Exploring Unfairness on Proof of Authority: Order Manipulation Attacks and Remedies

Qin Wang∗
Swinburne University of Technology & CSIRO, Data61
Melbourne, Australia
qinwangtech@gmail.com

Rujia Li∗
Southern University of Science and Technology & University of Birmingham
Shenzhen, China
rxl635@student.bham.ac.uk

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

Shiping Chen
CSIRO, Data61
Sydney, Australia
shiping.chen@data61.csiro.au

Yang Xiang
Swinburne University of Technology
Melbourne, Australia
yxiang@swin.edu.au

ABSTRACT

Proof of Authority (PoA) is a type of permissioned consensus algorithm with a fixed committee. PoA has been widely adopted by communities and industries due to its better performance and faster finality. In this paper, we explore the unfairness issue existing in the current PoA implementations. We have investigated 2,500+ in the wild projects and selected 10+ as our main focus (covering Ethereum, Binance smart chain, etc.). We have identified two types of order manipulation attacks to separately break the transaction-level (a.k.a. transaction ordering) and the block-level (sealer position ordering) fairness. Both of them merely rely on honest-but-profitable sealer assumption without modifying original settings. We launch these attacks on the forked branches under an isolated environment and carefully evaluate the attacking scope towards different implementations. To date (as of Nov 2021), the potentially affected PoA market cap can reach up to 681, 087 million USD. Besides, we further dive into the source code of selected projects, and accordingly, propose our recommendation for the fix. To the best of knowledge, this work provides the first exploration of the unfairness issue in PoA algorithms.

CCS CONCEPTS

• Security and privacy → Distributed systems security;

KEYWORDS

Proof of Authority, Fairness, Order Manipulation

∗These authors contributed equally to the work.
† 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Qin Wang∗, Rujia Li∗, Qi Wang, Shiping Chen, and Yang Xiang. 2022. Exploring Unfairness on Proof of Authority: Order Manipulation Attacks and Remedies. In Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS ’22), May 30–June 3, 2022, Nagasaki, Japan. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3488932.3517394

1 INTRODUCTION

Proof of Authority (PoA) [1, 2] is a type of permissioned consensus algorithm that provides a practical and efficient solution for current blockchain systems, especially for consortium blockchains. The algorithm was firstly proposed by Wood and deployed by Ethereum [3] in 2017. PoA is based on a limited number of authority nodes, called sealers, to perform consensus procedures. Blocks and transactions have to be verified by these eligible nodes. The design of a small size committee enables fast confirmation of transactions and easy management of involved members. Many blockchain projects thereby adopt PoA as their consensus algorithms. The most famous application is Ethereum, implemented in the form of two different types [2]: Aura (short for Authority Round) in Parity [4], and Clique in Geth [5]. Aura modifies the traditional Byzantine Fault Tolerance (BFT)-style protocols by simplifying them into a two-phase protocol (which correspondingly requires stronger assumptions). Aura is further adopted by Kovan Testnet [6], Laava [7], VeChainThor [8], Microsoft Azure (deployment) [9], and xDai DPOS network [10]. Clique [11] extends the permissioned settings into permissionless, by relying on the proof-of-work (PoW) consensus. It adds the procedure of authentications and brings the role of leaders. This version has been (or attempted to be) applied to Binance smart chain [12], Amazon blockchain network [13], POA network [14], Rinkeby [15], Quorum [16], Görli network [17], Apla blockchain [18] and HPB [19]. However, despite its wide adoption, PoA and its variants have not seriously proved to be secure. This motivates our investigation, starting with the following observations:

We firstly dive into the mechanisms of PoA protocols. Both Aura and Clique follow the basic idea of BFT-style consensus (cf. Fig.1), where only one authority leader determines the block with instant finality. The differences lie in two aspects: (i) the ways to elect this leader; and (ii) the methods to agree on the block with peers.
In *Aura*, the sealer rotation depends on the formula of \( l = \frac{T}{\text{duration}} \mod N \), where \( l \) represents the index of the leader, \( T \) is the current UNIX time, \( \text{duration} \) is the time interval between blocks and \( N \) is the size of the committee (equal to the number of sealers). Once the leader is elected, the protocol launches a two-phase consensus: *block proposal* and *block acceptance*. The leader proposes a proposal (blocks containing a set of transactions) and broadcasts them to peers. Then, peer nodes (sealers) vote for this proposal. Only when the received responses exceed half of the involved sealers (51%), this block is formally deemed to be confirmed. As in *Clique*, the leader is rotated under the formula of \( g = h \mod N \) where \( h \) represents the block height. Such a way is inherently a *round-robin* rotation, as \( h \) increases linearly along with the chain growth. Compared to *Aura*, the main difference is that multiple sealers are allowed to propose blocks but restricted to propose a block every \( N/2 + 1 \) block. Here, the leader has a priority of proposing the block while non-leader sealers (but can still propose blocks) have to lag their blocks with a random delay. The *difficulty* field of these non-leaders is set as 1 whereas the leader’s as 2. To break ties\(^1\), the chain with the highest scores will be selected as the valid chain (GHOST rule [20, 21]). Since a leader will be granted a higher score than other sealers, the block generated by this leader will be included in the final chain with a high probability.

Then, we investigate their properties and assumptions. According to its BFT-style design, PoA prioritizes consistency over *liveness* [2] (equiv. *safety* and *termination* in the context of BFT). Safety means all the honest sealers should have the same view of blocks at a specified height, while liveness emphasizes the honest blocks will eventually get included in the (longest) chain. To achieve safety, PoA relies on committee members (a.k.a. sealers) to offer deterministic finality. The proposed block, once agreed by the majority of sealers, will be instantly formatted as an unchangeable block proposal with a high probability. This provides faster state without the possibility of being reversed. This provides faster confirmation and greater throughput. Liveness in PoA is achieved via both ongoing *sealer rotation* and the GHOST rule (the highest-score/weightiest subchain wins). Sealer rotation makes the system still proceed in the case of single-point failures, and GHOST ensures that the chain will always grow under frequent forks. However, readers may easily omit an implicit assumption guaranteeing such properties: *a majority of sealers (51%) have to honestly behave in the synchronous network*. This assumption is very strict among existing distributed systems, which results in the lack of practicability in real-world applications. Despite this, we still find vulnerabilities under its strong assumption.

We observe that PoA consensus protocols cannot guarantee *fairness*. Fairness focuses on the fair orderings, covering both block-level and transaction-level. The block-level means the block should be proposed in a proper turn. This requires their producers (a.k.a. sealers) to faithfully take turns to mine blocks, without breaking the prearranged order. Transaction-level fairness refers to their fair orderings [22, 23] where transactions that have not yet been confirmed (pending status) confront the risk of being reordered by sealers for extra revenues (a.k.a. miner extracted value).

---

\(^1\)In blockchain systems, *tie* means a temporary status where two competitive branches (subchains) have the same length (or equiv. scores/weights/graphs) that results in difficulties in subchain selection by previous blocks.

MEV [24]). These sealers may conduct arbitrary manipulation of the actual ordering of transactions by frontrunning, backrunning, or sandwitching [25] transactions inside their proposed blocks. The threat of unfairness in PoA comes from its ambiguous statement of so-called *honest behaviors*. The protocol merely assumes sealers should behave honestly, but in fact, they can act as well as *profitably*. The task to order transactions in classical blockchain protocols is like *dark forest* [26] for miners. Similarly, PoA protocols follow such a design, in which transactions in the mempool [24] are intransparently ordered. This does great harm to users if the block producer intentionally manipulates their received transactions, causing a potentially huge amount of monetary loss for users who have honestly behaved in every step.

