Time-manipulation Attack: Breaking Fairness against Proof of Authority Aura

Xinrui Zhang[♮]
University of Sydney[★]
Sydney, Australia
xzha5518@uni.sydney.edu.au

Rujia Li[♮]
Tsinghua University
Beijing, China
rujia@tsinghua.edu.cn

Qin Wang
CSIRO, Data61
Sydney, Australia
qinwangtech@gmail.com

Qi Wang[♭]
Southern University of Science and Technology[†]
Shenzhen, China
wangqi@sustech.edu.cn

Sisi Duan[♭]
Tsinghua University
Beijing, China
duansisi@tsinghua.edu.cn

ABSTRACT

As blockchain-based commercial projects and startups flourish, efficiency becomes one of the critical metrics in designing blockchain systems. Due to its high efficiency, Proof of Authority (PoA) Aura has become one of the most widely adopted consensus solutions for blockchains. Our research finds over 4,000 projects have used Aura and its variants. In this paper, we provide a rigorous analysis of Aura. We propose three types of *time-manipulation attacks*, where a malicious leader simply needs to modify the timestamp in its proposed block or delay it to extract extra benefits. These attacks can easily break the legal leader election, thus directly harming the fairness of the block proposal. We apply our attacks to a mature Aura project called OpenEthereum. By repeatedly conducting our attacks[1] over 15 days, we find that an adversary can gain on average 200% mining rewards of their fair shares. Furthermore, such attacks can even indirectly break the finality of blocks and the safety of the system. Based on the deployment of Aura as of September 2022, the potentially affected market cap is up to 2.13 billion USD. As a by-product, we further discuss solutions to mitigate such issues and report our observations to official teams.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**;

KEYWORDS

Proof of Authority, Fairness, Timestamp Attack, Aura

ACM Reference Format:
Xinrui Zhang[♮], Rujia Li[♮], Qin Wang, Qi Wang[♭], and Sisi Duan[♭]. 2023. Time-manipulation Attack: Breaking Fairness against Proof of Authority Aura. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3543507.3583252

[1] Attacks implemented and archived at https://doi.org/10.5281/zenodo.7627400

[♮] Equal contribution.
[♭] Corresponding authors.
[★] X. Zhang was affiliated with Southern University of Science and Technology.
[†] Full affiliation: Department of Computer Science and Engineering & Research Institute of Trustworthy Autonomous Systems & National Center for Applied Mathematics (Shenzhen), Southern University of Science and Technology.

1 INTRODUCTION

Proof of Authority (PoA), first proposed by Wood [1, 2], is a mainstream category of consensus algorithms for blockchain systems. PoA is inspired by Byzantine fault-tolerant (BFT) protocols [3], which allows a group of designated authorities to vote for the proposed blocks. As a result, Aura does not rely on heavy computational power and enables fast transaction confirmation. Many algorithms [4] have been proposed under the category of PoA. Among them, Aura (Authority Round) is the most featured one [5]. Aura and its variants have been widely used in multiple mature blockchain projects such as VeChain [6], Sokol [7], Kovan [8], and OpenEthereum (formerly Parity) [9]. However, despite its wide adoption, Aura has not rigorously been proven to be secure.

Aura assumes a synchronous network [4]. It divides the UNIX time into equal-length *step*s (also called *slot*s). In each step, only one authority (equiv. sealer [10, 11]) is elected as a leader to propose a new block, while other sealers verify blocks upon receiving the proposed block (see Figure 1.a). However, in practice, the synchronous assumption is not practical as unexpected network delay is very common. As a complementary mechanism, Aura allows sealers to postpone validating the delayed blocks (Figure 1.b). A delayed block, or equivalently a block with the *near future* timestamp, is still regarded as a part of the canonical chain that can be accepted as time elapses.

Unfortunately, after a careful investigation of the Aura protocol, we find that the validation deferral mechanism could be exploited as a loophole. By simply manipulating the block timestamp as a malicious leader (also called the *in-turn* sealer [10, 11]), an attacker can easily make other honest sealers fail to obtain any rewards. Specifically, a malicious leader may falsify his timestamp or delay his newly generated block to suppress the next sealer's block. Through this posterior occupation, the adversary can extract more mining rewards and transaction fees [12] beyond his fair share, thereby threatening the fairness of the system.

Following this connotation, we formalize our attack strategies and propose three time-manipulation attacks [13] that break the security goals of Aura protocol [11, 14]. We demonstrate that the Aura algorithm suffers from fairness vulnerabilities: a malicious in-turn sealer can always *deprive the privilege of block generation*

for the next sealer via producing a *falsified but legal*[2] block, which directly breaks *fairness* [11, 14, 15] property. By launching such an attack whenever becoming an in-turn sealer, an adversary can easily include a larger number of transactions in its proposed blocks than expected, and gain extra rewards without being noticed by other sealers.

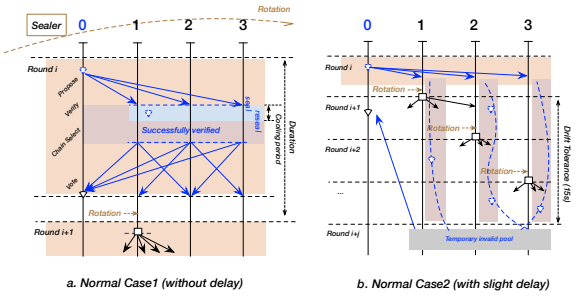


Figure 1: Aura operating mechanism

In fact, only ONE malicious sealer is sufficient to successfully launch the time-manipulation attacks. While one sealer's attack cannot directly destroy the entire network (making the system fail to achieve its security goals), such attacks may cause participating authorities to drop out due to failure to gain any rewards. In particular, if the attacker continues the attack over a long period, the victim authority will fail to gain any rewards as its proposed block will NEVER be finalized, making it lose the financial incentive to participate in the system. This may further lead to a massive dropping of out-of-network participants and jeopardize consensus stability (e.g., similar to stake bleeding [16]). An even worse scenario is that if f malicious Byzantine sealers simultaneously launch such attacks, where f is the maximum number of failures the system can tolerate, the proposed blocks of up to $2f$ sealers will never be finalized. Such an attack indirectly destroys the *deterministic finality* [17, 18] and *accountable safety* [19, 20], i.e., the security goals of the system.

To formalize our attacks and understand the impacts of our attacks, we provide formal definitions of fairness and conduct a theoretical analysis of our attacks. We first clarify the notion of *fairness* by disconnecting its fuzzy definition into fine-grained subterms that are driven from its *transaction-*, *block-* and *leadership-*perspectives. Then, we formally prove that our attacks can effectively harm all the defined fairness properties. For the attack of each malicious sealer, the victim sealer cannot insert its block at a regular position on-chain, and the block it proposes will never be confirmed. Our theoretical analysis also details the root cause of our attacks. Finally, we delineate each strategy by showing its intrinsic mechanisms and actual damages.

To evaluate the aforementioned damages in practice, we design, implement and evaluate these attacks by conducting experiments on top of OpenEthereum, a widely adopted Aura implementation. We fork the main branch, modify the code and compile them into 3 executable files. With the help of our analysis tool, we run them in an isolated environment with 21 nodes for experiments. By repeatedly conducting three types of attacks over 15 days, all of our attacks successfully achieve their goals: the attacker can always frontrun other sealers' chances of being the leader. Under the second and third strategies, attackers get up to 200% of their fair mining reward. These results prove the feasibility and effectiveness of our attacks. We have made the aforementioned tools and experimental data publicly available (see Appendix C.1). As of September 2022, over 4,000 projects[3] are using Aura and the potentially affected market cap of our attacks is up to 2.13 billion USD[4].

We summarize our contributions in the following.

- We present a family of *time-manipulation attacks* (three strategies, Sec.3), where only one attacker is sufficient for extracting extra rewards than its fair share.
- We introduce fine-grained fairness definitions from different perspectives of blockchains. We theoretically prove that our attacks can successfully break Aura's fairness (Sec.3.4), deterministic finality and accountable safety (Appendix B.3).
- By launching attacks (Sec.4) for over 15 days on OpenEthereum, a widely adopted Aura implementation, our attacks successfully frontrun the victim leading sealers' chances (Sec.4.2).
- We explore the root reason and the impacts of the attacks (Sec.5) and provide our countermeasures to mitigate such issues (Sec.6).

