where

- Prop_s^e - \mathcal{A} proposes a block in Slot s .
- Miss_s^e - \mathcal{A} misses in Slot s ; i.e., does not publish a block.
- Hide_s^e - \mathcal{A} builds a block in Slot s but does not publish it.
- Fork_s^e - \mathcal{A} builds a block in Slot s on the last hidden block.
- Regret_s^e - \mathcal{A} “forgets” its private blocks, foregoes forking.

Subscripts are omitted whenever they are clear from the context.

Here we illustrate the previously introduced attack actions on the decision trees shown in Section 3.2. Let $A := \mathcal{B}^*$. In case of a **selfish mixing** attack string (A^t , $t \in \{1, \dots, 32\}$), cf. Figure 3, each 2^t edges correspond to the $\{\text{Prop}, \text{Miss}\}^t$ actions. As mentioned above, forking attack strings admit a richer action space. For the sake of concreteness and simplicity consider a forking attack: $\text{eas} = AH\cdot A$. In this case, \mathcal{A} has two decisions to make. The first decision is before \mathcal{A} sees the RANDAO contribution r_{31}^e , and the second decision is after the publication of H_{31}^e .

- Prop_{30}^e : \mathcal{A} behaves honestly, i.e., \mathcal{A} publishes the block A_{30}^e .
- Hide_{30}^e : \mathcal{A} starts forking by privately building A_{30}^e . After r_{31}^e is published, \mathcal{A} makes another decision:
 - Fork_0^{e+1} : completes forking by proposing A_0^{e+1} upon A_{30}^e .
 - $\text{Regret}_0^{e+1} \text{Prop}_0^{e+1}$: the so-called “regret” branch of the decision tree. \mathcal{A} foregoes forking by proposing A_0^{e+1} on top of H_{31}^e , essentially forfeiting A_{30}^e .

4.2.3 The MDP’s policy. We assign a decision tree (e.g., Figures 4 and 5) to each eas_e , which \mathcal{A} traverses. This yields R^e and the next state eas_{e+1} .

At a high level, we briefly describe now the structure of the decision trees corresponding to forking attack strings (starting with $A^{a_1}H^hA^{a_2}$). \mathcal{A} decides to behave honestly based on a threshold (cf. Definition 6) or starts forking. If \mathcal{A} decides to start forking (e.g., by privately building all a_1 blocks: Hide^{a_1}), after H^h , \mathcal{A} can finalise the ex-ante reorg or “regret” the attack by abandoning A^{a_1} . An important observation is that, in both cases, after a successful forking or regretted attack, \mathcal{A} still aims to maximise the utility until the epoch boundary, regardless of the previously sacrificed blocks. Such a recursive structure allows us to describe the new state using a shorter attack string derived from the remaining slots in epoch e , cf. Section 4.3. This observation holds for the case of “no forking” as well, since after proposing $A^{a_1}H^h$, \mathcal{A} still seeks to maximise utility. We define (the multisets of) observations to model each decision situation. Intuitively, whenever \mathcal{A} can manipulate the RANDAO, it observes and arbitrarily chooses from different RANDAO outputs $R^{e,*}$ along with their corresponding extended attack strings eas , actions act leading towards $R^{e,*}$ and the number of sacrificed blocks sac . On the other hand, there are no observations for honest actions, and the expected utility is compared to the derived utilities of the observations.

DEFINITION 5. Let $\mathcal{O} := \{(c, \text{act}, \text{eas}, \text{sac}) \in \{\mathbf{C}, \mathbf{N}\}^m \times \mathcal{B}^* \times EAS(m, n) \times \mathbf{N}\}$ be the observation set, where c is the realisation string yielding eas and sac , while act is the corresponding action towards c . Ω denotes the set of all possible multisets of observations.

We compute the immediate reward by evaluating $\text{cnt}(\text{eas}, \text{sac}) := \|\text{eas}[-32 :]\|_{\mathbf{A}} - \text{sac}$ by choosing a certain eas . A policy Π is defined as a total ordering on Ω and a threshold corresponding to the “no forking” action, cf. Definition 6. For selfish mixing states s with t tail slots, one can evaluate the utility $U(s) = \max_{i \in [1, 2^t]} (\text{cnt}(\text{eas}_i, \text{sac}_i) + U(\text{eas}_i))$ where each $(\text{eas}_i, \text{sac}_i)$ corresponds to the i^{th} observation. There are different actions \mathcal{A} can perform: it can either decide not to build a private chain, referred to as the “no forking” action, or it

can choose to build a private chain, which can be done in multiple ways. Next, we define utility thresholds based on which \mathcal{A} decides which action to choose.

DEFINITION 6 (NO FORKING THRESHOLD). For any MDP state $s \in S$, we define (cf. [29, Eq.(17.4)]) a threshold for the “no forking” action $a \in A(s)$ (the action a is defined by the corresponding eas forking string): $N_\alpha(s) := \sum_{s' \in S} P(s'|s, a)U(s')$. Here, s' denotes the states reached after the “no forking” action, i.e., \mathcal{A} proposes a block honestly in all its assigned slots A^{a_1} . Note that this threshold does not depend on the observations of s .

DEFINITION 7 (FORKING THRESHOLD). For a given MDP state s , and forking action $a \in A(s)$, \mathcal{A} can observe a set of observables $O \in \Omega$. The forking threshold is defined as $F_\alpha(s, a, O) := \sum_{s' \in S_O} P(s'|s, a)U(s')$. Here, S_O is the set of states in O , i.e., the observable states in the decision tree after the forking action a .

THEOREM 3. For any forking eas of a state s and observations $O \in \Omega$, the following equation holds for the optimal utility:

$$U(s) = \max(\{N_\alpha(s), F_\alpha(s, a_1, O), \dots, F_\alpha(s, a_n, O)\}) , \quad (5)$$

where $\{a_1, \dots, a_n\} \subset A(s)$ are the actions whereby \mathcal{A} starts ex-ante reorging in a specific way.

The proof is deferred to Appendix A.1.