In this paper, we explore the *unfairness* issue in PoA protocols by initiating adversarial order manipulation attacks. Unfairness is caused from two folds, and we accordingly initiate two types of profitable order manipulation strategies. Firstly, like miners in Ethereum [3], authority sealers in PoA-engineed systems can arbitrarily sort transactions when sealing blocks, leaving an undiscovered area for adversarial manipulations by profit-pursuing sealers. This is a transaction-level frontrunning attack (denoted as the *Type-I* attack) that is effective for all PoA implementations, as every honest block producer can insert her profitable transaction to outpace others or delay peer transactions that are disadvantageous. Secondly, different from most permissionless consensus protocols (e.g., PoW) whose miners are randomly chosen from dynamic participants, PoA relies on a simplified BFT-style consensus with a static committee. The sealer rotation procedure depends on an open-sourced formula to select incoming block producers (more than one producer in *Clique*). This brings threats because a profit-pursuing sealer can precisely predicate her turn to create blocks, and then forfeit the priority parameters (e.g., the field of *delay time*) to occupy an advantageous position. Thus, we launch an attack (Type-II) to frontrun the turn of privilege positions and further hold this position via broadcasting faked parameters (the very short *delay time*). We implement these attacking strategies by slightly modifying the source code of Geth [27] to simulate a profitable sealer. We apply both attacks in an isolated private PoA network (due to ethical reasons) and evaluate results in terms of effectiveness and success rates. Meanwhile, we submit the issue report to several projects and get their approval (cf. PR report [28]).
Furthermore, we evaluate the potential loss of PoA markets affected by our attacks and provide in-depth discussions to mitigate the unfairness with our recommended countermeasures. We review existing PoA-engined projects from Github and select 10+ typical projects as our research focuses (see Table 2). The selected projects can cover most mainstream PoA implementations with their variations, where many projects directly apply PoA without any modifications (e.g., Ethereum Classic [29], Olecoin [30]). We dive into these exemplified projects to explore the impact scope of our attacks. The Type-I attack performs effectively against all types of PoA implementations, directly or indirectly influence up to 681,087 million USD market cap [CoinMarketCap, Nov 2021]. In contrast, the Type-II attack is merely effective on part of Clique-based projects. The reason lies in their different implementations of verification function. The function that checks both identity and difficulty for every sealer who is sealing blocks can resist to our Type-II attack, whereas the function checking merely difficulty cannot prevent it, especially for non-leader block-producing sealers (difficulty set as 1). This is because difficulty is a fungible field in the checklist, requiring no further consensus procedures, while the identity field has to be confirmed by all peer nodes. Our attack focuses on those non-leader sealers, manually shortening their delay time (set to zero) for a priority of becoming dominant when the leader shuts down. Based on that, we provide our recommendations to mitigate the unfairness problem in PoA, avoiding the Type-II attack as well as decreasing the potential loss. We provide three types of solutions. The first is to modify its core verification function, equipping identity checks. This type of solution applies direct modifications to source codes without additional complexity, but with sacrifices on system availability. The second approach introduces the physical randomness sampled from hardware. The third aims to make the turn of rotation unpredictable. This method requires on-top protocols or strong guarantees. A comprehensive discussion is attached in terms of their core benefits and drawbacks.

In a nutshell, we summarize our contributions as follows.

- We explore unfairness in PoA protocols. We review 2,500+ in the wild projects and select 10+ typical systems as our analysed targets. We dive into their source code and find the root cause that results in the unfairness issue.
- We initiate two types order manipulation attacks against both transaction-level and block-level fairness of PoA protocols. We simulate our attack under the testnet of real-world projects (e.g., Ethereum Geth [27]). Experiment results show that our attacking strategies can cause effective damages.
- We conduct a rigorous analysis of our attacks and evaluate potential loss in real-world markets. We compare the differences among the selected projects and discuss their potential attacking impacts in terms of on-site markets. Results indicate that the potentially affected market cap of these PoA projects can reach up to 681,087 million USD.
- We give our recommended remedies against mentioned vulnerabilities. Our recommendations include various methods, covering code modification, hardware-assisted solutions, and crypto-based enhancement. We have further disclosed our results to some ongoing teams, referring to [28].

Paper Structure. A high-level protocol design is presented in Section 2. The attacking strategies are introduced in Section 3, and are detailed in Section 4 and Section 5, respectively. A comprehensive discussion is presented in Section 6. Our recommended fixes are provided in Section 7. Related studies are discussed in Section 8. Finally, Section 9 concludes our work. Appendix A-D provide supplementary details omitted in the mainbody.

2 POA ALGORITHMS

This section presents two mainstream PoA implementations, Aura and Clique, by separately introducing their commons and operating mechanisms.

2.1 PoA Model

We abstract the PoA model from multiple related implementations. Despite their ways of completing consensus being different, we still find their commons in terms of participated roles, network assumptions, and adversary threat models.

Roles. PoA algorithms rely on a small group of authorities to make up the committee to perform consensus. Committee members are known as sealers, who can package transactions and produce blocks. In each round, one of them will be selected as the leader to propose the block and gather responses from others. To distinguish their roles, a sealer who becomes the leader is denoted as in-turn sealer (difficulty as 2), while others are out-of-turn sealers as claimed in [31]. However, we clarify out-of-turn sealers with more precise definition by adding a new term called the edge-turn sealer, referring to a non-leader block-producing sealer (difficulty as 1). The sealer who cannot produce any blocks are called out-of-turn (difficulty as 0). The in-turn sealer enjoys a priority of sealing blocks within her turns, whereas edge-turn sealers create blocks with a lagging time.

Assumptions. PoA is designed for permissioned blockchain systems, where a limited size of sealers can seal blocks during the consensus procedures. We only put focus on communications among these committee nodes. The network is assumed to be partially synchronized, in which all the messages can be eventually delivered to their destinations with a maximum upper-bound delay of δ. This means if one sealer sends a transaction to another sealer at the time τ, the transaction will be delivered no later than the time τ + δ. We consider partial synchrony in our model due to its wide applicability towards most in the wild projects. Few PoA implementations [4] emphasize their total synchrony to guarantee their consistency and safety, but sacrifice the general adoption by real applications.

Threat Model. Previous attacks are generally based on negative assumptions such as the Byzantine failure in which a node acts arbitrarily: drop, delay, duplicate, or even fake messages. In contrast, our attack is based on a rational model where all the committee members behave honestly but can still pursue profits under predefined rules. We narrow down the scope of the utility function of these profits to simple monetary rewards, rather than any other type of virtual assets that are external to the system, like reputation mentioned in [31]. Therefore, in our attack, we assume two related
profitable models. (i) an in-turn sealer (leader) can arbitrarily reorder the received transactions within her proposed block. This strategy does not break any honesty rules of the system (equiv. no modifications of any code). (ii) an edge-turn sealer (diff $= 1$) can continuously send falsified parameters (e.g., set lagging time as 0) to frontrun the advantageous position when the leader shuts down. This is a weaker promise than the former one, but still satisfies the legal rule because the sealer sends a valid number within the scope of protocol definitions. Here, legal means there are no checks towards the identity of edge-turn sealers and delay time in verification. Besides, the basic claim of 51%-honesty assumption for general PoA systems is kept as original in our attack.

2.2 PoA Implementations

We provide two mainstream PoA implementations. In particular, we emphasize mechanisms from a block’s proposal to verification.

**Aura Algorithm.** Aura (Authority Round) is implemented in Parity. The network in Aura is assumed to be synchronous, and all authorities are assumed to hold the same UNIX time $T$ (cf. Alg.1 line 9). The leader of each step is deterministically calculated by duration (line 10) and the number of authorities $\text{sealers}$ (line 11), which is represented as $i = T \div \text{duration} \mod \text{sealers}$. Each authority executes an infinite loop that periodically checks whether $i$ equals her position index. If it does, this authority will propose a block, and broadcast it to other authorities. If the received block is not produced by the current leader with correct difficulty $\text{diff}$, it will be rejected. This is achieved by the verification algorithm (line 19-24). Otherwise, if the block is valid, the authorities will send received blocks to others. The leader is always expected to produce a block with correct difficulty, even if no transactions are available.

**Clique Algorithm.** *Clique* is a PoA algorithm that was first implemented by Geth [27]. Similar to the Aura algorithm, in each round, a new block is proposed by the current leader. However, a leader is selected by combining the block number and the number of authorities rather than the UNIX time (cf. Alg.2 line 11). The main difference between *Clique* and other algorithms is that the selected leader is not exclusive in each block period, while other authorities are still allowed to propose blocks with a random-delayed time (Alg.2 line 16). The parallel-block-producing mechanism effectively prevents a single Byzantine authority from havoc on the whole network. Forks may frequently occur where a block is proposed by more than one authority. *Clique* solves this issue by two approaches. Firstly, a random delay time with wiggle must be added to a new block that is created by a non-leader block-producing sealer (Alg.2 line 15). Thus, the block produced by a leader has advantageous possibilities of being accepted by peer sealers. Secondly, an accumulation scoring mechanism and the GHOST protocol (or longest chain rule, depending on specific implementations) have been applied to resolve forks. In particular, a leader’s block will be granted higher scores than peers. Based on that, GHOST [20] guarantees that such a block will be eventually accepted by a majority of authorities. Here, we emphasize that in each round, the number of authorities, including the leader and authorities who can propose delayed blocks (in/edge-turn), is limited to $\text{sealers} = \left\lfloor \frac{T + \text{duration}}{\text{duration}} \right\rfloor + 1$ (Alg.2 line 37-42, with associated source code of Geth at [27] line 477-483). The design provides high availability and simultaneously reduces network redundancy for the system, guaranteeing the system can still operate when the in-turn sealer and followed edge-turn sealers (not all) shut down.