2 POA AURA

System model. Aura assumes a synchronous network, i.e., if a correct node sends a message at the time t, it will be received by a correct node no later than time $t + \delta$. Aura grants the block mining right to a committee of authorized nodes (i.e., authorities). Each authority maintains a local list of elected authorities that are capable of proposing and validating blocks in a pre-defined order. In this paper, we use the term *sealers* [10, 11, 21] and authorities interchangeably. In Aura, time is divided into consecutive *steps* (denoted as s). Each step is based on a fixed length, called *step duration*, denoted as d. In each step, one of the sealers is assigned as the leader (a.k.a. in-turn sealer [11]). The leader has the highest priority of proposing blocks in the current round. Aura assumes $n = 2f + 1$ sealers, where f is the maximum number of Byzantine nodes, matching the lower bound for synchronous BFT [17].

The protocol. Aura protocol consists of four phases: *block proposal*, *block confirmation*, *fork choice mechanism*, and *voting mechanism*. We present the core procedures in each phase in Algorithm 1.

Block proposal. Each node is aware of the current step as the steps are related to physical clocks (line 3, Algorithm 1). At each step, a leader will be assigned (line 3–5, Algorithm 1, where the elected leader is indexed with *id* on the sealer list) for issuing a block. It is misbehavior to propose more than one block per step or to propose a block out of turn [4]. New blocks record the timestamps when they are generated (line 7, Algorithm 1), and the accumulated difficulties (abbr. *diff*) of their ancestors in the path to the genesis

[2]Here, the term '*falsified*' means the block is embedded with an incorrect timestamp, whereas '*legal*' means that the block is aligned with original code specifications such that it will still be regarded as valid.

[3]Aura is generally used as a pluggable algorithm embedded in mainstream implementations. At the time of writing, Github projects using these implementations are statistically listed as follows: Nethermind (200+), Openethereum (340+), ParityEthereum (1700+) and Substrate (2400+). Here, the notation "+" represents *forked by*.

[4]Data is captured on Sep. 15th, 2022 from *CoinMarketCap*, including the market shares of VeChain, PoA Network, Gnosis Chain, etc.

Algorithm 1 The Aura algorithm

```
1:  procedure BLOCK_PROPOSE(sealer_i)
2:      while (true) do
3:          s ← now/step_duration
4:          id ← s  mod |sealers|                          ▷ leader index
5:          block.sig ← sealer_i
6:          if (sealer_i = sealer_id ∧ sealer_i ∈ sealers) then
7:              block.timetamp ← TS(lastblock.timestamp)
8:              block ← sign(TXs, weight)                  ▷ seal a block
9:              broadcast(block)
10:         sleep(step_duration)                           ▷ wait for next step
11:     end while
12: end procedure
13: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
14: procedure BLOCK_VERIFY(block_received)
15:     block ← block_received
16:     max_time ← now + acceptable_drift
17:     if ReceiveTime(block) − now > receive_threshold then
18:         return (Too Late)                              ▷ block discard
19:     if block.timestamp > invalid_threshold then
20:         return (Invalid)                               ▷ malicious block
21:     if block.timestamp > max_time
22:         return (Temporarily Invalid)                   ▷ wait a while
23:     route ← FORK_CHOICE(block, current_route)
24:     view, flag ← VOTE_FOR_FINALITY(block, route)
25:     broadcast(view, flag)      ▷ broadcast confirmed local view
26: end procedure
27: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
28: function FORK_CHOICE(block, current_route)
29:     parent ← block.lastblock
30:     block.diff ← parent.step − block.step + constant
31:     new_route.diff ← block.diff + parent.diff
32:     return (max(diff(new_route, current_route)))
33: end function
34: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
35: function VOTE_FOR_FINALITY(block, route)
36:     sig ← Sign(route)
37:     view ← Merge(sig, block.sig)
38:     flag ← (Num(block.sig) > |sealers|/2)
39:     return view, flag
40: end function
```

block $b_{\mathcal{G}}$. If a correct leader does not have pending transactions, it is supposed to propose an empty block.

Block confirmation. Non-leader sealers executes block verification procedure (line 14) once they receive a new block. Several values in block headers are verified here and the total *diff* of the new block is measured. Particularly, the timestamp value of a block should be checked (line 19–20) to prevent accepting blocks unreasonably far in the future (line 21–22). A block will be verified only if it is proved to be valid and is chosen under a *fork choice mechanism* when forks occur. A confirmed block will be regarded as a part of the canonical chain, and the block header with the path ending with it (referred as to $P[b_0, b_1 \dots b_K]$) will be broadcast (line 25).

Chain selection. Aura adopts a simplified GHOST [22] protocol to solve forks. The mechanism is a variant of GHOST [23] and Bitcoin's longest-chain [24]. According to technical specifications [25], when a fork (subtree) exists, the path with the most intensive computing power will be identified as the *heaviest* branch and be accordingly merged into the canonical chain. The concept of *total difficulty* evaluates how much computing power has been invested in a block. Aura adopts a similar mechanism, but each block's difficulty is related to the current step number and the difficulty of the chain of blocks led by the proposed block. Briefly, the current block's difficulty is calculated as follows: the total difficulty (*diff*) of a block is calculated recursively by adding its parent's total difficulty ($diff_{\mathcal{P}}$) and its header difficulty ($diff_{\mathcal{H}}$). A block proposed in a lower step has a higher header difficulty (Equation 4) and thus is likely to be accepted by the canonical chain.

Voting mechanism. Aura is a hybrid consensus protocol that combines both probabilistic finality and deterministic finality [4]. The voting mechanism aims to provide deterministic finality instantly. Each sealer in Aura votes for a proposal on the longest common prefix, evaluated via a $\mathcal{F}(\cdot)$ function. In short, the longest common prefix denotes a chain of proposed blocks with the highest difficulty. In each step, the in-turn sealer issues a block extending the longest common prefix it is aware of, which is not necessarily the chain led by the block proposed by the previous in-turn sealer. Then, the in-turn sealer signs this newly proposed block and broadcasts it to other sealers (i.e., the block proposal phase). According to the synchrony assumption, honest sealers will eventually receive the signed block from a correct in-turn sealer within the step duration and then evaluate whether this is the longest common prefix they are aware of. If over half of the sealers confirm the prefix, they will vote for it when they become in-turn sealers, i.e., this proposed block together with all its ancestors on the prefix will be finalized.

3 TIME-MANIPULATION ATTACKS

In this section, we first provide a set of definitions of *fairness*, aiming at defining the term generically from different perspectives of blockchains. Then, we present three concrete strategies, each targeting different levels of fairness.

3.1 Fairness in Aura

We define the notion of *fairness* in each operating layer by considering transactions, blocks, and committee rotation and combine them to deliver a complete definition of the protocol fairness.

$\mathcal{T}$-fairness. The transaction-level fairness adopts the definition of *receive-order-fairness* [14][15][26]. A transaction Tx_i received at the time T_i, with sufficiently many witnesses (or confirmations), should always appear before the transaction Tx_j at the time T_j in the final chain where $\mathsf{T}_i < \mathsf{T}_j$.

$\mathcal{B}$-fairness. The block-level fairness [27] emphasizes the correct *receive-order* of blocks. Much similar to *$\mathcal{T}$-fairness*, a block b_i with the timestamp T_i, confirmed by sufficiently many validators, should always be allocated in the front of the block b_j with the timestamp T_j in the final chain where $\mathsf{T}_i < \mathsf{T}_j$.

$\mathcal{L}$-fairness. Apart from *$\mathcal{T}$/ $\mathcal{B}$-fairness*, the leadership-order fairness [28] refers to the legal order of committee members: a leader

candidate with the index x should always, by following the rotation formula $f(\cdot)$, have more advantages in becoming the leader than the candidate with the index y where $f(x) < f(y)$.

Π-*fairness*. The protocol (Π)-level fairness covers the above three types of fairness: when achieving all $\mathcal{T}/\mathcal{B}/\mathcal{L}$-fairness, the protocol can be deemed as reaching Π-fairness.

3.2 Time Features

Aura follows a *time-based block proposal* scheme, where leader election is related to physical clocks. Based on this fact, we highlight two key features, *drifting tolerance* and *cooling period*. As we show later on, our attacks are tightly related to these two features.

Drifting tolerance. Sealers start to verify the timestamp when they receive a new block. In a real system, benign time-drifting could be common. To improve the practicality, Aura sets a drifting tolerance. For example, in OpenEthereum [9], a block holding the timestamp within 15 seconds will be accepted instantly. If the timestamp is over future 15 seconds but no more than 150 seconds, it will be marked as *temporarily invalid* and be stayed in the block queue for later verification until requirements are met. Otherwise, it will be considered invalid and discarded immediately.