As expected, \mathcal{A} chooses the action with the largest utility. Note that when \mathcal{A} starts this type of attack by hiding some of the A^{a_1} blocks, and once observes the H^h blocks, the adversary might regret its action. From the new observations \mathcal{A} can decide to do so accordingly in the shorter eas e' . Recall, that we can describe a new state still in epoch e after a regretted or successful forking with the remaining slots, taking into account the already sacrificed blocks.

4.3 Forking attacks’ recursive characterization

The advantage of the $EAS(2, 1)$ example was that it contained only a single forking attack. As shown in Figure 5, many more such attack strings exist, and they can be reduced recursively to one another. The interested reader is referred to Figure 5 where a selfish mixing attack is launched on the branch after a block is proposed in slot 29. In our evaluation, we employed a recursive construction to characterise these attacks.

The following examples illustrate some of the intricacies of forking attack strings that make their analysis challenging.

Self-forking at the head slot (AHA·A) The tail-forking attack can be even more profitable if the adversary also controls the next head slot. In such cases – perhaps unexpectedly – the decision tree for $\alpha \geq 0.32$ includes a branch where it may be more profitable for the adversary to omit its own block A_{31} . Put differently, if $\alpha \geq 0.32$, then $\{P_{29}^e, R_{30}^e, M_{31}^e, H_0^{e+1}\}$ is a viable configuration. \mathcal{A} may be incentivised to prolong the forking attack and *miss its own block* A_{31} , akin to the sacrificed blocks in selfish mixing.

Different decision tree for tail forking depending on α (A^2HA ·) When $0.15 \leq \alpha < 0.2$, then \mathcal{A} must start building a branch from A_{28} preceding H_{30} for forking. \mathcal{A} is able to fork out H_{30} , while doing selfish mixing at the same time. Consider the two possible slot statuses $\{P_{28}^e, P_{29}^e, R_{30}^e, H_{31}^e\}$ and $\{P_{28}^e, M_{29}^e, R_{30}^e, H_{31}^e\}$. In other words, \mathcal{A} can fork out H_{30} even sacrificing A_{29} . On the other hand, if $0.2 \leq \alpha$, \mathcal{A} can minimize the sacrifice of a regretted fork while

reaching the same $R^{e,*}$ with $\{H_{28}^e, P_{29}^e, R_{30}^e, H_{31}^e\}$ by proposing A_{28} . This way, \mathcal{A} builds on top of A_{28} , only sacrificing A_{29} . Finally, \mathcal{A} can fork out H_{30} , while missing A_{28} , because \mathcal{A} can gather enough votes even if the forking starts forking at Slot 29. A generalization of how much slot is required before H is stated in Theorem 1.

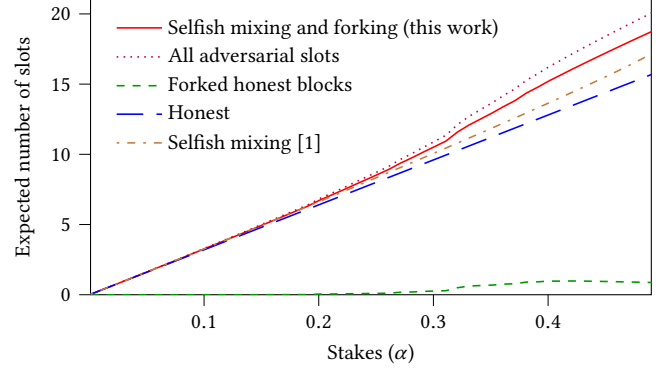


Figure 7: Efficacy of various RANDAO manipulation strategies. Note that a strategic player who applies both forking and selfish mixing strategies outperforms the RANDAO manipulator using only selfish mixing. As a result of RANDAO manipulation, chain quality is also reduced, i.e., see the increased number of forked honest blocks.

Recursive (forking) attack strings As Theorem 2 suggests attack strings can be built recursively. We illustrate this recursion with the attack string as = $AHAAHA$. Each action performed by \mathcal{A} reduces the attack string to a shorter as' , which has already been evaluated.

- **Prop₂₆** yields A^2HA · (since H_{27} is always proposed). Therefore the realisation string ($\in \{C, N\}^6$) will start with $C_{26}C_{27}$.
- **Hide₂₆** action produces the following possibilities:
 - **Fork₂₈**: the realisation string will start with $C_{26}N_{27}C_{28}$ and \mathcal{A} has to maximise the utility in AHA .
 - **Regret₂₈**: in this case, the realisation string starts with $c = N_{26}C_{27}$. Afterwards, \mathcal{A} maximises the utility in the attack string A^2HA · regardless of the lost block A_{26} .

4.4 Results

We solved our generalised MDP for the extended attack strings for $EAS_\alpha(6, 2)$. Our observations indicate that RANDAO manipulations employing forking strategies outperform selfish mixing with $\alpha = 0.08$, as demonstrated in Figure 7. A RANDAO manipulator staking entity with $\alpha = 41.95\%$ can attain 50% of the proposed slots. In contrast, selfish mixing alone would require an $\alpha = 46.24\%$ to get half of the slots. As depicted in Figure 8, the presence of a RANDAO manipulator validator leads to a significant degradation in blockchain throughput due to missed blocks in the adversarial slots and by performing purposeful forks. For example, \mathcal{A} with $\alpha = 0.33\%$ causes a 3.86% reduction in blockchain throughput. The decreased transaction throughput is a negative externality for every user of the system.

In Figure 9, we analyse the probability distribution of the different actions in the stationary distribution of our generalised MDP.

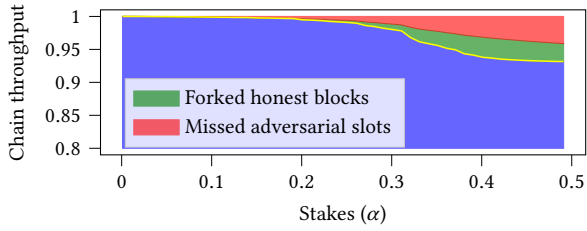


Figure 8: Blockchain throughput degradation as a function of the RANDAO manipulator adversary's staking power α .