3 THE ORDERING ATTACK

This section introduces our attacks, covering both Type-I (frontrunning attack in transaction-level) and Type-II (break the proper turn of enrolled sealers) attacks. We present a bird view of our attacks. Instances towards *Aura* and *Clique* are discussed in Section 4 and Section 5, respectively.

3.1 Breaking Fairness of Transaction Ordering

Transactions once sent by users will firstly arrive in a pending pool [24], waiting for the miners’ pickup. In most cases, miners conduct transactions according to a set of priorities such as paid fees (e.g., Ethereum [3]), timestamp, randomness or combinations of them (as discussed in [32]). A user can facilitate her transaction by increasing the gas fee if she has lagged out of peers. It is an eligible way for regular users because the rules are known and agreed upon by them in advance. However, that fact is that miners have advantageous opportunities to earn more revenues by inserting/reordering/delaying transactions in addition to normal block rewards, as they can perform arbitrage strategies without being monitored. Based on that, fairness, in terms of transaction ordering, requires that miners (sealers) to execute received transactions in a proper turn according to their predefined rules. Alternatively, the

| Algorithm 1 Block Propose and Block Verify in Aura |
| --- |
| 1: procedure QUEUE(tx) \[ queue a transaction \] |
| 2: \[ if(nonce_check(tx) \land gas_check(tx)) then \] |
| 3: mempool <- add(tx) \[ \rightarrow add tx to mempool \] |
| 4: broadcast(tx) \[ \rightarrow send the block \] |
| 5: end procedure |
| 6: |
| 7: procedure PROPOSE(sealer) \[ propose a block \] |
| 8: while (true) do |
| 9: $T = clock \div \text{time()}$ |
| 10: step <- $T \div \text{duration}$ |
| 11: if($i \in \text{sealers} \land \text{step} \mod \text{sealers} = i$) then |
| 12: $\text{TXs} \leftarrow \text{mempool.pop()}$ \[ \rightarrow \text{attack point-I} \] |
| 13: block <- sign($\text{TXs}$) \[ \rightarrow \text{in-order sealing} \] |
| 14: broadcast(block) \[ \rightarrow \text{send the block} \] |
| 15: sleep($\text{step} \div \text{duration}$) \[ \rightarrow \text{attack point-II} \] |
| 16: end while |
| 17: end procedure |
| 18: |
| 19: procedure VERIFY(sealer) \[ verify a block \] |
| 20: $\text{diff} = \text{calculate_difficulty}($\text{header.number}, \text{sealer}$) |
| 21: if ($\text{diff} == \text{header.difficulty}$) then \[ \rightarrow \text{block is valid} \] |
| 22: return true |
| 23: else |
| 24: return false \[ \rightarrow \text{block is invalid} \] |
| 25: end procedure |
Order of transactions is sorted merely by their intrinsic properties rather than external manipulation.

The Type-I attack aims to enforce the manipulation of transaction ordering (see Fig.2.a). This attack typically needs to find a profitable transaction as the target. We assume the victim transaction as $Tx_v$, and the frontrunning transaction as $Tx_f$. As discussed in [33], the manipulation covers three types of activities, namely displacement, insertion and suppression. Displacement means the sealer’s transaction $Tx_s$ is sorted before the victim’s $Tx_v$. Insertion includes two sealers’ transactions $Tx_{s1}$ and $Tx_{s2}$, where $Tx_{s1}$ is ordered before $Tx_v$ and $Tx_{s2}$ is after $Tx_v$ (also known as the sandwich attack). Suppression indicates that the attacker sends a set of $Tx_s$ to fill up the block by reaching the maximum of gas limits. In this way, $Tx_v$ is suppressed and cannot be included in the next block. We apply the first strategy and second strategy to our PoA model, and skip the third strategy, as this type of attack is generally launched by extra assumptions. Exceeding the gas limit will cost a lot of fees, greatly reducing the miner’s revenue. We provide our detailed attacking procedures in Section 4.

### 3.2 Breaking Fairness of Election Turn

Compared with transaction-level ordering, another important aspect of fairness is to keep the proper order of sealers’ positions, especially in BFT-style consensus that relies on powerful authorities. The PoA algorithm, as a variant of BFT protocols, follows similar principles. The identities of sealers in PoA are visible to anyone, resulting in the risk of being manipulated by internal (profitable) parties. An edge-turn sealer may spare no effort to pursue profits by ways of frontrunning the position to produce blocks when the leader shuts down. Even worse, some PoA implementations (e.g., Clique) provide probabilistic finality where the chain grows with inevitable forks. The faster a chain grows, the more forks it produces. Any unpredictable events such as delayed blocks or malicious occupation may lead to system failure. Therefore, making sealers sit in correct positions during the rotation can reduce negative impacts.

Our Type-II attack targets to break the eligible order of sealer rotation. The attack is to seize a leading position among all edge-turn sealers. Assume that totally nine sealers participate in the network, one of them is an in-turn sealer $N_0$, two are edge-turn sealers $N_5$, $N_4$ and others are out-of-turn sealers $N_2$, $N_3$ with the turn rotation of $N_0 \rightarrow N_1 \rightarrow \ldots \rightarrow N_9$ (cf. Algm.1 line 10-11 and Algm.2 line 1). Here, we suppose that the attack starts at the round $i$ and $N_4$ in this round is a profitable edge-turn sealer who attempts to frontrun the turn of $N_5$ (see Fig.2.b). Similar attacks of profitable sealers $N_5$ in round $i+1$ and $N_2$ in round $i+2$, are also applied against benign sealers $N_4$ and $N_3$, respectively.

Both Aura and Clique create an eligible difficulty of either 2 for in-turn sealer or 1 for edge-turn sealers in Block Propose. The block with 2 will outpace peers due to its highest priority of the weight (Algm.2 line 12). Our attack performs ineffective in this situation because an in-turn sealer must be set with 2 (a strict verification step, Algm.2 line 37-42). Therefore, we put our focus on edge-turn sealers who can also generate blocks when the leader fails for some reason (e.g., being compromised). The key of our attack is to frontrun the position of others sitting in front. Since all edge-turn sealers equivalently hold the difficulty of 1, peer nodes will pass the first-received block without having to distinguish who has sent them. Our attack exactly leverages the unexplored area in the ambiguous verification towards edge-turn sealers: peer nodes merely check whether the difficulty field is legal but omit the check on their origins. Based on that, our modifications of the source code are constrained merely to the delay time of edge-turn sealers (Algm.2 line 17), without breaking any code rules. Our modifications pose negligible impacts to peer nodes. However, due to significant differences between Aura and Clique in their Block Verify, our

**Algorithm 2 Block Propose and Block Verify in Clique**

1. procedure Queue($tx$) $\triangleright$ queue a transaction
2. if (nonce_check($tx$) ∧ gas_check($tx$)) then
3. mempool ← add($tx$) $\triangleright$ add tx to mempool
4. broadcast($tx$) $\triangleright$ send the block
5. end procedure
6.
7. procedure Propose($sealer_i$) $\triangleright$ propose a block
8. while (true) do
9. $n \leftarrow$ lastblock.number
10. wait until ¬sign_recently($sealer_i$, $n$) $\triangleright$ wait to seal
11. if ($n+1$) mod |sealers| = $i$ then
12. weight = 2 $\triangleright$ in-turn sealer
13. else
14. delay = block.time − now
15. wiggle = (|sealers| + 1) * 500ms $\triangleright$ wiggle time
16. sleep(delay) $\triangleright$ sleep for a while
17. $TXs \leftarrow$ mempool.pop() $\triangleright$ attack point-I
18. block ← sign($TXs$, wiggle) $\triangleright$ seal a block
19. broadcast(block) $\triangleright$ send the block
20. end while
21. end procedure
22.
23. procedure Verify($sealer_i$) $\triangleright$ verify a block
24. ¬sign_recently($sealer_i$, block.number)
25. diff = header.difficulty
26. if (diff == nil) then $\triangleright$ not nil
27. return false
28. else if (diff = 1 && diff ! = 2) then $\triangleright$ 1 or 2
29. return false
30. else then
31. return true $\triangleright$ block is valid
32. end procedure
33.
34. procedure sign_recently($sealer$, number)
35. for (i < |recents|) do $\triangleright$ amongst recent sealer, wait
36. if ($sealer = =$ recents[i])
37. limit = |sealers| + 1
38. if (number < limit && number > limit) then
39. return false
40. return true
41. end for
42. end procedure
43.
Type-II attack against them will generate very different results, as discussed in Section 5.