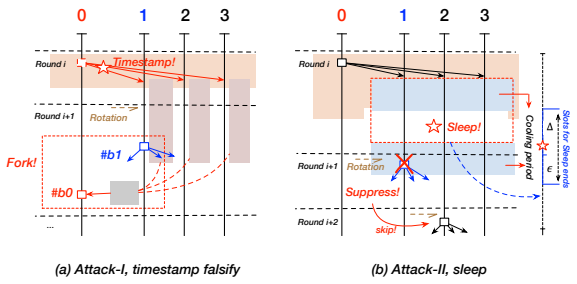
Cooling period. After a sealer receives a block at T_0, it first sends a vote message to other sealers at T_1. Then, it starts to propose a new block at T_2 if it is selected as a leader for this step. We call the time interval between T_1 and T_2 as the *cooling period* Δ. Obviously, $\Delta = \mathsf{T}_2 - \mathsf{T}_1$. It represents the *smallest* duration between voting and a new block production (blue slot in Figure 1). The mechanism is designed to prevent sealer racing when accessing pending blocks.

3.3 The Attacks

The core idea of our attack is based on the *malicious time modification*. We outline three different types of attack strategies. Each strategy maliciously modifies the timestamp of a proposed block in a different way. Commonly, all strategies are conducted from the attackers' side, where the attacker changes the timestamp assignment functions from the original source code and runs the malicious program $\mathsf{P}_{\mathcal{M}}$ on its local machine. For the rest of this section, we show that by controlling only one sealer, a malicious adversary can successfully launch the attacks. Here, we denote the timestamps of parent blocks by $\mathsf{timestamp}_{\mathcal{P}}$.

Attack-I: timestamp falsify. To launch a timestamp falsify attack, an adversary only needs to *falsify the timestamps of his newly proposed blocks* (see Figure 2.a). In normal cases, the original timestamp $\mathsf{timestamp}_{\mathcal{O}}$ of a leader's block is the system time when the timestamp is assigned, which should be equal to $\mathsf{timestamp}_{\mathcal{P}} + \mathsf{d}$. In the attack case, the timestamp of an attacker's block becomes $\mathsf{timestamp}_{\mathcal{P}} + \mathsf{d} + \eta$, where η is a valid deferment. Through this way, the adversary's block $\#b_0$ cannot be the *immediately verified* after being received, and it will be marked as "temporarily invalid". With such a strategy, the next in-turn sealer's block $\#b_1$ will be discarded due to *relatively less difficulty* (calculated by Equation (3)).

Attack-II: sleep delay. To launch this attack, we let an adversary delay block generation and propagation, making the next sealer's cooling period override the block production time so that the incoming normal block production will be suppressed. In particular,



(a) Attack-I, timestamp falsify (b) Attack-II, sleep

Figure 2: Attack strategies (I/II)

the second strategy requires *inserting a single line of sleeping instruction* during the malicious sealer's block production process. In normal cases, a block is generated after the cooling period, and the cooling period can be covered by the duration slot. However, in our attack, the adversary sleeps for a while before generating a block, making the next in-turn sealer delay the block receiving and further suppressing the block production process of the next in-turn sealer. To be specific, to ensure our attack can halt the generation of the next in-turn sealer's block, we let the cooling period Δ override the time when the next in-turn sealer should seal a block.

Namely, we require sleep time $\mathsf{D}_{\mathcal{A}} > \mathsf{d} - \Delta$. Additionally, to avoid missing the attacker's legal step, $\mathsf{D}_{\mathcal{A}}$ is also required to be less than $\mathsf{d} + \epsilon$, thus

$$\mathsf{d} - \Delta \leq \mathsf{D}_{\mathcal{A}} < \mathsf{d} + \epsilon, \tag{1}$$

where ϵ is a short drifting period to tolerate the network latency.

Attack-III: timestamp falsify & sleep delay. The third strategy is *a hybrid scheme* integrating the former two methods. An adversary first sleeps for a while, and then, assigns a profitable $\mathsf{timestamp}_{\mathcal{M}}$ to his newly generated block. In this way, $\mathsf{timestamp}_{\mathcal{M}}$ can be more *subtle* (hard to detect) than that in the first strategy, since no "temporarily invalid" is reported. The attack only needs to satisfy the condition that,

$$\begin{cases} \mathsf{timestamp}_{\mathcal{P}} < \mathsf{timestamp}_{\mathcal{M}} \\ \mathsf{timestamp}_{\mathcal{M}} \leq \mathsf{timestamp}_{\mathcal{P}} + \mathsf{d} + \mathsf{D}_{\mathcal{A}} + \eta. \end{cases} \tag{2}$$

Notably, the manipulation can be profitable when using specialized types of smart contracts [29], which is demonstrated with an example in Appendix B.1.

In summary, all three attacking strategies create *fairness* issues that cover all the leadership, block, and transaction levels. This is rooted in the fact that *a malicious leader can propose out-of-step (but legal) blocks*. Specifically, in *Attack-I*, an adversary delays the validation of its proposed block to the next step, competing with the following innocent block. The block proposed by the next in-turn sealer will therefore not be confirmed. In *Attack-II*, an adversary delays the normal block issuance. While sleeping for a period of time, as we mentioned, the next sealer will miss the chance to issue a new block. This deprives the legal block proposing rights of incoming sealers. Meanwhile, during the delay, the adversary obtains additional time to include more transactions from the pool, gaining more transaction fees as extra revenues. Similar to *Attack-II*, an adversary in *Attack-III* can make the next in-turn sealer's

proposed block fail to be confirmed. Furthermore, this adversary can gain extra rewards.

3.4 Theoretical Analysis

In this section, we provide a formal analysis to theoretically prove the feasibility of our attacks.

Theorem 3.1. *Assume that $\mathcal{B}_{\mathcal{P}}$ is an arbitrary block that has reached deterministic finality at the time $\mathsf{T}_{\mathcal{P}}$, $\mathcal{B}_i$ and $\mathcal{B}_j$ are two following consecutive blocks with $\mathsf{T}_i < \mathsf{T}_j$, under our Attack-I, $\mathcal{B}_j$ is never allocated after $\mathcal{B}_i$, namely, Aura cannot achieve $\mathcal{B}$-fairness.*

Proof. In step s_i, the timestamp of the attacker's block exceeds drifting tolerance, making the newly created block $\mathcal{B}_i$ marked as *temporarily invalid*. Under the *round-robin* rotation, if a leader $\mathcal{L}_i$ is believed to be failed, the next leader $\mathcal{L}_j$ simply moves on to the next step s_j, and generates another block $\mathcal{B}_j$. Suppose that *Aura* protocol reaches $\mathcal{B}$*-fairness*. That means the block $\mathcal{B}_j$ is chained after the block $\mathcal{B}_i$, namely $\mathcal{F}(\mathcal{B}_j) > \mathcal{F}(\mathcal{B}_i)$. However, this ***contradicts*** to the chain selection mechanism, where the higher total difficulty should be accepted by the canonical chain. Observe that,

$$\mathcal{F}(\mathcal{B}_\star) = diff_{\mathcal{P}} + diff_{\mathcal{H}}, \tag{3}$$

$$diff_{\mathcal{H}} = \mathsf{s}_{\mathcal{P}} - \mathsf{s}_{\mathcal{H}} + \mathsf{fixed\_parameters} \tag{4}$$

We have known that $\mathsf{s}_i + 1 = \mathsf{s}_j$, and both $\mathcal{B}_i$ and $\mathcal{B}_j$ share the same parent block $\mathcal{B}_{\mathcal{P}}$, making that $\mathcal{F}(\mathcal{B}_j) < \mathcal{F}(\mathcal{B}_i)$.

□

Theorem 3.2. *Assume that adjacent $\mathcal{S}_x, \mathcal{S}_y$ and $\mathcal{S}_z$ are three arbitrarily neighbour sealers with the rotation rule $f(\mathcal{S}_x) < f(\mathcal{S}_y) < f(\mathcal{S}_z)$, under our Attack-II/III, $\mathcal{S}_y$ will never become a valid sealer after $\mathcal{S}_x$'s attack, and $\mathcal{S}_x$ is always followed by $\mathcal{S}_z$, namely, the Aura protocol cannot achieve $\mathcal{L}$-fairness.*

Proof. Suppose an attacker $\mathcal{S}_x$ delays $\mathsf{D}_{\mathcal{A}}$ after its step begins at T_x and $\mathcal{S}_y$ starts to generate block at T_y. In the normal case, $\mathsf{T}_y = \mathsf{T}_x + \mathsf{d}$. Given our attacking strategy $\mathsf{D}_{\mathcal{A}} \geq \mathsf{d} - \Delta$, and $\mathsf{D}_{\mathcal{A}} < \mathsf{d} + \epsilon$, making $\mathcal{S}_x$ start to generate blocks at $\mathsf{T}_x{}'$ where $\mathsf{T}_x{}' = \mathsf{T}_x + \mathsf{D}_{\mathcal{A}}$, so that

$$\mathsf{T}_x + \mathsf{d} - \Delta \leq \mathsf{T}_x{}' < \mathsf{T}_x + \mathsf{d} + \epsilon. \tag{5}$$

Since there is a cooling period Δ before $\mathcal{S}_y$ proposes a block, T_y is required to be greater than $\mathsf{T}_x{}' + \Delta$. Considering the minimum value of $\mathsf{T}_x{}'$ according to Equation (5),

$$\mathsf{T}_y > \mathsf{T}_x{}' + \Delta \geq \mathsf{T}_x + \mathsf{d} - \Delta + \Delta = \mathsf{T}_x + \mathsf{d}. \tag{6}$$

Assume that Aura reaches $\mathcal{L}$*-fairness*, the rotation order should be $\mathcal{S}_x, \mathcal{S}_y$, and $\mathcal{S}_z$. To achieve this goal, T_y should never miss the moment $\mathsf{T}_x + \mathsf{d}$ for successful block generation. However, it ***contradicts*** to Equation (6). Hence, the theorem is proved.