For example, for $\alpha = 0.30$, we see that a strategic \mathcal{A} performs an ex-ante reorg in 18.94% of the epochs, regrets forking in 3.31% of the epochs, selfish mixes in 15.99% of the epochs and only acts honestly in 61.76% of the epochs. Ideally, a blockchain implementation using an *unbiased randomness beacon* should only incentivise honest behaviour.

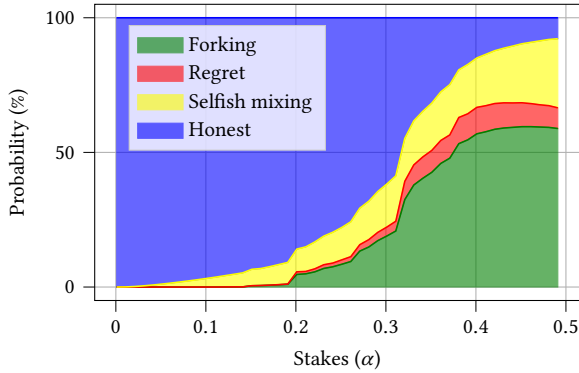


Figure 9: The probability of each strategic behaviour in the stationary distribution. If \mathcal{A} ex-ante reorged once at the epoch, it is counted as Forking, if regretted an attack but did not fork, it is counted as Regret etc. Observe, that large staking entities are often incentivised to manipulate the RANDAO. For instance, an economically rational staker with $\alpha = 0.3$ manipulates the RANDAO in 38.24% of the epochs.

Table 2 compares the strategic players' manipulative power in PoS Ethereum with that of in PoW Bitcoin using selfish mixing with/without forking and selfish mining. Observe that the corresponding effective stakes with the forking strategy surpass that of selfish mixing alone when the attacker has large stakes, and it can efficiently fork out honest blocks. Currently, the largest staking pool controls 28.1%⁶ of the stakes. This staking power can be increased by 8.37% with our combination of block withholding and forking attacks. By comparing the numbers in Table 2 we can say that the current DRB in the Ethereum network is less biasable than that of the Bitcoin PoW blockchain.

⁶See: <https://beaconcha.in/pools>.

Stake (α)	Ethereum (PoS)		Bitcoin (PoW)
	Selfish Mixing (SM) [1]	This work SM+forking	Selfish Mining [14, $\gamma = 1$ in (3)]
1%	1.0011%	1.0011%	1.0099%
5%	5.0483%	5.0483%	5.2387%
10%	10.1881%	10.1882%	10.9194%
15%	15.3996%	15.4243%	17.0110%
20%	20.6777%	20.9583%	23.5165%
25%	26.0247%	26.6090%	30.4878%
30%	31.4516%	32.8356%	38.0622%
35%	36.9735%	40.2870%	46.5560%
40%	42.6244%	47.4946%	56.7442%
45%	48.4918%	53.7777%	70.9423%

Table 2: Advantages in terms of staking power of strategic RANDAO manipulators. We compare this work and prior work by Alpturer and Weinberg [1] for various stake sizes. As a reference, we show the increased revenue achievable using selfish mining in the largest permissionless consensus protocol by market capitalisation, i.e., Bitcoin PoW [24].

5 RANDAO Manipulations in Practice: Empirical Measurements

In this section we present our results of investigation regarding RANDAO manipulation in the wild. Primarily, we examined epoch boundaries where there were missed slots at the beginning or end of an epoch. To effectively investigate these attacks, we grouped the validators and assumed that the attacker is part of one or more of these groups. Each block proposer can be identified by a public key. Note that this data alone is insufficient to perform statistical tests because a staking entity typically controls numerous validators. Some entities voluntarily reveal their public keys, which permits the derivation of a mapping between proposers and their corresponding identities. To that end, we use public data obtained using various APIs,⁷ and process the blocks over a two-year period, from September 22, 2022, to October 1, 2024 (epochs in the range [148412, 314998]),⁸ extracting the actual block proposers and the number of missed blocks for each epoch.

5.1 Selfish mixing

First, we search for statistical evidence of systematic RANDAO manipulations using selfish mixing on Ethereum mainnet.

5.1.1 Missing consecutive tail slots. Major staking entities have had numerous opportunities to conduct selfish mixing RANDAO manipulations, see Figure 10. We found that the biggest staker, Lido, could have manipulated the RANDAO with selfish mixing 47 694 times. In particular, Lido had 4 consecutive tail slots in 737 different epochs and missed at least one slot in 3 of these instances.

⁷<https://beaconcha.in/>, <https://beaconscan.com/>, <https://etherscan.io/>.

⁸Before the merge (September 2022) the beacon chain and the RANDAO had not been used to select validators. Hence, we disregard epochs $e \in [0, 148411]$.

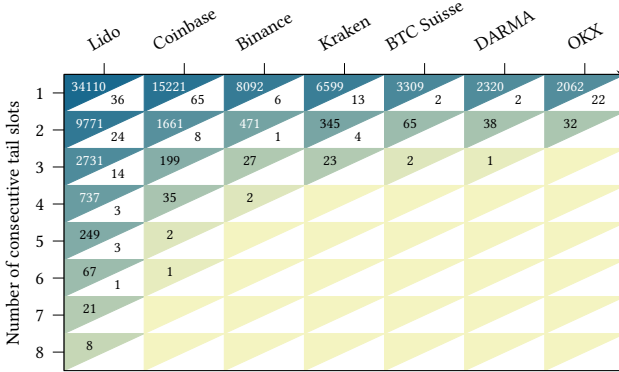


Figure 10: Selfish mixing manipulation opportunities for major staking entities in epochs [148412 – 314998]. The values in the coloured triangles indicate the number of epochs during which the staking entity had a certain number of consecutive tail slots, while the white triangles indicate how many of these epochs had at least one missed tail slot.