**Experiment Configuration.** To test the feasibility of our attacks, we clone the source code from Geth [5] and apply our attacking logic to it. Then, we run a private PoA network in an isolation environment. Our experiments run on Centos 7 x86_64 at a virtual private server (VPS) located in California, US, with hardware configurations of 4GB RAM, 80GB disk storage, and 3000GB bandwidth capacity. We create several sets of virtual nodes in this VPS to simulate independent consensus sealers. Then, we start our attacks and investigation. The details are discussed in the following sections.

4 BREAKING TRANSACTION ORDERS

In this section, we show how to apply the Type-I attack against Aura and Clique. We provide the experiment configurations, logic, results, and attacking impacts, respectively.

**Experimental Logic.** A transaction is required to experience four stages from user’s submission to final confirmation by sealers: (i) A user initiates a transaction with the signature signed by her private key, and then sends the transaction to blockchain nodes. (ii) The blockchain node verifies this transaction and adds it to the local transaction pool. Then, the node broadcasts the transaction to peers. The peer nodes perform the same procedures by validating and moving transactions into their transaction pools. (iii) As a specific type of node, the blockchain sealer receives transactions and packages them into a block. Sealers have the flexibility to decide transaction orders. Taken Geth [27] as an example, transactions are ordered by gas price and nonce value. Generally, the higher the gas fee is provided, the faster a transaction enters the block. (iv) As mining nodes receive the new block from their peers, they will re-execute all transactions inside the block. When a majority of nodes receive the same block, and re-execute included transactions, the block is deemed as confirmed and becomes immutable (together with its contained transactions). This ensures that all blockchain nodes have agreed on the same history of transactions.

4.1 Running the Attack-I

**Attack Simulation.** In our experiments, the attacker is assumed to be a profitable sealer with the intention to manipulate transactions by frontrunning and inserting. To be simple, we denote all these profitable transactions as frontrunning transactions for generality. The attacker is empowered with capabilities to find arbitrage opportunities and accordingly generate a frontrunning transaction (refer to [24, 33] for ways to create such transactions). Then, the attacker inserts her own transaction before/after the user’s transaction during the procedure of sealing a block.

To launch the attack, an attacker actively checks arbitrage opportunities. Afterward, the attacker issues one transaction \( T_{x_1} \) and attempts to insert \( T_{x_3} \) at an advantageous position in the mempool (equiv. a fronted position). Here, the mempool requires slight modification from the code aspect. Then, it seals a block and broadcasts to other peers. The Type-I attack is considered successful if a block contains a list of transactions that the users’ transaction is ordered after \( T_{x_3} \). It means that we have successfully allocated the attacker’s frontrunning transaction before a targeted transaction in the queue. Then, this block is successfully verified and confirmed by the majority of honest miners. To simulate this attack, we present a simple way to apply the frontrunning attack by sending a series of transactions \( T_{x_1}, T_{x_2}, ..., T_{x_9} \) to a miner who will generate a frontrunning transaction \( T_{x_3} \). To cover accidents and achieve a fair testing result, we repeat tests for six times.

4.2 Attacking Result Analysis

In the primary version of experiments, we conduct tests on a single node to check the attacking feasibility. Here, both Aura and Clique show a very similar result (see Tab.1). Experiment 1 shows the original order of the transaction list \( (T_{x_1}, T_{x_2}, T_{x_3}, T_{x_4}) \) while Experiments 2-6 show the results under our attacks. From these results, we observe that the frontrunning transaction \( T_{x_3} \) can be arbitrarily inserted to any position between these transactions. Even worse, the transaction list \( (T_{x_3}, T_{x_2}, T_{x_1}, T_{x_4}) \) can be re-arranged in an arbitrary order \( (T_{x_3}, T_{x_1}, T_{x_2}, T_{x_4}) \) by a profitable in-turn sealer.

Further, we extend our attack to multiple nodes (cf. the blue line without Attack-II of each sub-figure in Fig.3). Our applied attacks...
Table 1: Attack-I Experiment Results

| Set         | Time Slots |
|-------------|------------|
| Experiment 1| Tx1        |
|             | Tx2        |
|             | Tx3        |
|             | Tx4        |
| Experiment 2| Tx5        |
|             | Tx6        |
|             | Tx2        |
|             | Tx3        |
|             | Tx4        |
| Experiment 3| Tx1        |
|             | Tx5        |
|             | Tx2        |
|             | Tx3        |
|             | Tx4        |
| Experiment 4| Tx1        |
|             | Tx2        |
|             | Tx3        |
|             | Tx4        |
| Experiment 5| Tx1        |
|             | Tx2        |
|             | Tx3        |
|             | Tx4        |
| Experiment 6| Tx2        |
|             | Tx4        |
|             | Tx5        |
|             | Tx3        |
|             | Tx6        |
|             | Tx1        |

Transactions with Green background are profitable transactions.

Towards Aura and Clique both perform effective. In Type-I case, all sealers behave honestly but can pursue more revenues. Firstly, we fix the number of sealers as 9. The success rate of this attack is approximate 11% (Fig.3.b/d), proved by its absolute number of victim transactions (Fig.3.a/c). Then, we investigate the situations under dynamic committee where the size of committee ranges as [3,9,18,27]. The result of associated success rates is inversely proportional to participated nodes (Fig.3.e/f/g/h). This is easy to understand when we notice a fact: only the in-turn sealer can successfully arbitrarily insert her frontrunning transaction. Otherwise, the proposed block by an edge-turn sealer cannot out-complete an in-turn sealer due to their differences in difficulty. Thus, the more participants get involved in the consensus, the less opportunity an attacker’s block could be accepted. Our attack towards Aura and towards Clique output the same results in terms of both victim transactions and their success rates, indicating that Type-I is a general attack that can be applied to all PoA implementations based on the current knowledge.

Root cause. Typically, the disorder of transactions comes twofold. Firstly, even if an honest sealer follows a fixed order rule, the final result may still be disordered. In a peer-to-peer network, the same set of transactions may arrive at different sealers with entirely different orders due to the network delay. This situation happens with a non-negligible probability before the final decision of the consensus agreement. Inconsistent sequences status may exist for a long time, which leaves enough time for conducting profitable activities by in-turn sealers. Secondly, as mentioned earlier, the sealer can decide how to sort transactions. When a profitable sealer is mining, she can re-arrange transactions in an arbitrary order within her block to obtain the best interests. Meanwhile, other honest sealers cannot distinguish who is the profitable sealer and whether she has done evil.

In the current version of Aura and Clique, their implementations lack of a verification mechanism of transaction orders. In Aura, the order of transactions depends on the preference or discretion of a sealer (see Algm.1 line 12). Without any order-check mechanism in the verification procedure, a sealer can always pursue the highest transaction fees by rearranging the order of received transactions. Similarly, the implementation of Clique shares the same transaction order mechanism (Algm.2 line 16). Despite existing PoA algorithms prescribing specific rules for consensus, guarantees are absent on the relationship between the sequence of incoming transactions and their final order published in the ledger. Thus, PoA algorithms cannot promise transaction-order-fairness [22].

5 BREAKING SEALER TURNS

Experimental Logic. PoA consensus algorithm relies on a rotation schema, where sealers take turns to mine blocks in different rounds based on the time and block number. In PoA, the time is divided into steps, and one of the authorities is elected as the leader to produce a blockchain in each step. The leader, an in-turn sealer, can generate a block exactly in her turn, while an edge-turn sealer can also produce blocks. By the design and assumption of PoA algorithms, these sealers follow a fixed formula (Algm.1 line 11 and Algm.2 line 11) to calculate their turns. An in-turn sealer can propose a block with a high priority of the weight. Edge-turn sealers with the secondary priority have to produce blocks when the leader fails. In this series of experiments, we test the potentiality of extracting extra benefits of those edge-turn sealers. In every single Type-II attack, we firstly sleep the legal in-turn sealer to activate competitive edge-turn sealers. We separately adjust the number of sent transactions/blocks and the size of participated committee to simulate different real scenarios. In each hybrid attack (Type-I+Type-II), we keep the legal in-turn sealer alive and conduct the same experimental procedures for evaluation. Details are presented as follows.

5.1 Running the Attack-II

Attack Simulation. To enable the capability of frontrunning blocks, we allow an attacker to tamper with a mining client (slightly modify the source code at the delay time field 4). The modified client can keep sealing a block in every round without considering the condition of block generation (always sending 0-time delayed blocks). Notably, only the attacker’s code needs to change, while all the peer nodes are still supposed to be benign without modifications.