□

Theorem 3.3. *Assume that Tx_i is received before Tx_j, under our Attack-I/II/III, Aura can not promise that Tx_i is positioned in front of Tx_j, namely, the Aura protocol cannot reach $\mathcal{T}$-fairness.*

Proof. (Sketch) If $\mathcal{T}$*-fairness* is achieved, then Aura protocol must satisfy $\mathcal{B}$*-fairness*. However, *Attack-I* cannot reach $\mathcal{B}$*-fairness*. *Attack-I/II* cannot reach $\mathcal{L}$*-fairness*, causing them to fail to satisfy $\mathcal{B}$*-fairness*. Thus, the proof is concluded. □

4 RUNNING ATTACKS AGAINST OPENETHEREUM

To demonstrate the feasibility of our proposed attacks, we launch attacks on a widely adopted project, OpenEthereum [9].

4.1 Malicious Programs

Experiment configuration. We build a test network on a server with Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-109-generic x86_64) operating system. All nodes are locally deployed and connected to different ip/ports, simulating a globally distributed system. Our simulation environment consists of 21 sealer nodes, which are indexed from 0 to 20 (e.g., $node_0$ to $node_{20}$). Each node has one sealer account responsible for block generation. We set the step duration d as 5 seconds. In this setup, sealers take turns to propose blocks every 5 seconds regardless if it is an empty block (a block with no transactions). We also deploy 5 client nodes and let them continuously send transactions to simulate sync nodes' activities in the real world.

Experimental procedure. Under our configuration, the nodes will propose and propagate blocks according to the round-robin scheduling of $\{node_0, node_1, node_2, ..., node_{20}, node0\}$ repeatedly in every 5 seconds. We also require the client nodes to constantly send transactions at a fixed rate of 400 transactions per minute. In total, we run the system for 45, 725 minutes and obtain 9, 450 blocks (450 rounds). Our Python-based analysis tool establishes an automatic procedure for both initiating the attack and processing raw experimental data. We observe the chain state for a relatively sufficient time and accordingly record the current state as a normal case. In our experiments, the attacker is set to be a malicious sealer with the intention of breaking the order of block generation. To simulate three different attacks, we let $node_0$ be the adversary node, executing each of the three attack strategies. To implement these malicious programs, we fork the original project and falsify their code in three different ways. Especially, we modify the function *fn new()* (Appendix C.1) in block proposing procedure as follows.

Attack-I program. In the original source code, the timestamp of a new block should be assigned the current system time $\mathsf{timestamp}_{\mathcal{O}}$. In our first strategy, we falsify such a timestamp as $\mathsf{timestamp}_{\mathcal{O}} + \mathsf{d} + \eta$, where η is the 15-second valid deferment. Practically, we modify the function *fn new()* in original source code (line 4–5, Algorithm 2) and then compile it for the malicious program $P_{\mathcal{M}1}$. To launch the attack, we let $node_0$ run $P_{\mathcal{M}1}$ while all other honest nodes execute the program $P_{\mathcal{O}}$.

Attack-II program. For the second strategy, we add a piece of code segment to sleep the program for 3 seconds during block generation, namely, $\mathsf{D}_{\mathcal{A}} = 3$ where $\mathsf{D}_{\mathcal{A}} < \delta$. The newly generated block thereby will not be broadcast until the sleep ends. We compile the falsified codes (line 6–8, Algorithm 2 in Appendix C.2) and obtain the malicious $P_{\mathcal{M}2}$. We run $node_0$ as $P_{\mathcal{M}2}$ while honest nodes execute the program $P_{\mathcal{O}}$.

Attack-III program. This strategy combines the former two falsifications. On the attacker's side, we sleep the program for 3 seconds during the block generation and assign its timestamp by serial values in a threshold t, where $\text{timestamp}_{\mathcal{P}} + 1 < t < \text{timestamp}_{\mathcal{P}} + 23secs$, here $(23 = \text{d} + \eta + \text{D}_{\mathcal{A}} = 5 + 15 + 3)$. We repeatedly conduct experiments with different timestamp modifications (e.g., $\text{timestamp}_{\mathcal{P}}$ + 1secs, $\text{timestamp}_{\mathcal{P}}$ + 20secs, $\text{timestamp}_{\mathcal{P}}$ + 23secs) and denote all programs compiled from these codes (line 9–11, Algorithm 2) as $P_{\mathcal{M}_3}$. We assume that the attacker $node_0$ executes $P_{\mathcal{M}_3}$ while honest nodes run $P_{\mathcal{O}}$.

4.2 Results

We observe the tracing logs of each node and the header information of confirmed blocks. In the normal case, block proposers follow the rotation order from $node_0$ to $node_{20}$. That means, after receiving $node_1$'s block, $node_2$ should take this block as its parent block and then proposes a new block based on this ancestor. All the authorities have an equal probability of generating blocks. In contrast, in attacking cases, a malicious block proposer can successfully deprive the block generation right of the subsequent in-turn candidate. In particular, in all three attacks, block proposers rotate in the order of $\{node_{20}, node_0, node_1, ..., node_{20}\}$ repeatedly. For consistency, we still assume the block generated by malicious $node_0$ as $\#b_0$, and the blocks generated by honest $node_1$ as $\#b_1$. The result indicates that $\#b_0$ are always followed by a normal $\#b_2$ created by $node_2$, and $\#b_1$ of $node_1$ is continuously vacant (see Figure 3.b), breaking the fair order of $node_1$ in normal cases. Figure 4 gives a wider range of block loss depending on multiple tamper combinations. Experimental results are as follows and Appendix C.3.

Attack-I results. In block generation stage, the attacker $node_0$ successfully generates $\#b_0$. Then the victim $node_1$ also successfully creates $\#b_1$. Both blocks are verified eventually, even if $\#b_0$ is recorded as *temporarily invalid*. However, at the confirmation stage, only $\#b_0$ can be accepted, and $\#b_1$ will be ultimately discarded, which proves the effectiveness of our attack. This situation directly forfeits the victim $node_1$'s profits. Normally, as we continuously create transactions at a stable rate, the overall confirmed transactions included in each block are stable (see the black dotted line in Figure 3.a). However, since $\#b_1$ has been discarded, transactions in it are no longer alive. In this case, we observe that $\#b_0$ did not obtain extra profits, but $\#b_2$ gains more profits than unusual since more transactions are included in its block (the green dotted line of Method 1 in Figure 3.a).

Attack-II results. For *Attack-II*, no errors are found by tracing the logs of block generation. All blocks can be instantly confirmed once they are received by sealers. However, $node_1$ cannot propose blocks even within his turn, which means $\#b_1$ has never been created. We notice that $node_1$'s log with the statement of "reseal too early" is followed by the termination of $node_1$'s block generation, meaning that the cooling period has overridden $node_1$'s time slot of block creation. In this way, $node_0$ can extract extra revenues by collecting more transactions within his block, where no penalty is applied. The number of transactions amount will rise as the sleeping period extends (cf. the second sub-graph in Figure 3.a).

Attack-III results. For *Attack-III*, we falsify $node_0$'s timestamp when it proposes $\#b_0$. Meanwhile, we also make the program sleep for 3 seconds. However, when compared with *Attack-I*, no error has been reported in the tracing log while $\#b_1$ was not proposed, either. The block generation process is also aborted by logging "reseal too early". We conduct a series of experiments and present transactions per block of both $\#b_0$ and $\#b_1$ in the third folding line chart in Figure 3, which is similar to the results of *Attack-II*. $\#b_0$ can collect more transactions than other blocks due to its longer transaction processing time.

5 ANALYSIS AND IMPACT

We develop this section by providing an in-depth discussion of the root reasons that make our attacks practical. Then, we analyze the potential impacts of the proposed attacks.