5.1.2 Not missing tail slots. Here our investigation centres on consecutive tail slots, specifically those during which the entities reliably proposed all assigned blocks. Recall that in this case, all counterfactual RANDAO outputs $R^{e,*}$ in epoch e are computable by anyone, cf. Section 3.4.1. Given t tail slots, the adversary could have chosen from 2^t RANDAO outputs. We label the realised R^e as **BEST** if not missing any slot was a better strategy for maximising the reward, defined as the difference between $\mathcal{P}(e+2, R^e, \mathcal{A})$ and $\mathcal{S}(e, R^e, \mathcal{A})$. We call a RANDAO output **NEUTRAL** for the tail slot owner if there were other achievable configurations (i.e., by missing tail slots) that would have yielded the same number of slots. Finally, the RANDAO output is labelled as **WORSE** if the adversary could have chosen a configuration that would have yielded strictly more slots in epoch $e+2$ than the one realised by choosing not to miss any slots. As illustrated in Table 3, in several cases, major stak-

Table 3: Selfish mixing strategies for stakers not missing any tail slots. Note that entities’ staking power typically changes over time. Here, α denotes the entities’ average staking power in the samples of selfish mixing candidates.

Entity	BEST	NEUTRAL	WORSE	Stake* (α)
Lido	14 788	3 244	12 883	28.5
Coinbase	9 224	2 685	4 954	10.96
Binance	5 151	1 616	1 745	5.63
Kraken	4 318	1 319	1 268	5.34
Bitcoin Suisse	2 359	701	290	2.06
Upbit	1 184	253	53	1.05
OKX	1 540	399	120	1.48

ing entities could have achieved better outcomes by withholding blocks in their tail slots. As an example, an economically rational Lido operator would be expected to forgo at least one slot in 12,883 instances, but in practice, none were missed.

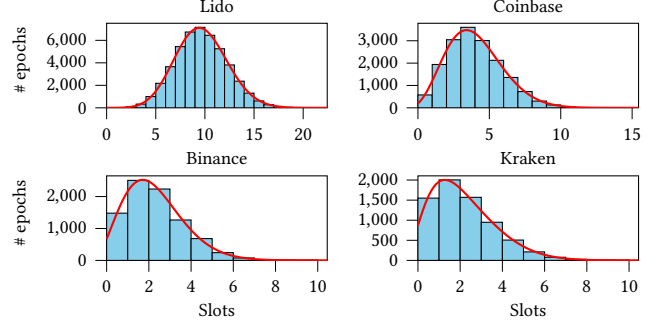


Figure 11: Number of slots in epoch $e+2$ without missed slots in epoch e for Lido, Coinbase, Binance, and Kraken. The blue bars represent the empirical distribution, while the red line is the expected theoretical distribution for the number of obtained slots in epoch $e+2$.

We examined the distribution of slots for the major staking entities when their validators proposed all tail slots. Let N_p be the number of such epochs and $1 \leq k \leq N_p$: e_k^p be the k -th such epoch. If no RANDAO manipulation occurred with the utility of maximising the number of proposed blocks, the number of obtained slots $\mathcal{P}(e+2, \cdot, \mathcal{A})$ in epoch $e+2$ by an entity would be binomially distributed see Figure 11. Thus, the null hypothesis is

$$H_0 : \forall k \in \mathbb{N}(1 \leq k \leq N_p) : \mathcal{P}(e_k^p, \cdot, \mathcal{A}) \sim \text{Binom}(32, \alpha_{e_k^p}) . \quad (6)$$

To statistically test whether $\forall k \in [1, N_p] : \mathcal{P}(e_k^p, \cdot, \mathcal{A})$ are from the established distributions, we take the histogram of the $\mathcal{P}(e_k^p, \cdot, \mathcal{A})$ values between 0 and 32. We define an indicator variable as $I_{k,i} = 1$ if $\mathcal{P}(e_k^p, \cdot, \mathcal{A}) = i$, and 0 otherwise. Let H^p be the empirical distribution of $\mathcal{P}(e^p, \cdot, \mathcal{A})$, where $H_i^p = \sum_{k=1}^{N_p} I_{k,i}$. If H_0 holds, $I_{k,i} \sim \text{Bernoulli}\left(\binom{32}{i} \alpha_{e_k^p}^i (1 - \alpha_{e_k^p})^{32-i}\right)$, then

$$\hat{H}^p := \mathbb{E}(H_i^p) = \sum_{k=1}^{N_p} \mathbb{E}(I_{k,i}) = \sum_{k=1}^{N_p} \binom{32}{i} \alpha_{e_k^p}^i (1 - \alpha_{e_k^p})^{32-i} . \quad (7)$$

We assess the agreement between the theoretical distribution H^p and the empirical histogram \hat{H}^p using the χ^2 -test. To comply with the assumptions of the chi-squared test, we aggregated adjacent histogram bins until each resulting bin H_i^p satisfied $H_i^p \geq 5$ for all $i \in [0, \text{Bins}]$. According to our analysis presented in Table 4, none of the entities exhibit a statistically significant deviation from the expected distribution at the 5% significance level. However, Bitcoin Suisse approaches significance, with a p -value of 0.055.

5.1.3 Missing a few tail slots. Selfish mixing entails selectively missing tail slots, but it can also be caused by other reasons, e.g., network outage. Next, we test this hypothesis. Under the null hypothesis, missed slots at the end of an epoch are attributed solely to external factors, not to strategic RANDAO manipulation. That is, a rejection of the null hypothesis provides evidence for systematic RANDAO manipulation. Generally, considering the epochs in which an entity has missed at least one slot, the number of slots obtained $\mathcal{P}(e, \cdot, \mathcal{A})$ should be distributed as the sum of discrete

binomial distributions, a Poisson binomial distribution. Let N be the number of epochs in which the entity in question missed at least one tail slot, and for $1 \leq k \leq N_m$, let e_k^m represent the k -th such epoch. It is important to note that α_e , the staking power of the validator entity, is time-dependent. As a result, conventional statistical tests assuming a fixed distribution, such as the Z-test, cannot be employed. Instead, we test whether

$$\sum_{k=1}^{N_m} \mathcal{P}(e_k^m, \cdot, \mathcal{A}) \sim \text{PoissonBinom}(32, (\alpha_{e_1^m}, \alpha_{e_2^m}, \dots, \alpha_{e_{N_m}^m})) \quad (8)$$