Details of our modifications in Aura and Clique are shown as below: (i) Change the UNIX time of the local client (Algm.1 line 9) or delete the conditional expressions (Algm.1 line 11 ) to ensure the profitable sealer can always seal a block in each round. (ii) Delete all the conditions of producing a block in the non-leader setting (Algm.2 line 14–17), and allow an edge-turn sealer to mine instantly in every round with the score 1. Afterward, the profitable sealer re-compiles the source code to obtain a heterogeneous client. Then, both Aura and Clique conduct the following attack simulations.

The first experiment mainly evaluates victim transactions (with the additional Type-II attack compared to the previous section) and blocks along with an increasing number of total transactions/blocks (marked by x-axis in Fig.3, which is based on a time-increased line). Here, victim blocks represent the blocks being frontrun by profitable edge-turn sealers, whereas the in-turn blocks cannot be deemed as the victim blocks.

To achieve this simulation, we firstly set the 9 mining nodes and 1 boot node in our network. The mining nodes can synchronize

4The term single indicates that, if we merely test the Type-II attack, an in-turn sealer should immediately keep sleeping to ensure the valid actions of edge-turn sealers. But when adding the Type-I attack, things become different. The Type-I attack relies on the manipulation of authorised sealers, no matter she is a legal in-turn sealer, or a profitable edge-turn sealer. Therefore, in case of testing the hybrid Type-I+Type-II attack, the original legal in-turn sealer should keep alive at an initial stage.

5Revised source code available at: https://github.com/poa-research/attacksimulator.
transactions and block data, and merge transactions into blocks. All those mining nodes, in our attack experiments, are granted with the ability to become edge-turn sealers with \( \text{diff} = 1 \), but not in-turn sealers with \( \text{diff} = 1 \) if out of their turns. Also, every legal in-turn sealer is intentionally set to sleep. This is to simulate the case of an in-turn sealer’s shutting down. As a result, edge-turn sealers stay active when solely testing the Type-II attack. In contrast, the boot node is only used for node discovery and does not need to synchronize chain data. All instances, including the malicious client, are connecting with each other. Then, we set the block generation rate at 3 seconds, which indicates that each block is generated at an average time of 3 seconds. An attacker is supposed to be a profitable edge-turn sealer with intentions to frontrun blocks, hidden in 9 mining nodes. The attacker then aims to produce a valid block out of her turn. To start the attack, the malicious sealer runs the modified client under the help of the boot node for joining the network and starting to mine. We allow all sealers to mine and users to continuously send transactions to any sealer at a constant rate (10 tx/s) within a fixed period (40 minutes). The Type-II attack is considered successful if a profitable (edge-turn) sealer can seal blocks before peered sealers.

The second experiment attempts to evaluate the relationship between the number of mining nodes and victim items. Again, we set the block generation rate at 3 seconds. Since PoA is more suitable for permissioned blockchain systems [2], we limit the size of mining nodes to an upper bound of 36. More specifically, we set 3 mining nodes and 1 boot node in our network. Then, the block generation rate is 3 seconds per block per node, while the transaction generation rate is 10 TPS per node. We adjust the participated mining nodes as \([9, 18, 27, 36]\) at the cumulative time of 40 minutes. The experiment will show the trend between the probability of sealing a block and the number of mining nodes (sealers).

5.2 Attacking Result Analysis.

In this part, we analyze the attacking results from two sides. Firstly, we fix the size of committee with 9 sealers (cf. Fig.3.a-d). We test the successful frontrunning transactions and corresponding success rates towards both Aura and Clique. Then, we put the focus on the dynamic committee by increasing participated sealers (Fig.3.e-h). Here, we provide detailed discussions.

Intuitively, in the first group of experiments, 800 blocks are produced in total. The created blocks will be uniformly distributed in 9 mining nodes (or sealers), meaning that each node will generate approximately 88 blocks. In Aura, the results of successful frontrunning blocks remain zero, satisfying our exception: Although a profitable sealer can generate 90 blocks every round, however, her proposal cannot pass the verification (cf. Alg.1 line 19-24) because Aura does not rely on the mechanism of difficulty as it in Clique. The adversary has negligible possibilities to frontrun blocks (Fig.3.a). Accordingly, the success rate turns to be zero as well (brown line in Fig.3.b). Then, if we enforce both Type-II and Type-I attack, the result of victim transactions and success rate keep consistent with results in the previous section, where each block should contain approximately 88, with a rate of 11% (11% \( \approx \frac{1}{9} \), blue line in Fig.3.a/b). On the contrary, in Clique, a profitable sealer can re-order any transactions since she can frontrun all blocks produced by edge-turn sealers in front of her under the absence of the leader (brown line in Fig.3.c/d). The indicates Clique is vulnerable to our attacks, especially for Type-II. Intuitively, victim transactions (blue line in Fig.3.c/d) follow the same trend with victim blocks as every edge-turn sealer, once become the leader by frontrunning, can dominate all included transactions within her blocks.

In the second group of the experiment of Aura, we observe that the number of victim blocks stays zero at all the time (brown line in Fig.3.e/f), which means that Type-II attack is ineffective to Aura no matter how many nodes are involved. A profitable sealer has no opportunities to conduct the block-level frontrunning attacks out of her turns. As for the victim transactions, the number of transactions mined by the profitable sealer and its associated rate always exist but start to decrease along with increased nodes (blue line in Fig.3.e/f). This is easy to understand: Suppose that there is a profitable sealer in the committee, even though she cannot frontrun the turn of others, she can still send frontrunning transactions within her legal turn. Similarly, the number of these frontrunning transactions declines when the number of participated sealers increases. This confirms our predictions that transactions are uniformly distributed in different blocks, decreasing the frontrunning blocks. In Clique, adding the Type-II attack will both affect the number of victim transactions and victim blocks in a positive relation (Fig.3.g/h). An interesting point is the success rate of victim transactions (adding Attack-II) is approximately equal to the sum of another two (victim block&victim Tx without Attack-II in Fig.3.h). This is because the count of victim transactions includes both sources from the in-turn sealer (her legal turn) and edge-turn sealers who successfully conduct frontrunning. In this case, a profitable sealer can always seize edge-turn sealers’ opportunities to seal the block with included transactions. PoA is thereby vulnerable to both Type-I and Type-II attacks.

Root cause. We have dived into the source code of both Aura and Clique, abstracted their operating workflow, and found the root cause resulting in the Type-II attack. The attack is ineffective to Aura whose consensus is designed from BFT protocols, totally different from Clique. We skip the discussion of Aura. The reason that the Type-II attack is effective to Clique lies in the incomplete design of identity-verification mechanism towards edge-turn sealers, which, unfortunately, indicates that the identity-verification towards an in-turn sealer applied to real implementations is valid, whereas similar checks towards edge-turn sealers are invalid, or at least, incomplete. It is clear that both the in-turn sealer (high priority) and edge-turn sealers (secondary priority) can produce blocks. When the in-turn sealer fails, edge-turn sealers start to become important. However, even though the protocol has checked\&passed \( \text{diff} = 1 \) for those edge-turn sealers, a serious fact has been ignored: multiple active sealers hold \( \text{diff} = 1 \). This is the most significant difference compared to the case with an in-turn sealer because only one eligible candidate holds \( \text{diff} = 2 \) in each round. Meanwhile, all edge-turn sealers are required to add a random delay during their block production. When the only one in-turn sealer fails, an adversary can easily break the normal turns of active edge-turn sealers by shortening the time delay (cf. Fig.2). Therefore, the lack of a complete
identity-verification mechanism towards edge-turn sealers enables profitable ones to break the fairness of normal rotation turns.

Beyond the fairness issues, we also notice that our order manipulation attacks can result in a decrease in performance. Here, the performance represents all valid transactions processed by benign block-producing sealers (excluding profitable sealers). A profitable sealer can rationally create many empty blocks for fast delivery, outpacing heavy transactions sent by peers. This will significantly hinder the normal procedure of packing real transactions from users. A user may have the feeling that she has waited for a long time whereas the transaction has still not been confirmed. In our predictions, a rational sealer will conduct such activities with high probability even if extra revenues are not as much as expected.

6 ANALYSIS AND DISCUSSION

This section discusses the impact of our attacks from the view of blockchain properties, comparisons with common attack vectors, attacking scope, and potential monetary loss.

6.1 Property Analysis

We consider three orthogonal properties\(^5\) (including consistency, liveness [32] and fairness [22]) that closely relate to the system security. Consistency is to ensure all honest nodes in the network decide on the same state while liveness guarantees all correct nodes eventually have decisions. These two properties refer to the intrinsic designs of a system that have been studied for decades. In contrast, fairness [22], as a new proposed property, has been overlooked for a long time due to the particularity of the miner mechanism. Fairness focuses on whether the sequence of transactions before being processed by miners are consistent with the final ordering. The property is used to prevent adversarial manipulation against the actual ordering of transactions in the mempool.