5.1 Root Cause

Attack-I, analyzed. The first method is conducted by making use of timestamp drift tolerance and the forking mechanism. In particular, the tolerance allows honest sealers to accept a *pending* block as long as it is within 15 seconds from the verification moment, regardless of whether the timestamp is tempered or benign. Secondly, the forking mechanism provides an advantage of acceptance for the block with a smaller step s (cf. Equation (3)). Block verification is completed within a rather short period after it is received, particularly in the same second as acceptance in our observed cases. Therefore, the proposed block $\#b_0$ is supposed to be verified at $\text{timestamp}_{\mathcal{O}}$. However, as we set $\text{timestamp}_{\mathcal{M}}$ to $\text{timestamp}_{\mathcal{O}} + \text{d} + 15\text{secs}$ (or more but less than $\text{timestamp}_{\mathcal{O}} + \text{d} * 2 + 15\text{secs}$), $\#b_0$ will be marked as *temporary invalid* immediately when being received by other sealers and will stay in the waiting queue for later processing until it can meet at least the upper bound of valid timestamp threshold (15 seconds ahead of the verification moment).

Since we offset the $\text{timestamp}_{\mathcal{M}}$ by d + 15secs from the actual timestamp, the successful verification of $\text{timestamp}_{\mathcal{M}}$ is delayed to the next step. As d elapsed, $\#b_0$ can be re-activated and verified. However, the next sealer's turn begins at the same time. The new sealer will not consider $\#b_0$ as its parent since it is not confirmed yet. He will create a new block $\#b_1$ to fill the gap. As a result, $\#b_0$ and $\#b_1$ both exist in the network with the same parent, and a fork occurs (see the red square in Figure 2.a). At this point, the sealers need to evaluate these two new valid blocks and choose one to merge into the canonical chain. In normal cases, blocks arrive/leave the pending pool after validation. In the attacks, $\#b_0$ gets stuck in the queue until $\#b_1$ arrives and gets verified, creating a fork.

Equation (4) defines the header difficulty $diff_{\mathcal{O}}$ of a block. The decisive components are $\text{s}_{\mathcal{P}}$ and $\text{s}_{\mathcal{H}}$. According to Aura's polling rule, $\#b_0$ is one step *ahead* from $\#b_1$, which means that ($\mathcal{H}$ is header):

$$\text{s}_{\mathcal{H}_{\#b_1}} = \text{s}_{\mathcal{H}_{\#b_0}} + 1. \tag{7}$$

And therefore,

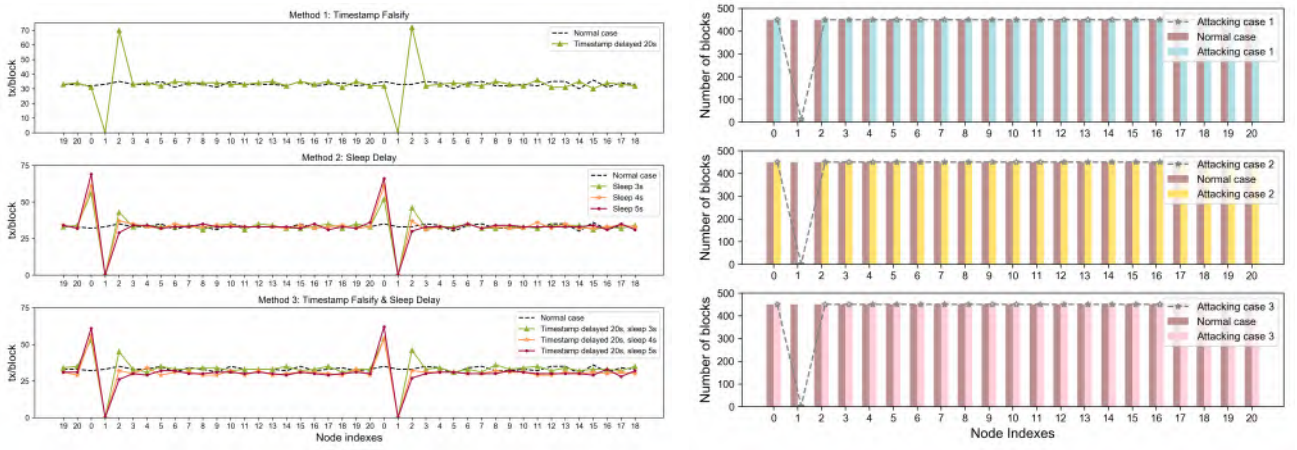
$$\begin{cases} diff_{\mathcal{H}_{\#b_0}} = \text{s}_{\mathcal{P}} - \text{s}_{\mathcal{H}_{\#b_0}} + \text{fixed\_parameters} \\ diff_{\mathcal{H}_{\#b_1}} = \text{s}_{\mathcal{P}} - (\text{s}_{\mathcal{H}_{\#b_0}} + 1) + \text{fixed\_parameters} \end{cases} \tag{8}$$

so that,

$$diff_{\mathcal{H}_{\#b_1}} = diff_{\mathcal{H}_{\#b_0}} - 1. \tag{9}$$

Since they share the same parent,

$$diff_{\mathcal{P}_{\#b_1}} = diff_{\mathcal{P}_{\#b_0}}, \tag{10}$$



(a) The number of transactions per block of each node in one polling round under the proposed attacks.

(b) The number of blocks proposed by each node under the attacks where the system generates 9, 500 blocks in total.

Figure 3: Attacking results on transaction and block levels across participant nodes

based on Equation (3), the total difficulty of the two blocks

$$diff_{\#b_1} = diff_{\#b_0} - 1. \tag{11}$$

With higher total difficulty *diff*, the attacker's block wins the forking competition against the next generated bock and thus deprives the next sealer of the right to propose a block within this turn. The attacker successfully frontruns the next legal candidate.

Attack-II, analysed. The key point of the second method is to make use of the cooling period between consecutive block proposals (implemented by *reseal_min_period*). Figure 2.b depicts this attacking strategy in the red square box. The white zone represents the attacker's sleeping period, while the light-blue zone is the short period between `reseal` operations. As introduced in Sec.2, a minimum period between transaction-inspired reseals is required as the *cooling period* (cf. Sec.3.2) in case of enormous empty blocks. The `seal` and `reseal` operations are triggered each time when the sealer proposes a new block or verifies a received block. In the practice of OpenEthereum, this period is represented by *reseal_min_period* and set to 2 seconds by default. When an adversary seals a block, *next_allowed_reseal* will be reset by adding *reseal_min_period* to the current time. Then the next reseal will not happen until *next_allowed_reseal*.

The mechanisms leading to $\#b_1$'s failure can be divided into two general cases. In the first case, if the sleeping period is longer than or equal to $d - \Delta$ (see Equation (1)) but less than d (sleep ends within the Δ range in Figure 2.b), the next sealer could miss its proposing chance because the *reseal_min_period* has not elapsed. The delayed verification of $\#b_0$ will occupy the valid slot of $\#b_1$'s. In this case, $\#b_0$ will be successfully confirmed by non-victim sealers, whereas the incoming sealer cannot propose any block. In the second case, if the sleeping period is longer than or equal to d but less than $d + \epsilon$ (the ϵ range), the incoming sealer's inner step s will calibrate to $s + 1$ before the sleeping ends. Therefore, the next sealer will start to propose $\#b_1$ and thereby create a fork when $\#b_0$ and $\#b_1$ are queued at nearly the same time. According to the chain selection rule (Sec.2), $\#b_0$ will be accepted whereas $\#b_1$ is abandoned.

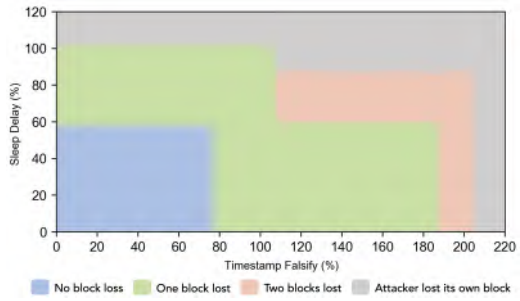


Figure 4: Block loss under different combinations of *sleep delay* & *timestamp falsify*, where the coordinates indicate the proportion of the *delay time* & *falsified timestamp* to the step duration.