If $N_m > 50 \wedge \forall k \in [1, N] : \alpha_{e_k^m} > 0.05$, we can approximate this distribution with a normal distribution. We perform a right-tailed normality test on the value $\sum_{k=1}^{N_m} \mathcal{P}(e_k^m, \cdot, \mathcal{A}) \sim \mathcal{N}(\mu, \sigma^2)$. Let $\mu_0 = 32 \sum_{i=1}^N \alpha_{e_i}$ and $\sigma_0 = \sqrt{32 \sum_{i=1}^N \alpha_{e_i} (1 - \alpha_{e_i})}$. The null hypothesis is $H_0 : \mu \leq \mu_0$, while the alternative hypothesis is $H_1 : \mu > \mu_0$. Rejection of the null hypothesis is not possible in any of the applicable cases due to the high p -values; refer to Table 5. Thus, missed tail slots are not part of a systemic RANDAO manipulation by any of the studied entities.

Table 4: χ^2 test on the number of slots entities obtained when proposed all tail slots. N denotes the number of examined epochs and $Bins$ the bins of the histogram.

Validator	N	Bins	p -value
Lido	46 879	19	0.852
Coinbase	16 883	13	0.325
Binance	8 512	8	0.142
Kraken	6, 905	9	0.803
Bitcoin Suisse	3 350	6	0.055
Upbit	1 490	4	0.465
OKX	2 059	5	0.303

Table 5: Z-test results applied to the number of slots obtained by Lido and Coinbase in epochs where they missed at least a tail slot. No evidence for systematic RANDAO manipulations.

Validator	N	μ	μ_0	σ_0	p -value
Lido	80	746	752.21	23.039	0.606
Coinbase	71	222	228.344	14.304	0.671

5.2 Forking RANDAO manipulations

Next, we search for statistical evidence of systematic RANDAO manipulations using ex-ante and ex-post forking.

5.2.1 Forking for slot number maximizing utility. With current staking powers, only a handful of entities can execute an ex-ante reorg. Possible candidates include Lido, Coinbase, Kraken, etc. The difficulty of performing a reorg attack lies not only in acquiring sufficient staking power, but also in the presence of a favourable tail slot configuration, e.g., **AHA**. We identified only two instances, at epochs 192 420 and 313 291, where Lido ex-ante reorged Gemini

and a solo staker at the 30th slot of the epoch, respectively. Subsequently, Lido got more slots (8 in both cases) in the upcoming epochs. Had Lido not forked them out and behaved honestly, it would have obtained only 7 slots in epochs 192 422 and 313 293. We empirically evaluated Lido’s unrealised rewards. In particular, Lido could have manipulated the RANDAO numerous times, even though it did not engage in these manipulations. This economically irrational behaviour resulted in 74 782 slots of unrealised rewards between epochs 190 000 and 314 994.

5.2.2 Forking and MEV. Many works analyse and quantify the incentives implied by MEV [12, 20, 27, 35]. Next, we study the potential correlation (or lack thereof) between the incentives due to MEV and RANDAO manipulation. Specifically, we considered instances of ex-ante reorgs and ex-post reorgs (cf. Section 3.3) that happened for the strings ending with **HA** or **HA**. Recall that ex-post reorgs can occur in practice accidentally if **H** had been published late into its slot and received an unusually low number of votes. We observe numerous reorgs executed by 3 major staking entities. However, we find no correlation between RANDAO and MEV incentives; see Appendix C.1 for our empirical measurements.

6 Countermeasures

A frequently discussed countermeasure against RANDAO manipulations within the Ethereum community is social slashing [34]. Several significant challenges exist with this approach. First, social slashing is difficult to agree upon and enforce in a decentralised environment. Second, social slashing could become an even greater centralising force than the one it seeks to prevent, potentially leading to the unfair enrichment of already wealthy validators. Third, algorithmic or cryptographic countermeasures are desired instead of social deterrents in a trust-minimised, geo-economically decentralised environment. On the other hand, there seem to be no simple, immediate algorithmic or cryptographic countermeasures without fundamentally redesigning the currently used RANDAO protocol. Alpturer and Weinberg observed that the manipulability of RANDAO is reduced with more slots per epoch, as the tail slots of longer epochs provide less opportunity for manipulation [1]. Additionally, the protocol could impose financial penalties for failing to publish blocks. If the proposer boost parameter were decreased, ex-ante reorgs would become more difficult to achieve. However, it is important to note that none of these countermeasures eliminate RANDAO manipulability *entirely*.

We highlight two cryptographic protocols that could offer a viable, *unbiasable* alternative to the current RANDAO protocol. First, verifiable delay functions (VDFs) could protect against DRB biasability, by delaying the full DRB output and forcing all participants to submit their contributions before learning the output [5]. If the delay parameter is chosen properly, no bias is possible in the *dishonest majority setting* (under proper cryptographic assumptions). VDFs are non-trivial to deploy [4] and have severe hardware requirements. We also consider weighted threshold VRFs [13] as a promising cryptographic primitive for instantiating an unbiased DRB in the *honest majority setting*. Unfortunately, the currently known only BLS-based weighted threshold VRF protocol [13] does not scale to millions of validators, the scale Ethereum would need.

7 Related work

The manipulability of leader selection mechanisms in Byzantine fault-tolerant PoS blockchains has been analysed in [15, 16] to uncover the relationship between the power of the adversary and the network connectivity parameter. Yaish et al. proceed by defining multiple variants of a timestamping attack and find that these attacks were performed in the wild [37], making it the first confirmed case of consensus-level manipulation in a major cryptocurrency.

A great summary of DRBs can be found in [11, 21, 28]. The vulnerabilities of the DRB in PoS Ethereum have been discussed by the community since 2018 [8, 25, 33, 34]. Then, an initial formal verification analysis of an earlier, two-round version of the RANDAO [38] protocol was presented in [2]. Alpturer and Weinberg [1] show that the RANDAO manipulation game in Ethereum can be formulated as an MDP and propose a state-space reduction method that allows policy iteration to converge quickly on a laptop.