Our proposed attacks do not affect the properties of either consistency or liveness, while only breaking the fairness property. The reason lies in the assumption that we maximally limit the activities of “adversaries”. Traditional attacks generally empower adversaries with many out-competing abilities, such as cloning an instance [31] or falsifying identities. Even much, the attack may require further requirements on total synchronous network assumptions. In contrast, we try to obey assumptions as they have been defined in the initial stage. We merely allow the adversary (a profitable sealer) to conduct activities that have not been clarified, rather than violating any explicit rules. Since miners’ activities have not been seriously monitored or discussed, our attacks can be successfully implemented in many long-term running PoA-engined projects, and instances are listed in Tab.2.

6.2 Comparisons with Parallel Attacks

We consider three types of attacks threatening the security of PoA protocols: 51% attack, cloning attack and balance attack. 51% attack means an adversary can take control over 51% of sealers. Due to the committee being static and transparent, compromising half of committee members is theoretically feasible when it is on a small scale. Cloning attack [31] allows an adversary to clone the instance of a sealer (key pair, metadata). Two equivalent groups containing the same number of sealers are isolated to create hard forks. Similarly, balance attack [34] aims to profitably keep (at least) two branches growing at the same pace by partitioning the network into multiple zones. An adversary can dynamically wander across those balanced zones and select the most profitable one to maximize her revenues. As shown in the attached table, all these attacking strategies rely on a similar attacking strategy: creating double-spending transactions through a hard fork. This will break both the consistency and liveness, destroying the operation from underlying systems. Our attacks differ from these strategies, where a system can still work as normal. It only impacts a single targeted victim, rather than all users participating in the system. Further, the victims may not even be aware of whether they were attacked, namely, frontrun by others or not.

| Attack       | Consistency | Liveness | Fairness |
|--------------|-------------|----------|----------|
| Type-I       | ✓           | ✓        | x        |
| Type-II      | ✓           | ✓        | X        |
| 51%          | x           | x        | N/A      |
| Cloning      | x           | x        | N/A      |
| Balance      | x           | x        | N/A      |

6.3 Attacking Impacts

We have explored 2,500+ in the wild projects from Github repositories and selected a small set of mainstream projects (10+, see Tab.2) as instances. The reason to select these projects is that most of the current PoA implementations follow very similar protocol designs, or even directly fork the code from other projects. We only narrow down our focus to mainstream implementations or high market cap projects that may be influential to communities. Specifically, we dive into the source code of exemplified projects, detecting vulnerabilities and positioning associated locations. We mainly analyze the Type-II attack rather than Type-I as the Type-I attack can be applied to almost every PoA implementation (which has authority nodes). Here, we conclude our insights.

Insight 1. The Type-II attack is effective in most Clique-based (including variants) PoA implementations. As discussed before, the current version of Clique (in Go-Ethereum) depends on a loose verification mechanism where the edge-turn sealers’ identities have not been seriously checked. Merely relying on difficulty is not reliable, as a profitable sealer can easily keep sending the 0-time delays while peer nodes cannot distinguish whether she is a normal edge-turn sealer who legally wins the game or a profitable adversary using falsified methods.

Insight 2. Many existing projects, even for well-known projects such as Ethereum Classic [29], ConsenSys [35], Binance smart chain [36] with high market caps, inherit the code from Go-Ethereum client (Geth). The vulnerabilities detected in these projects are witnessed in the same line of source code (e.g., see Tab.2 line No.658). Attacks taking effect in Go-Ethereum will also impact these projects with similar damage.

Insight 3. Several Clique-based variants can avoid being frontrun by profitable sealers (Type-II) due to the constricitions on sealers’ behaviours. In most mainstream consensus algorithms, the
behaviours of miners are free of supervised (a.k.a. dark forest [26]) where they can reorganize transactions according to their preferences. A clear set of predefined rules for these miners is necessary for avoiding the potential ambiguous activities. These rules can define the methods of sorting transactions, such as relying on paid tips, or transaction appearance.

**Insight 4.** Combining PoA with existing consensus protocols cannot mitigate the ordering attacks. Additional top-up algorithms can only resolve the problem on how to enter the committee, rather than how to limit members’ activities. For instance, BSC integrates DPoS and PoA [12] to a new hybrid algorithm called Proof of Staked Authority, where validators (equiv. sealers in the context of this work) take turns to produce blocks in a PoA manner (similar to Go-Ethereum’s Clique). However, the combined DPoS scheme is merely utilized to select a set of eligible authorities based on their stakes. The person who holds more deposits will earn the opportunity of entering the committee.

### 6.4 Monetary Loss in Marketcap

As of November 2021, there are 13,890 crypto projects existing in decentralized markets, reaching up to a total market cap of 2,882 billion USD, with a 24h volume of 135 billion USD [CoinMarketCap, Nov 2021]. This breaks the historical records, and the corresponding market cap becomes the highest value ever. The close relation to finance attracts adversaries to keep their eyes on every feasible attack against these cryptocurrencies. A successfully launched attack can benefit these attackers with thousands of returns. To emphasize the influence on markets, we evaluate the potential monetary loss of PoA projects affected by our attacks.

We first count several leading examples (cf. Tab.2). The total market value of impacted projects is over 681,087 million USD\(^6\), which mainly comes from Ethereum (81.4\%) and Binance (15.4\%). Meanwhile, Ethereum and Binance smart chain are also the most prevailing smart contract-enabled platforms that support up to more than 2,028 decentralized applications (DApps)\(^7\). The attacks against these DApps can cause hundreds of times of loss than their basic single-chain values. Despite our attacks merely target to profitable miners without breaking normal system operations, the impact can still result in huge monetary loss, especially in financial DApps on top of these platforms, such as DeFi [48], NFT [49], etc.

Here, we dive into two instances. The first case focuses on DeFi. PancakeSwap [50] is a decentralized exchange (DEX) built on top of Binance smart chain. Currently, PancakeSwap reaches up to 28,478 million USD. The attack towards the underlying BSC will

\(^6\)We capture several leading projects with associated market caps here: Ethereum (554,476,866,559 USD), Binance (104,980,840,031 USD), Polygon (12,413,978,032 USD), Ethereum Classic (7,874,100,490 USD), GoChain (46,826,436 USD), Denny (1,199,666,501 USD), PoA network 179,093,222 USD [As of Nov 15, 2021].

\(^7\)Among these DApps, 1,963 of them are built on top of Ethereum while 65 are on Binance smart chain. Data captured from https://www.stateofthedapps.com/dapps.
Table 2: Attacking Impacts (towards Type-II Attack)

| Project       | Client | Location | Notes                     |
|---------------|--------|----------|---------------------------|
| Go-Ethereum   | Clique | No.293 in [27] | No checks on identity |
| BSC           | Clique | No.403 in [36] | No checks on identity |
| Polygon (MATIC) | Clique | No.403 in [37] | Forked from Go-Ethereum |
| OpenEthereum  | Clique | No.658 in [38] | No checks on identity |
| PoA network   | Clique | No.658 in [39] | Forked from Open Ethereum |
| Ethereum Classic | Clique | No.403 in [35] | Forked from Go-Ethereum |
| ConsenSys     | Clique | No.403 in [35] | Forked from Go-Ethereum |
| GoChain       | Clique | No.315 in [40] | Forked from Go-Ethereum |
| Daisy         | Clique | No.305 in [41] | No checks on mining |
| Olecoin       | Clique | No.293 in [30] | Forked from Go-Ethereum |
| EEX           | Clique | - in [42]    | Claim to use original Clique |
| HPB           | Clique | No.126 in [43] | No checks on mining |

1. Data accessed in November, 2021.
2. Projects in Yellow are vulnerable to the attack.
3. Projects without the background can resist the attack.

also significantly affect PancakeSwap. Users can trade tokens in PancakeSwap as in a centralized exchange, but differently, no third parties participate in the system. As a typical DeFi product, users can use the liquidity pool [51] to exchange different tokens. When some users exchange tokens for another type of tokens, other users can earn rewards by staking tokens in the liquidity pool. At the same time, any exchange causes the slippage to move in a reverse direction. Assume that game players send their trades in this liquidity pool. A frontrunning transaction can be processed at a lower price than others. At this time, a profitable sealer may take advantage of trading at a good price. The second case discusses a real scenario of auctions. Suppose that there is an auction event for many valuable art products (based on NFT [49]), the winner for this bid can obtain thousands of returns in the future with a high probability. A real-world NFT attacking example can be found at [52]. A profitable sealer, with aims to win the bid, can launch order manipulation attacks (either by Type-I or Type-II) to outpace competitors. This results in the unfairness of entire participants.