Attack-III, analysed. The third method combines the previous two attacks. The mechanism behind this method is similar to *Attack-II*. $\#b_1$ might confront the failure of either being proposed or discarded due to the forking mechanism. This depends on the length of the sleeping period. The only difference from *Attack-II* is the timestamp that can be manipulated. The manipulation range depends on the length of the sleeping period. The timestamp earlier than $\text{timestamp}_{\mathcal{P}} + 1\text{sec}$ is considered to be invalid. As discussed in Sec.3, the timestamp within 15 seconds of validation is acceptable. Since the sleeping period delays $\#b_0$'s propagation and validation for $\mathrm{D}_{\mathcal{A}}$, the upper limit of valid timestamps becomes $\text{timestamp}_{\mathcal{O}} + 15\text{secs} + \mathrm{D}_{\mathcal{A}}$. Moreover, if the timestamp is earlier than the honest time within the valid range or delayed less than $15\text{secs}+\mathrm{D}_{\mathcal{A}}$, other nodes will not detect this misbehavior and report the error (e.g., the block cannot be marked as *temporarily invalid*).

5.2 Discussion

Difficulty of launching our attacks. The proposed attacks can be conducted by one single node. As for permissioned networks, the greedy authority is more likely to become malicious for seeking more economic benefits. Therefore, the only barrier to conducting the attack is to become a sealer, in which the code falsification process can be considered costless. Beyond that, we found such attacks are hard to be detected. A *rational and dynamic* adversary may not seek to directly destroy the consensus stability (safety or liveness). Instead, he aims to pursue optimal profits for the long-term running. Obviously, if a malicious sealer always behaves in an abnormal way, he will be easily caught or kicked off. To avoid such troubles, he can premeditatedly deviate from the protocol specification for a short period of time while honestly following the specification most of the time. This rational strategy is practical in real scenarios for avoiding punishment, since distinguishing between intentional or unintentional block delay is challenging.

Attacks on other Aura projects. Although we present the analyses by using OpenEthereum as an instance, our attacks represent a universal problem that potentially lies in all Aura-based projects with the drifting tolerance mechanism. To the best of our knowledge, such affected projects include OpenEthereum, Substrate [30] and their variants. We summarize that a common reason for the unfairness issue lies in its *strong synchronous network assumption*: as the leader election in Aura relies on the calculation of machine times within a synchronous network, it is necessary to leave a *drifting tolerance* for messages to be delivered to every participant. Without such tolerance, it is unfeasible for participants to share the exact same time record in real networks. The practicality will thereby be significantly reduced.

5.3 Impacts of Time-manipulation Attacks

Economic impact. We give an extensive analysis of existing protocols and find that Aura has been applied to many mature industries such as energy transition [31], cross-border payment [32], and supply chain [33, 34]. Beyond that, in the cryptocurrency space, Data disclosed by *CoinMarketCap* [35] shows that more than 2.13 billion USD market share (cf. Table 1) is involved. Conceivably, by conducting our attack, an adversary may gain a considerable amount of economic benefits, which will ultimately be paid for by other sealers and users. This economic loss and the absence of unfairness restraint will damage the current blockchain economy and undermine the long-term trust and value of relative cryptocurrencies.

Impact on upper-layer applications. It is well known that a miner (sealer in our case) can manipulate the block timestamp to obtain advantages when attacking the targeted smart contracts [13]. For example, if a miner plays a timestamp-based betting game, he naturally has the advantage of selecting a suitable timestamp on the block he is mining. A good rule of thumb is that a contract never rely on an interval of fewer than 15 seconds. However, our *Attack-II* has proved that such experience and lesson are not reliable: by subtly designing the sleep period, we can manipulate timestamp drift up to $(15 + D_{\mathcal{A}})$ seconds or $(15 + 2 * d)$ so the proposed block can still be confirmed, expanding the unsafe time-frame. We accordingly provide an active smart contract that can be potentially affected by our attack in Appendix B.1.

Impact on consensus stability. As described in Sec.2, Aura is a hybrid consensus protocol that combines both probabilistic chain selection and deterministic finality [20]. Intuitively, our attacks only have impacts on the probabilistic sector, as the victim sealer's block cannot include in the chain, and such block loss will not clash with the best chains' voting. However, we found the block is the carrier for the voting message. The lost block may further make consensus lose the properties of *deterministic finality* [17, 18] and *accountable safety* [19, 20]. Details are provided in Appendix B.3.

6 COUNTERMEASURES

A natural approach to defend our attack is to adjust the time tolerance in the Aura protocol. In particular, for the loophole in *Attack-I*, we can simply let validators reject blocks with timestamps deferred more than η (the valid deferment). As for *Attack-II* and *Attack-III*, we can decrease *receive_threshold* (line 17, Algorithm 1), which narrows down the time tolerance for the delayed block sent by the malicious leader. With this patch, the victim sealer is always expected to generate a new block even if he did not receive the block from the last step. An alternative solution is to introduce an *audit mechanism* with punishment. By observing the block density within a time period, an auditor can learn which sealer becomes malicious. Then, he can report the abnormal behaviours and kick out the malicious sealer from the committee. Note that the timestamp loophole is an unavoidable problem in Aura because of its intrinsic time-based block-producing scheme. Under the assumption that the network is fully synchronous, and all participants have to follow the same clock time, Aura is forced to make trade-offs between safety and utility. If the range of drifting tolerance is strictly restricted, malicious behaviours might be successful in a lower probability, while correspondingly, benign behaviours (e.g., slight clock asynchrony) will also be restricted, which may reduce system practicality. This demonstrates the significance of our attacks and points out the drawback lying in Aura.

7 CONCLUSION

In this paper, we design, implement and apply a series of *time-manipulation attack*s against PoA Aura implementations, where a malicious sealer can break the leadership fairness in the block proposal stage. Our constructed strategies increase the attacker's probability of mining blocks beyond its fair share, causing an economic loss for the next honest sealer and threatening the consensus stability of the entire system. Experimental results demonstrate the effectiveness of our attacks. After that, we further dig into the root causes of such vulnerabilities and accordingly provide the countermeasures. To the best of our knowledge, this work is the first to explore the time-manipulation attack against the PoA Aura.

ACKNOWLEDGMENT

Rujia Li and Sisi Duan were supported in part by National Key R&D Program of China under grant No. 2022YFB2701700 and National Science Foundation of China under grant No. 92267203. Qi Wang was supported by Guangdong Provincial Key Laboratory (Grant No. 2020B121201001). Rujia Li was also supported by postdoctoral fellowships from the Tsinghua Shuimu Scholar.

REFERENCES

[1] Gavin Wood. PoA private chains. https://github.com/ethereum/guide/blob/master/poa.md, November 2015. accessed: Dec., 2021.

[2] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *https://ethereum.github.io/yellowpaper/paper.pdf*, 2022. accessed: Apr., 2022.

[3] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, volume 99, pages 173–186, 1999.

[4] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. 2018.

[5] Aura - authority round - wiki (OpenEthereum documentation). https://openethereum.github.io/Aura. accessed: Jul., 2022.

[6] Proof of Authority (VeChain docs). https://docs.vechain.org/thor/learn/proof-of-authority.html, 2022. accessed: Aug., 2022.

[7] Sokol sydney. https://sokolsydney.com/. accessed: Aug., 2022.

[8] Kovan testnet. https://kovan-testnet.github.io/website/. accessed: Aug., 2022.

[9] OpenEthereum. OpenEthereum. https://github.com/openethereum/openethereum. accessed: Jul., 2022.

[10] Ekparinya Parinya, Gramoli Vincent, and Jourjon Guillaume. The attack of the clones against proof-of-authority. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS)*, 2020.

[11] Qin Wang, Rujia Li, Qi Wang, Shiping Chen, and Yang Xiang. Exploring unfairness on proof of authority: Order manipulation attacks and remedies. In *Proceedings of the 17th ACM ASIA Conference on Computer and Communications Security (ASIACCS)*. ACM, 2022.

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

[13] Aviv Yaish, Saar Tochner, and Aviv Zohar. Blockchain stretching & squeezing: Manipulating time for your best interest. In *Proceedings of the 23rd ACM Conference on Economics and Computation (EC)*, pages 65–88, 2022.

[14] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Annual International Cryptology Conference (CRYPTO)*, pages 451–480. Springer, 2020.

[15] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, pages 3–14, 2022.

[16] Peter Gazi, Aggelos Kiayias, and Alexander Russell. Stake-bleeding attacks on proof-of-stake blockchains. *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 85–92, 2018.

[17] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–356, 2019.

[18] Yahya Hassanzadeh-Nazarabadi and Sanaz Taheri-Boshrooyeh. A consensus protocol with deterministic finality. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–2. IEEE, 2021.

[19] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[20] Joachim Neu, Ertem Nusret Tas, and David Tse. The availability-accountability dilemma and its resolution via accountability gadgets. *International Conference on Financial Cryptography and Data Security (FC)*, 2022.

[21] Xinrui Zhang, Qin Wang, Rujia Li, and Qi Wang. Frontrunning block attack in PoA: A case study. *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2022.

[22] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *Cryptology ePrint Archive*, 2013.

[23] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 507–527. Springer, 2015.