8 Conclusion and future directions

We presented the currently known most powerful RANDAO manipulations by considering the selfish mixing strategy and also the ex-ante forking strategy, where the adversary forks out honest blocks from the canonical chain to increase its manipulative power.

We foresee three main areas for future work. A complete, albeit technically much more difficult, MDP could model even more strategies for possible RANDAO manipulation. In our current model, we do not consider slot-varying rewards (i.e., slots' values are not uniform), neither ex-post reorgs nor the manipulation and suppression of honest attestations in sync committees.

We foresee the emergence of a RANDAO bribery market, where validators can auction off their RANDAO manipulation rights if they obtain tail slot(s) in an epoch. A corrupt validator could publish their own RANDAO contribution before their allotted tail slot(s). After learning every RANDAO reveal, market participants could offer bribes to the corrupt validator not to publish their tail slot(s). Note that such a market could be implemented trustlessly using smart contracts. The game-theoretic implications of such a complex market could be studied in future work.

Future work should amend (e.g., longer epochs), fix or even replace the current RANDAO protocol with an adequate non-biasable DRB protocol (e.g., VDFs [5], weighted threshold VRFs [13]), given the Ethereum consensus protocol's stringent latency requirements.

Acknowledgments

We are grateful to Joachim Neu for insightful discussions on various attacks on Ethereum. We thank Arantxa Zapico and Gottfried Herold for discussions on RANDAO bribery markets. We are indebted to Kaya Ito Alpturer and Matt Weinberg for sharing their manuscript on optimal RANDAO manipulations with us. Last but not least, we thank Toni Wahrstätter for inspiration. István András Seres was supported by the Ministry of Culture and Innovation and the National Research, Development, and Innovation Office within the Quantum Information National Laboratory of Hungary (Grant No. 2022-2.1.1-NL-2022-00004). Bence Ladóczki and János Tapolcai received financial support from the National Research, Development, and Innovation Office (NKFIH, Grant No. K-146347).

References

- [1] Kaya Alpturer and Matthew Weinberg. 2024. Optimal RANDAO Manipulation in Ethereum. *Advances in Financial Technologies* (2024).
- [2] Musab A Alturki and Grigore Roşu. 2020. Statistical model checking of RANDAO's resilience to pre-computed reveal strategies. In *Formal Methods. FM 2019 International Workshops: Porto, Portugal, October 7–11, 2019, Revised Selected Papers, Part I* 3. Springer, 337–349.
- [3] Vivek Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. 2022. Proof-of-Stake Longest Chain Protocols: Security vs Predictability. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus (Los Angeles, CA, USA) (Consensus-Day '22)*. Association for Computing Machinery, New York, NY, USA, 29–42. doi:10.1145/3560829.3563559
- [4] Alex Biryukov, Ben Fisch, Gottfried Herold, Dmitry Khovratovich, Gaëtan Leurent, Maria Naya-Plasencia, and Benjamin Wesolowski. 2024. Cryptanalysis of algebraic verifiable delay functions. In *Annual International Cryptology Conference*. Springer, 457–490.
- [5] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. 2018. Verifiable delay functions. In *Annual international cryptography conference*. Springer, 757–788.
- [6] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *International conference on the theory and application of cryptography and information security*. Springer, 514–532.
- [7] Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S. Matthew Weinberg. 2019. Formal Barriers to Longest-Chain Proof-of-Stake Protocols. In *Proceedings of the 2019 ACM Conference on Economics and Computation (Phoenix, AZ, USA) (EC '19)*. Association for Computing Machinery, New York, NY, USA, 459–473. doi:10.1145/3328526.3329567
- [8] Vitalik Buterin. 2018. RANDAO beacon exploitability analysis, round 2. <https://ethresear.ch/t/randao-beacon-exploitability-analysis-round-2/1980>. Accessed: 2024-11-12.
- [9] Vitalik Buterin, Diego Hernandez, Thor Kamphofner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. 2020. Combining GHOST and casper. *arXiv preprint arXiv:2003.03052* (2020).
- [10] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. 2016. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 154–167.
- [11] Kevin Choi, Aathira Manoj, and Joseph Bonneau. 2023. SoK: Distributed Randomness Beacons. *Cryptology ePrint Archive*, Paper 2023/728. <https://eprint.iacr.org/2023/728>
- [12] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2019. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. *arXiv:1904.05234 [cs.CR]*
- [13] Sourav Das, Benny Pinkas, Alin Tomescu, and Zhuolun Xiang. 2024. Distributed randomness using weighted vrf. *Cryptology ePrint Archive* (2024).
- [14] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In *Int. Conf. on Financial Cryptography and Data Security (FC)*. Springer, 436–454.
- [15] Matheus V X Ferreira, Ye Lin Sally Hahn, S Matthew Weinberg, and Catherine Yu. 2022. Optimal strategic mining against cryptographic self-selection in proof-of-stake. In *Proceedings of the 23rd ACM Conference on Economics and Computation*. 89–114.
- [16] Matheus V. X. Ferreira, Aadityan Ganesh, Jack Hourigan, Hannah Huh, S. Matthew Weinberg, and Catherine Yu. 2024. Computing Optimal Manipulations in Cryptographic Self-Selection Proof-of-Stake Protocols. In *The 25th ACM Conference on Economics and Computation (EC '24)* (New Haven, CT, USA, July 8–11, 2024). ACM, New York, NY, USA, 43. doi:10.1145/3670865.3673602
- [17] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.
- [18] Ethereum Foundation. 2023. Ethereum Proof-of-stake documentation. <https://github.com/ethereum/consensus-specs/blob/8696fbf75387fb37a32fc08a6b934653198c6c0c/specs/phase0/validator.md#attesting>. Accessed: 2024-11-12.
- [19] Viet Tung Hoang, Ben Morris, and Phillip Rogaway. 2012. An enciphering scheme based on a card shuffle. In *Advances in Cryptology—CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2012. Proceedings*. Springer, 1–13.
- [20] Aljosha Judmayer, Nicholas Stifter, Philipp Schindler, and Edgar Weippl. 2022. Estimating (miner) extractable value is hard, let's go shopping!. In *International Conference on Financial Cryptography and Data Security*. Springer, 74–92.
- [21] Alireza Kavousi, Zhipeng Wang, and Philipp Jovanovic. 2024. SoK: Public Randomness. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. 216–234. doi:10.1109/EuroSP60621.2024.00020
- [22] Andrew Lewis-Pye and Tim Roughgarden. 2023. Permissionless Consensus. *arXiv preprint arXiv:2304.14701* (2023).