7 COUNTERMEASURES

In this section, we give countermeasures to mitigate the unfairness issue in PoA protocols. Both recommended methods hold the same goal for guaranteeing a proper turn of sealers and transactions.

Intuitive Fix. As analyzed before, we observe that the Type-I unfairness is caused by authorities from the block proposers. It is hard to be fixed without introducing external techniques like using zero-knowledge proof to hide the transaction ordering against miners. In contrast, the Type-II unfairness comes from the incautious design of code, especially for the verification procedure. An intuitive method of avoiding vulnerability is to ensure that the identities of edge-turn sealers are properly matched before proposing blocks when the leader is absent. Equivalently, modifying the function of \( \text{Verify}(\ldots, \text{difficulty}) \) in Algm.2 line 26 (by adding identification checks) to \( \text{Verifying}(\ldots, \text{difficulty}, \text{id}) \), as it is in Algm.1 line 20, can work well against the issue. This type of fix is the simplest way to mitigate the Type-II attack, resolving the root cause from the source code. The fix can be further applied to the projects that follow similar designs or adopt the original Git-code forked from Ethereum Clique. Here, we have also submitted our recommend fix to an in the wild project, with details referring to Appendix A.

It is worth mentioning that, however, this type of fix is not a perfect solution because it decreases the system availability to some extent. In original settings, when the leader fails, all the edge-turn sealers produce blocks, and the fastest one will become the winner. This is the most economical way for the system. Once adding the identification check, the closest edge-turn sealer will produce the block with a significantly higher accepting probability than others, which obviously decreases the competition among those sealers who can produce blocks. Such a situation is the meaning of availability decreasing. Moreover, adding identification checks towards edge-turn sealers brings more computing workload.

Physical Randomness. Existing PoA implementations set the difficulty in a deterministic way (either as 1 or 2). This makes an adversary precisely predicate the position of sealers, especially in the Clique algorithm. An example is to further bring physical randomness into the system (see Algm.3 in Appendix A, source code in [53]). The physical random numbers are periodically sampled from embedded FPGA hardware [54], which provides completely unpredictable randomness compared to pseudo-randomness (generated by a fixed seed). The project resolves the unfairness issue caused by the Type-II attack in two key procedures. Firstly, it adds the verification process of the edge-turn sealers’ identities (cf. Algm.3 line 18). It is consistent with the solution discussed previously. Secondly, it embeds the hardware-based randomness into the verification step. The edge-turn sealers need to involve the sampled randomness when producing the block, and sealed sealer who receives the block should check its validity by sampling the same randomness from the same source (Algm.3 line 3/5/16/20). This ensures the following events: (i) the edge-turn sealer is correctly elected within her turn by verifying the identity; and (2) the adversary cannot fake enrolled sealers’ identities by cloning open-sourced information because physical randomness is unpredictable, which acts as a one-time pad. The method avoids the threat of the Type-II attack and guarantees sealers’ fairness of their incoming turns. However, this solution cannot mitigate the general Type-I attack.

Crypto-based Solution. Cryptographic primitives can help to mitigate the unfairness problem in multiple aspects. For the Type-I attack, we introduce three types of crypto-based methods to minimize the loss of MEV. The first approach [55] employs threshold encryption to distribute the power of a single sealer into multiple cooperated sealers. The second method [56] is to use commit-reveal schemes to make miners’ behaviours accountable. The third way [57] leverages the hardware-assisted trusted execution environment to ensure the integrity of program execution. These methods...
add input causality to the ordering process, improving the cost of behaving profit from miners.

For the Type-II attack, a promising way is to realize the confidential sealers rotation by hiding the identities of sealers. Traditional ways of leader election that rely on the static formula (e.g., Alg.1 line 4) is a public procedure to all participants. An adversary can easily predict who will become the leader in a certain round and launch a single-point attack like DDOS. Hiding the turn or identities of sealers can reduce the risk of being compromised by external attackers. To achieve confidentiality, we introduce the technique of verifiable random function (VRF) [58] as an illustrative instance (see details in Appendix C) to demonstrate how to seal the identities of candidates properly.

8 RELATED WORK

In this section, we focus on the intrinsic property of PoA in terms of its permissionless setting and security property. Then, we review the fairness notion in the context of blockchain.

BFT in Permissionless Settings. BFT protocols [59] are the most prevailing consensus family in distributed systems. BFT protocols provide guarantees of strong safety and instant finality, even under a partial synchronous network. BFT protocols are permissioned, splitting network participants into two types, committee members (perform consensus) and clients (synchronize information). The committee is fixed, in most cases, static as well. In contrast, classic blockchain systems use the Nakamoto consensus (PoW and variants) [60], a family of permissionless protocols that rely on a probabilistic way for consistency and better liveness. However, this method is inefficient, as every block has to be buried sufficiently deep to avoid being reversed. Adapting BFT to permissionless systems is an alternative solution [61] to balance consistency and performance. These systems select a group of authorities via trusted ways to run consensus, same as they act in original designs. One major difference lies in their dynamic committees, where members can conditionally join and leave following predefined rules. Exemplified protocols include the PoA in Ethereum, dBFT in NEO [62], vote-based BFT in Algorand [63] and variant PBFT in Thunderella [64]. Meanwhile, many systems adopt a hybrid or two-layer architecture to further leverage advantages of both consensus such as PoS+PBFT in Tendermint [65], etc.

Security Analysis in PoA. Angelis et al. [2] provide the first discussion of PoA by comparing it with PBFT under the CAP theorem. They point out the threat of inconsistency under an asynchronous network. Shi et al. [66] further explore the situation when an adversary is able to maintain two branches in equal lengths. The malicious sealer only needs to falsify the timestamp when claiming its priority. Based on a similar idea, Parinya et al. [31] propose the cloning attack by allowing an adversary sealer to clone her key pair into peered nodes. Committee members isolated in different groups cannot have negotiations, and as a result, hard forks happen without chances to converge. The attack demonstrates that PoA is not secure under common assumptions of partial synchronous network and the presence of malicious sealers. Meanwhile, several analyses utilizing different methodologies also put focus on PoA protocols. Liu et al. [67] leverage Markov Decision Process to quantitatively measure the balance-off between security and scalability. The proposed framework is applied to VeChainThor’s PoA algorithm. Cyril et al. [68] propose a load-balancer middleware to test Ethereum clients, covering Geth, Parity, and Besu (Hyperledger). They evaluate clients from the perspectives of behaviours and performance, and discuss bottlenecks with reasons. Toyoda et al. [69] provide a function-level analysis by developing a Golang’s profiling tool pprof. They find the targeted functions that slow the confirmation and hinder the concurrency. Besides, other generic tools for the testing can also be found in [70, 71].

Fairness in Blockchain. The term of fairness refers to block-level security that may threaten the chain stability. It is used in block mining in PoW protocols, indicating that the mining rewards should be proportional to the node’s contribution of computing power. A miner cannot perform selfish mining [72] for more payments than its deserved rewards (faire share [22]). In this sense, a set of protocols, including FruitChain [73], SmartPool [74], Miller’s work [75], have been proposed to achieve fairness by strengthening protocols from structure designs to incentive mechanisms. Later, the fairness notion focus on transaction-level honesty. Helix et al. [76] propose the way of using threshold encryption to select a random set of pending transactions during the block creation. Meanwhile, several DAG-based blockchain protocols [32], such as Hashgraph [77] and Prism [78], also provide their solutions to sort transactions in a linear order, rather than letting them staying in disordered status [79, 80]. Further, Kelkar et al. formally define the transaction-ordering fairness in both permissioned [22] and permissionless settings [23]. They present the strict models and related analyses, with aims to prevent adversarial manipulation of the actual transaction ordering.

9 CONCLUSION

PoA algorithms have been adopted by numerous blockchain projects. Their design principles used by different variants are basically the same. However, the lack of restrictions on node behaviours and further authentication of identities (regarding implementation level) make them vulnerable to being tampered within the order of transactions and blocks. These projects reach a market cap of approximately 681, 087 million USD. Corresponding high values will, inevitably, result in severe loss of real money once attacked by adversaries. In this paper, we explore such vulnerabilities, denoted as the unfairness issue, existing in PoA protocols. We have classified two types of attacks based on a comprehensive review of 10+ source codes of influential projects among 2,500+ in the wild ones. The first attack (transaction-unfairness) breaks the legal order (equiv. first-in-first-out) of transactions, while the second attack (block-unfairness) targets to break the normal rotation of sealers, which can produce high-priority blocks. Both of them merely rely on honest-but-profitable sealer assumption, which may happen in the real world with a high probability. We apply our attacks to two most prevailing implementations, including Aura and Clique, and evaluate their effectiveness. In addition, we propose several ways of remedies by diving into their source code and analysing root causes. To our knowledge, this paper is the first work to investigate the unfairness issue of PoA algorithms, enlightening the following designs and providing potential insights to communities.
Acknowledgement. With the highest importance, we want to express our sincere thanks to expert researchers who first formally propose the notion of fairness in the context of blockchain and developers who have publicly released such various PoA implementations. Also, we thank Xinrui Zhang (SUSTech) for her help. Lastly, Qi Wang and Rujia Li are partially supported by the Shenzhen Fundamental Research Programs under Grant No.2020925154814002.