[24] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[25] Ethereum yellow paper: a formal specification of Ethereum, a programmable blockchain. https://ethereum.github.io/yellowpaper/paper.pdf. accessed: Jul., 2022.

[26] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[27] Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies (AFT)*, pages 25–36, 2020.

[28] Kai-Min Chung, T-H Hubert Chan, Ting Wen, and Elaine Shi. Game-theoretic fairness meets multi-party protocols: the case of leader election. In *Annual International Cryptology Conference (CRYPTO)*, pages 3–32. Springer, 2021.

[29] Block timestamp manipulation attack. https://cryptomarketpool.com/block-timestamp-manipulation-attack/. accessed: Jul., 2022.

[30] Substrate – consensus. https://docs.substrate.io/fundamentals/consensus/, 2022. accessed: Sep., 2022.

[31] Charlotta Edeland and Therese Mörk. Blockchain technology in the energy transition: An exploratory study on how electric utilities can approach blockchain technology. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1235832&dswid=-1425, 2018.

[32] Md. Mainul Islam, Md. Kamrul Islam, Md. Shahjalal, Mostafa Zaman Chowdhury, and Yeong Min Jang. A low-cost cross-border payment system based on auditable cryptocurrency with consortium blockchain: Joint digital currency. *IEEE Transactions on Services Computing (TSC)*, 2022.

[33] Neo CK Yiu. Decentralizing supply chain anti-counterfeiting and traceability systems using blockchain technology. *Future Internet*, 13(4):84, 2021.

[34] Anselm Busse, Jacob Eberhardt, Sebastian Frost, Dong-Ha Kim, Thore Weilbier, Lukas Renner, Matthias Roth, and Stefan Tai. A response to the united nations cites blockchain challenge: Incremental and integrative PoA-based permit exchange. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 320–328. IEEE, 2019.

[35] Coinmarketcap. *Accessed in Oct, 2022 https://coinmarketcap.com/*, 2022.

[36] Haahr Marianne, Foster Katherine, et al. Blockchain: Gateway for sustainability linked bonds. *Accessed in Oct, 2022 https://greendigitalfinancealliance.org/wp-content/uploads/2019/12/blockchain-gateway-for-sustainability.pdf*, 2022.

[37] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 281–310. Springer, 2015.

[38] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain cap theorem allows user-dependent adaptivity and finality. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 84–103. Springer, 2021.

[39] Yunhan Hu, Guohua Tian, Anqi Jiang, Shuqin Liu, Jianghong Wei, Jianfeng Wang, and Shichong Tan. A practical heartbeat-based defense scheme against cloning attacks in PoA blockchain. *Computer Standards & Interfaces (CSI)*, 83:103656, 2023.

[40] Kentaroh Toyoda, Koji Machi, Yutaka Ohtake, and Allan N Zhang. Function-level bottleneck analysis of private proof-of-authority Ethereum blockchain. *IEEE Access*, 8:141611–141621, 2020.

[41] Xuefeng Liu, Gansen Zhao, Xinming Wang, Yixing Lin, Ziheng Zhou, Hua Tang, and Bingchuan Chen. MDP-based quantitative analysis framework for proof of authority. In *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pages 227–236. IEEE, 2019.

[42] MIT Digital Currency Initiative. 51% attacks. https://dci.mit.edu/51-attacks. accessed: Apr., 2022.

[43] Yackolley Amoussou-Guenou, Bruno Biais, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Rational behaviors in committee-based blockchains. In *24th International Conference on Principles of Distributed Systems (PODC)*, page 1, 2021.

APPENDIX

A POA AURA PROJECTS

We estimate potential monetary loss of PoA projects. We list the projects using Aura and their market values in Table 1, including both cryptocurrency projects and academic projects.

B ANALYSIS OF IMPACTS

Following the discussions in Sec.5, we provide more details showing the impacts of the attacks.

B.1 Smart Contracts

We present an instance that is based on Ethereum smart contracts, where the timestamp plays an essential role in contract functionalities. In the following *lottery* contract, timestamps are set to be a critical condition for gaining rewards. Players pay 1 ether for admission qualifications, and the contract randomly selects the winner by checking if the timestamp of the block is divisible by 16 (line 14). Following the Ethereum design, malicious players can easily manipulate their timestamps within 15 seconds to meet the winning condition without the risk of rejection. Therefore, by modifying the modulus to 16, an adversary's misbehaviour cannot be

Table 1: Selected projects and their market cap

Sector	*Project*	*Market Cap*	*Notes*
Cryptocurrency projects	Parity	-	https://parity.io
	OpenEthereum	-	https://openethereum.org
	VeChainThor	$1,716,313,385	
	POA Network	$5,257,900	
	Gnosis Chain*	$393,061,738	pre-merged version
	Kovan	-	Ethereum testnet
	Sokol	-	Ethereum testnet
	Nethermind	-	https://nethermind.io
	Arianee Mainnet	$ 6,960,361	https://www.arianee.org/
	Rio Chain	$1,999,820	https://www.riochain.io/
	Shiden	$9,676,668	https://shiden.astar.network
	Ingotsex	-	
	Bloxberg	-	https://bloxberg.org
	Substrate	-	https://substrate.io
	Chronicled	-	https://chronicled.com
Academic projects	Permit Exchange	-	[34]
	Energy Transition	-	[31]
	Cross-Border Payment	-	[32]
	World Bank Blockchain	-	[36]
	Supply Chain	-	[33, 34]

[1] Data accessed in September 2022, from *CoinMarketCap*.

successful with 100% probability. However, our attack expands the unsafe time frame of current construction, increasing the success probability. Such a contract becomes insecure since a malicious miner has the ability to manipulate block timestamps up to 20 seconds (by conducting *Attack-III*).

Listing 1: A smart contract for lottery

```
Contract Roulette {
    uint public pastBlockTime;
    //initially contract
    constructor() {}
    //receive function
    receive() external payable {}
    //fallback function used to make a bet
    fallback() external payable {
        //must send 1 ether to play
        require(msg.value == 1 ether);
        //only 1 transaction per block
        require(block.timestamp != pastBlockTime);
        pastBlockTime = block.timestamp;
        if(block.timestamp % 16 == 0) { //winner
            payable(msg.sender).transfer(address(this).balance);
        }
    }
}
```

B.2 Impacts on System Performance

We discuss the performance trends affected by our attacks. Instead of damaging the entire network, only one of the sealers' interests will be affected (the incoming leader) by the malicious leader in the current turn. Figure 5 shows the comparison of transaction confirmation rates between normal cases, and multiple attacking cases under the assumption that no sealer quits the network during continuous attacks. As the number of transactions increases, the confirmed transactions grow steadily. The impact on the total confirmed transactions rate is negative, but not prominent. However, in the real world, the victim sealer may quit mining when being aware of the unfair distribution of mining rewards. Thus, the full-loaded confirmation rate may decrease as the network scale shrinks.

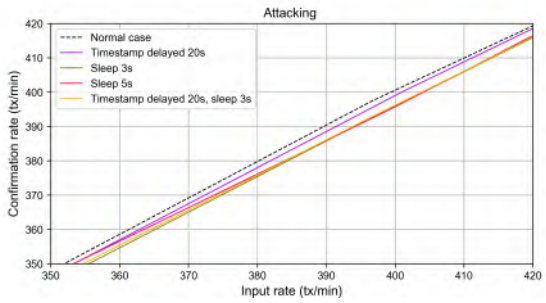


Figure 5: System transaction confirmation rates under different attack strategies, when transaction input rate grows.

B.3 Impact on System Stability

Deterministic finality [17, 18] means that a transaction becomes impossible to revert and is considered to be final once it is added to the public ledger. Deterministic finality in Aura aims to provide provable finality for the newest longest common prefix [37].

Theorem B.1. *Suppose that f Byzantine sealers launch Attack-II/III simultaneously. The protocol cannot reach deterministic finality.*

Proof. We show that for any valid block $\mathcal{B}_i$ with the corresponding longest common prefix, it never reaches a deterministic finality. Suppose $\mathcal{B}_i$ comes to a deterministic finality. In that case, it requires over a half sealer's confirmation on $\mathcal{B}_i$ or on the block treated $\mathcal{B}_i$ as an ancestor. Such confirmation messages are achieved by collecting signatures on the blocks attached to $\mathcal{B}_i$. In extreme circumstances, f Byzantine sealers simultaneously launch our attacks on the chain, and each of them prevents the block proposed by the next candidate from being attached to $\mathcal{B}_i$. This will, in the worst case, result in a total of $2f$ sealers of creating invalid signatures, which means $\mathcal{B}_i$ can only collect 1 confirmation messages, less than $\frac{2f+1}{2}$. Thus, the protocol cannot reach a deterministic finality. □

Accountable safety, proposed in Casper [19], focuses on fault detection and accountability for consensus algorithms when there is a violation of safety property. In Aura, it aims to find the sealers who violate the consensus rules in case of two conflicting votes [20, 38].