- [23] Ábel Nagy, János Tapolcai, István András Seres, and Bence Ladóczki. 2025. Forking the RANDAO: Manipulating Ethereum's Distributed Randomness Beacon. *Cryptology ePrint Archive* (2025).
- [24] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
- [25] Paul D. 2018. Limiting last-revealer attacks in beacon chain randomness. <https://ethresear.ch/t/limiting-last-revealer-attacks-in-beacon-chain-randomness/3705/1>. Accessed: 2024-11-12.
- [26] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. 2022. Ethereum Proof-of-Stake under Scrutiny. *arXiv preprint arXiv:2210.16070* (2022).
- [27] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying blockchain extractable value: How dark is the forest?. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 198–214.
- [28] Mayank Raikwar and Danilo Gligoroski. 2022. Sok: Decentralized randomness beacon protocols. In *Australasian Conference on Information Security and Privacy*. Springer, 420–446.
- [29] Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach*. Pearson.
- [30] Roozbeh Sarenche, Svetla Nikova, and Bart Preneel. 2025. Deep Selfish Proposing in Longest-Chain Proof-of-Stake Protocols. In *Financial Cryptography and Data Security*, Jeremy Clark and Elaine Shi (Eds.). Springer Nature Switzerland, Cham, 24–40.
- [31] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. 2022. Three Attacks on Proof-of-Stake Ethereum. In *Financial Cryptography and Data Security*, Ittay Eyal and Juan Garay (Eds.). Springer International Publishing, Cham, 560–576.
- [32] Michael Sproul. 2023. *Allow honest validators to reorg late blocks*. <https://github.com/ethereum/consensus-specs/pull/3034>
- [33] V. Buterin. 2018. RNG exploitability analysis assuming pure RANDAO-based main chain. <https://ethresear.ch/t/rng-exploitability-analysis-assuming-pure-randao-based-main-chain/1825> Accessed: 2024-11-12.
- [34] Toni Wahrstätter. 2023. Selfish Mixing and RANDAO Manipulation. <https://ethresear.ch/t/selfish-mixing-and-randao-manipulation/16081>. Accessed: 2024-11-12.
- [35] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. 2022. A flash (bot) in the pan: measuring maximal extractable value in private pools. In *Proceedings of the 22nd ACM Internet Measurement Conference*. 458–471.
- [36] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.
- [37] Aviv Yaish, Gilad Stern, and Aviv Zohar. 2022. Uncle Maker: (Time)Stamping Out The Competition in Ethereum. *Cryptology ePrint Archive*, Paper 2022/1020. doi:10.1145/3576915.3616674
- [38] Yaning Zhang and Youcai Qian. 2019. Randao: A DAO working as RNG of Ethereum. <https://github.com/randao/randao/>. Accessed: 2024-11-12.

A Deferred proofs

A.1 Proofs for forking attack strings

THEOREM (CONDITION FOR FORKING). *Given \mathbf{A}^{a_1} slots followed by \mathbf{HXA} , (where $\mathbf{X} \in \{\mathbf{A}, \mathbf{H}\}^{h-1}$, $a_1, h > 0$) \mathcal{A} can perform an ex-ante reorg with $0 < \alpha < 0.5$ stakes forking out \mathbf{HX} if*

$$a_1 \geq \frac{h(1-2\alpha) - p_{\text{boost}}}{\alpha} . \quad (9)$$

PROOF. When \mathcal{A} decides to fork the blockchain, the forking attack is successful if the sum of all the votes (sometimes attestations) on \mathcal{A} 's blockchain fork is larger than the sum of votes on the honest fork. The adversarial strategy entails the following steps:

- (1) \mathcal{A} secretly builds the first adversarial slots \mathcal{Y} , and a subset of its remaining $a_1 - 1$ slots. Additionally, all its validators vote at each slot on \mathcal{Y} .
- (2) Since the honest parties do not see the secret block(s) of \mathcal{A} , they build their blocks on top of the parent of block \mathcal{Y} and start a new blockchain fork. Every honest block in \mathbf{HX} is built on this fork, and honest validators vote on them.
- (3) \mathcal{A} proposes its block \mathbf{A} on top of the last hidden block following \mathbf{HX} , along with its secret fork and corresponding votes.

\mathcal{A} 's fork has $a_1\alpha + h\alpha + p_{\text{boost}}$ votes, while the honest parties' blockchain fork accumulated $(1-\alpha)h$ attestations. Honest validators will switch to \mathcal{A} 's fork if it has more votes, *i.e.*,

$$a_1\alpha + h\alpha + p_{\text{boost}} \geq (1-\alpha)h . \quad (10)$$

By rearranging Equation (10), and applying that $a_1 > 0$, we obtain the claimed inequality for a_1 . \square

As a corollary, we define the following quantity:

DEFINITION 8 (MINIMUM SLOTS TO FORK). *Let $\text{minFork}(\alpha, h)$ represent the minimum value of a_1 required for successfully ex-ante reorging h blocks given α stake. This function combines Theorem 1 and the condition that $a_1 \in \mathbb{N}^+$.*

$$\text{minFork}(\alpha, h) = \max\left(\left\lceil \frac{h(1-2\alpha) - p_{\text{boost}}}{\alpha} \right\rceil, 1\right) . \quad (11)$$

Next we prove Theorem 2: Given any $S \in AS_\alpha$, for a forking string $\mathbf{A}^{a_1}\mathbf{H}^h\mathbf{A}$ where Equation (4) holds, $\mathbf{A}^{a_1}\mathbf{H}^h\mathbf{AS}$ is also an attack string.

PROOF. Given $S \in AS_\alpha$ attack string, there is a possible RANDAO value $R^{e,*}$ that can be computed upon reaching the first slot of S . \mathcal{A} can first ex ante reorg \mathbf{H}^h by privately building \mathbf{A}^{a_1} , then building on top of it in the last block of the forking string. After \mathcal{A} can follow the same actions required to reach $R^{e,*}$ in S . \square

THEOREM. *For any forking eas of a states and observations $O \in \Omega$:*

$$U(s) = \max(\{N_\alpha(s), F_\alpha(s, a_1, O), \dots, F_\alpha(s, a_n, O)\}) , \quad (12)$$

where $\{a_1, \dots, a_n\} \subset A(s)$ are the actions where \mathcal{A} starts ex-ante reorging in a specific way.

PROOF. The set of actions $A(s)$ can be divided into two disjoint subsets: actions related to honest behaviour and actions related to ex-ante reorgs. As such,

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s') = \max(\{N_\alpha(s), F_\alpha(s, a_1, O), \dots, F_\alpha(s, a_n, O)\}) . \quad (13)$$

Note that $R(s) = 0$ in these states of the decision tree, as \mathcal{A} is rewarded (and punished for the sacrifice) at the end of epoch e . \square

B Additional objective functions for RANDAO manipulations

As we alluded to in Section 2.3, a RANDAO manipulator might be motivated by other objective functions than just maximizing the number of obtained slots. In particular, one objective function is the *target slot utility*, where the adversary is motivated to propose a specific slot in the next epoch. More formally, we define the target-slot reward function $\Gamma_\pi^{\text{tslot}}$ for validator i and the j^{th} slot in epoch $e + 2$ as:

$$\Gamma_\pi^{\text{tslot}} := I(\mathbf{v}_R^{e+2}[j] == i) , \quad (14)$$

where $I(\cdot)$ is an indicator variable.

As an extension, one might also consider a forward-looking objective where the validator aims only to maximize the number of blocks obtained in epoch $e + 2$ discounting any losses incurred by the manipulation strategy in epoch e , that is

$$\max_{R^e} \mathcal{P}(e + 2, R^e, \mathcal{A}) . \quad (15)$$

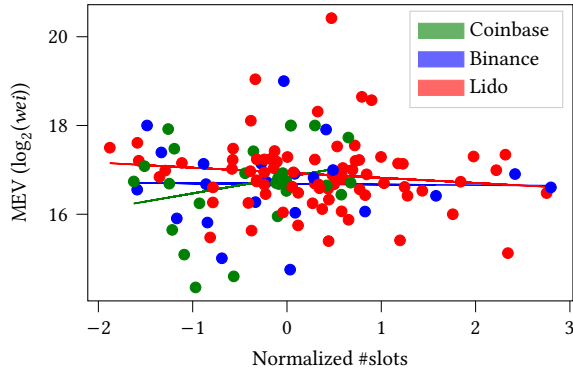


Figure 13: MEV incentives of three major validators whenever they ex-post reorged blocks at the end of an epoch. We normalised the slots obtained in the next but one epoch by the entities’ stake and the MEV of the reorging block. We see no correlation between the MEV content of the reorged blocks and the normalised RANDAO outcome.

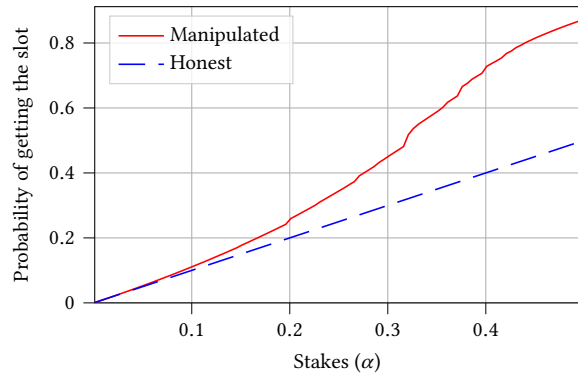


Figure 12: The expected value of obtaining a target slot for an honest player (blue) and a RANDAO manipulator using selfish mixing and forking strategies.

In addition, one might consider various other manipulation objectives. However, this work focuses on the utility functions defined

in Equations (3) and (14), disregarding any other incentives validators may have. We leave the exploration of different utility functions to future work.

B.1 Evaluating the target slot utility

We evaluated the manipulative power of an adversary that wishes to optimize for the target slot utility function, cf. Equation (14). In particular, we consider a model in which the adversary manipulates the RANDAO *once* in epoch e to obtain a specific slot, say the i^{th} slot in epoch $e + 2$. We evaluate the adversarial probability of this “one-shot” RANDAO manipulation to obtain the desired slot in Figure 12. We find that, for instance, for $\alpha = 0.3$, the adversary has a 0.451 probability of obtaining a desired target slot in epoch $e + 2$. We leave it to future work to study the target slot utility function in other, perhaps stronger adversarial models. For example, one could study the target slot utility in an adversarial model in which \mathcal{A} manipulates the RANDAO multiple epochs before the desired target epoch to enhance its manipulative power and increase its probability of obtaining the target slot.

C Additional empirical measurements

In this section, we provide measurements that we could not include in the main text due to space constraints.

C.1 Forking and MEV: are they related?

We collected ex-ante and ex-post reorgs executed by three major entities (*i.e.*, Lido, Coinbase, Binance) where they reorged a tail slot H_{30} or H_{31} . For each entity and tail slot reorg event H_A or $H_{\cdot A}$, we created a pair $(\mathcal{P}(e + 2, R^e, \mathcal{A}), f)$, where f is the amount the block builder paid to the validator in the block A . Since it is hard to assess the MEV content of a block, we use f as a proxy to approximate the MEV content of a block. We do not observe a significant correlation between the MEV content of the block A and the number of slots obtained by the ex-post forking entity in epoch $e + 2$, see Figure 13. The observed correlation coefficients are $-0.13, -0.02, 0.26$ for Lido, Binance, and Coinbase, respectively. This indicates that, as of the time of writing, ex-post forking decisions show no significant correlation with RANDAO manipulation considerations.