Corollary B.2. *Suppose that f Byzantine sealers launch Attack-II/III simultaneously. The protocol cannot reach accountable safety.*

Proof. From Theorem B.1, we have known that arbitrarily latest block $\mathcal{B}_i$ cannot achieve consensus among sealers. Thus, the committee members who depend on the block consensus cannot be determined, and thereby Byzantine sealers will not be detected. □

C RESOURCES

This section presents source code, developed tools, experimental results and mechanism comparisons.

C.1 Codes and Tools

We have made our source code publicly available, including Rust source code and Python auxiliary tools. Both raw and processed experimental data can be found in the following repository links.

- **Attacking Point I** https://github.com/openethereum/openethereum/blob/main/crates/ethcore/src/engines/mod.rs#L504
- **Attacking Point II** https://github.com/openethereum/openethereum/blob/main/crates/ethcore/src/block.rs#L196
- **Codes on *Attack-I*** https://github.com/TEEs-projects/Time-manipulation-Attack/tree/main/openethereum-3.3.4_25s
- **Codes on *Attack-II*** https://github.com/TEEs-projects/Time-manipulation-Attack/blob/main/openethereum-3.3.4_sleep3s
- **Codes on *Attack-III*** https://github.com/TEEs-projects/Time-manipulation-Attack/blob/main/openethereum-3.3.4_23s_sleep3s
- **Analysis Tool (Python)** https://github.com/TEEs-projects/Time-manipulation-Attack/tree/main/testchain/shellgen
- **Experimental results** https://github.com/TEEs-projects/Time-manipulation-Attack/tree/main/results

C.2 Pseudocodes of the Attacks

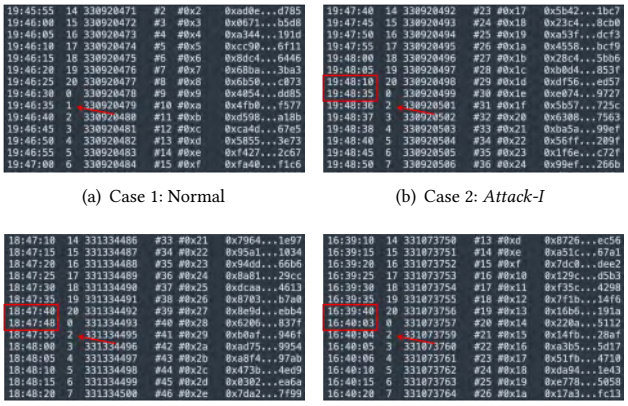
We provide the detailed code segments of each attack in Algorithm 2. Our attacks target on the *block proposal* stage without any modification on *block verification*. We mainly attack the `timestamp` (TS) function by instantiating three strategies: falsify timestamps (line 4-5), delay blocks (line 6-8), and combine them (line 9-11). Notably, for *Attack-III*, we choose 23 seconds as an instance that falls in the range of $[\text{timestamp}_{\mathcal{P}} + 1\text{sec}, \text{timestamp}_{\mathcal{O}} + 15\text{secs} + \text{D}_{\mathcal{A}}]$ (Equation 2).

Algorithm 2 Attacks on Aura (OpenEthereum)

```
1:  function TS(lastblock.timestamp)
2:      if Normal then                                  ▷ Normal case
3:          return (system.time)
4:      else if Attack-I then                           ▷ Attack-I
5:          return (lastblock.timestamp + 25)
6:      else if Attack-II then
7:          sleep(3s)                                   ▷ Attack-II
8:          return (max(now, lastblock.timestamp + 1))
9:      else if Attack-III then                         ▷ Attack-III
10:         sleep(3s)
11:         return (lastblock.timestamp + 23)
12: end function
```

C.3 The Screenshots of Experiments

Figure 6 shows a series of block information that is generated by separately running the normal *Aura* clients and malicious clients. From left to right, the first column shows the block timestamps, while the second column shows the sealers' indexes. The following columns show block numbers (decimal and hexadecimal) and the block hashes. In the normal case, a sealer should rotate in a numerical order. As shown in Figure 6.a, the $sealer_0$'s block is followed by the $sealer_1$'s block. In our three attacks, $sealer_1$'s blocks are all lost, indicating the effectiveness.



(a) Case 1: Normal (b) Case 2: *Attack-I*

(c) Case 3: *Attack-II* (d) Case 4: *Attack-III*

Figure 6: Screenshots on running *Aura* clients

C.4 Symbol Definitions

Table 2 lists the symbols used in our paper.

Table 2: Symbol definitions

.		*Definition*			*Definition*	
Parameter	*diff*	difficulty / target	$\text{timestamp}_{\mathcal{P}}$	parent timestamp		Tag
	d	step duration	$\text{timestamp}_{\mathcal{O}}$	the original timestamp		
	s	the step of a block	$\text{timestamp}_{\mathcal{M}}$	the falsified timestamp		
	δ	network upper-bound delay	$\mathcal{F}(\cdot)$	used to evaluate common prefix		Function
	n	number of sealers in committee	$f(\cdot)$	rotation function to find a leader		
	f	number of sealers in committee	TS(·)	function to set time duration		
	$\text{D}_{\mathcal{A}}$	sleeping period of adversaries	$\mathcal{S}_i$	i-th sealer in the committee		Entity
	Δ	the cooling period, $\Delta = \text{T}_2 - \text{T}_1$	$\mathcal{L}_i$	i-th leader in the committee		
	T_0	block received time	$\#b_i$	blocks produced by the sealer		
	T_1	voting start time	$b_{\mathcal{G}}$	the genesis block		
	T_2	new block generation time	$\mathcal{L}-$	leader set		
	η	a valid deferment	$\mathcal{B}-$	block set		
	ϵ	short drifting period	$\mathcal{T}-$	transaction set		

D RELATED ATTACKS

A series of PoA attacks threaten the security promises of PoA protocols. Vincent et al. [10] proposed the *cloning attack* by creating a fork and maintaining two branches to break PoA protocols' safety. Subsequently, Hu et al. [39] provided a heartbeat-based defense scheme to mitigate this attack. For *fairness* of PoA Clique, Wang et al. [11] proposed two types of frontrunning attacks that break both the transaction-level and block-level fairness in Clique implementations. They introduced a notion of "edge-turn" sealer to describe the block proposer with lower priority per turn, and

modified the block delay configurations when legal edge-turn sealers are creating blocks. Meanwhile, Zhang et al. [21] attacked the PoA *Clique* implementation and caused the block disorder by targeting in-turn sealers. Besides, several common attacks, such as 51% attack, also break the *liveness* and *safety* property. We provide a detailed comparison with our attack in Table 3. In addition, a line of researcher work has drawn attention to delivering security analyses from the views of the CAP theory [4], functional methods [40] and programming logic [41]. However, none of the above work presents an attack either against *fairness* of Aura or achieved by time manipulation.

Table 3: Comparison on security properties

Attack	*Threshold*	*Rationality*	*Safety*	*Liveness*	*Fairness*	$\mathcal{L}$*-level*	$\mathcal{B}$*-level*	$\mathcal{T}$*-level*	*Target*
PoA attack Type-I [11]	$> \frac{n}{2}$	✓	✓	✓	✗	✓	✓	✗	edge-turn
PoA attack Type-II [11]	$> \frac{n}{2}$	✓	✓	✓	✗	✗	✗	✗	edge-turn
Clique attack [21]	$> \frac{n}{2}$	✓	✓	✓	✗	✗	✗	✗	in-turn
Clone attack [10]	$> \frac{n}{2}$	-	✗	✗	N/A	N/A	N/A	✗	in-turn
51% attack [42]	51%	-	✗	✗	N/A	N/A	N/A	✗	in-turn
This work	$> \frac{n}{2}$	✓	✗	✓	✗	✗	✗	✗	in-turn

[1] *Rational*: Seeking for maximal profits within the scope of code principles [43].
[2] ✓Holding such a property (attack resistance), while ✗represents the opposite.
[3] An edge-turn sealer proposes blocks when the in-turn sealer shuts down [11].

Comparisons to competitive attacks. We highlight the comparisons between our attacks and existing PoA attacks as in Table 3. We analyze their critical assumptions and their respective outcomes in terms of consensus properties. Similar to other PoA attacks, the majority of validators are assumed to be honest. Our attacks have the same assumption with [11, 21], where the attacker is rational and greedy to obtain more rewards than his fair share [43]. In contrast, our attacks directly break the fairness property and indirectly break the safety property, which is more harmful than other attacks.