REFERENCES
[1] Proof-of-authority chains. https://openerherum.github.io/Proof-of-Authority-Chains, 2016.
[2] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Pibf vs proof-of-authority: Applying the cap theorem to permissioned blockchain. 2018.
[3] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 15(1)(2014):1–32. 2014.
[4] Ethereum parity. https://www.parity.io/, 2019.
[5] Ethereum geth. https://geth.ethereum.org/, 2021.
[6] Kovan testnet. https://kovan-testnet.github.io/website/, 2021.
[7] Ingo Weber, Qinghua Lu, An Binh Tran, et al. A platform architecture for multi-tenant blockchain-based systems. In 2019 IEEE International Conference on Software Architectures (ICSA), pages 101–110. IEEE, 2019.
[8] Vechain blockchain. https://www.vechain.org/, 2021.
[9] Clique deployment. https://docs.microsoft.com/en-us/azure/blockchain/templates/ethereum-poa-deployment, 2021.
[10] Igor Barinov, Vadim Arasev, Andreas Fackler, Vladimir Komendantskiy, Andrew Gross, Alexander Kolotov, and Daria Isakova. Proof of stake decentralized autonomous organization. 2019.
[11] Clique. https://github.com/ethereum/EIPs/issues/225, 2021.
[12] Poa network. https://github.com/ethereum/go-ethereum/blob/master/consensus/poa.md, 2021.
[13] Ethereum geth. https://example.com/en-us/ethereum/blockchain/develop/ethereum-poa-development, 2021.
[14] Poa network. https://github.com/ethereum/go-ethereum/blob/master/consensus/poa.md, 2021.
[15] Clique in go-ethereum. https://github.com/ethereum/go-ethereum/blob/master/consensus/clique/clique.go, 2021.
[16] High performance blockchain (hpb) – consensus. https://github.com/hpb-project/go-hpb/tree/master/consensus, 2021.
[17] Proof-of-authority chains. https://goerli.net/, 2021.
[18] Chain generation process in hpb. https://github.com/hpb-project/go-hpb/blob/master/consensus/pbm/chain_generation.go, 2021.
[19] Clique in go-ethereum. https://github.com/ethereum/go-ethereum/blob/master/consensus/clique/clique.go, 2021.
[20] Clique in openethereum. https://github.com/openethereum/openethereum/blob/main/consensus/clique/clique.go, 2021.
[21] Qin Wang and Rujia Li. Non-fungible token (nft) overview, evaluation, opportunities and challenges. arXiv preprint arXiv:2105.07447, 2021.
[22] Clique in gochain. https://github.com/etclabscore/core-geth/blob/master/consensus/clique/clique.go, 2021.
[23] Novacrypt. https://pancakeswap.finance/, 2021.
[24] Proof-of-staked-authority. https://github.com/etclabscore/core-geth/blob/master/consensus/prometheus/prometheus.go, 2021.
[25] Blockchain networks on aws in minutes with kaleido a consensys solution. https://pancakeswap.finance/, 2021.
[26] Clique in gochain. https://github.com/etclabscore/core-geth/blob/master/consensus/clique/clique.go, 2021.
[27] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. Non-fungible token (nft) overview, evaluation, opportunities and challenges. arXiv preprint arXiv:2105.07447, 2021.
[64] Rafael Pass and Elaine Shi. Thunderrealla: Blockchains with optimistic instant confirmation. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 3–33. Springer, 2018.

[65] Yakobole Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci-Pietrigianni. Correctness and fairness of tendermint-core blockchains. arXiv preprint arXiv:1805.08429, 2018.

[66] Elaine Shi. Analysis of deterministic longest-chain protocols. In 2019 IEEE 32nd Computer Security Foundations Symposium (CSF), pages 122–1223. IEEE, 2019.

[67] Xuefeng Liu, Gansen Zhao, Xinxing 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.

[68] Cyril Naves Samuel, Severine Gloc, Francois Verder, and Patricia Guittion-Oudanou. Choice of ethereum clients for private blockchain: Assessment from proof of authority perspective. In 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pages 1–5. IEEE, 2021.

[69] Kentaro Teredia, Koji Machi, Yutaka Ohtake, and Allam N Zhang. Function-level bottlenecks: Analysis of private-of-authority etherum blockchain. IEEE Access, 8:141611–141621, 2020.

[70] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD), pages 1085–1100, 2017.

[71] Sara Rouhani and Ralf Deters. Performance analysis of ethereum transactions in private blockchain. In 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), pages 70–74. IEEE, 2017.

[72] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In International conference on Financial Cryptography and Data Security (FC), pages 436–454. Springer, 2014.

[73] Rafael Pass and Elaine Shi. Fructichains: A fair blockchain. In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), pages 315–324, 2017.

[74] Loi Luu, Yaron Velner, Jason Teutsch, and Prateek Saxena. Smartpool: Practical decentralized pooled mining. In 26th USENIX Security Symposium (USENIX Security), pages 1409–1426, 2017.

[75] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable long-lived hardware functions. In 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 680–691, 2015.

[76] Avi Asayag, Gad Cohen, Inno Grasevsky, et al. A fair consensus protocol for transaction ordering. In 2018 IEEE 26th International Conference on Network Protocols (ICNP), pages 55–65. IEEE, 2018.

[77] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 585–602, 2019.

[78] Serguei Popov. The tangle. White paper, 1(3), 2018.

[79] Qin Wang and Ruija Li. A weak consensus algorithm and its application to high-performance blockchain. In IEEE INFOCOM 2020 Conference on Computer Communications, pages 1–10. IEEE, 2021.

[80] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), pages 51–68, 2017.

APPENDIX A. HPB ALGORITHMS

We provide the core procedures (mainly focus on the Propose and Verify steps) of HPB [19], an PoA-engined project (see Alglm.3). Reasons to capture these two functions are: (i) Both Propose and Verify functions invoke the hardware-based randomness from the same source (Algml.3 line 3/16). This ensures that the identified items (e.g., trace/rag) can keep consistent, proving that the incoming sealer correctly enters her legal turn. (ii) The root cause of Type-II vulnerability is the absence of identity checks towards incoming sealers. We thereby suggest adding associated check procedures (e.g., Algml.3 line 16-21, 27). Similar to the solution of Intuitive Fix, the key revision is to add the verification of sealers’ identities into the current version. The difference lies in the usage of physical random numbers sampled from FPGA hardware [54]. The hardware-based randomness contributes to the system from two sides. Firstly, it is set as the system as a checking parameter, guaranteeing that every peer node (sample tag) can verify the block producer’s identity (sample trace). Secondly, randomness is used in the sealer rotation step (Algml.3 line 5), where any participant cannot predict who will become the incoming block producer. This fix has been submitted to the HPB official team. Note that after discussing with the team, they have finally accepted part of our suggestions – fixing the synchronization problem, to ensure the smooth operations of the difficulty verification (cf. [28]).

Algorithm 3 HPB with Hardware Randomness

1: procedure Propose(sealer,)
2: \hspace*{1em} \textbf{while} (true) do
3: \hspace*{2em} \textbf{if} (trace \leftrightarrow \text{sample (Hardware)}) \hspace{5em} \triangleright \text{propose a block}
4: \hspace*{2em} \textbf{while} (true) do
5: \hspace*{3em} \textbf{end while}

APPENDIX B. NOTATIONS

In this part, we highlight several terms used in this paper. We use very easy-understand notations and try to keep consistent with previous similar definitions in concurrent studies such as [2, 31]. Meanwhile, we further provide more precise definitions than existing statements in terms of sealers’ behaviours (honest and benign). Our work is exactly based on these slight differences.

APPENDIX C. INSTANCE USING VRF

Confidential Election via VRF: Verifiable random function (VRF) [58] is a typical public-key pseudo-random function used to prove that

\[ \text{Algorithm 3 HPB with Hardware Randomness} \]

\begin{align*}
1: & \text{ procedure Propose(sealer,)} \\
2: & \hspace{1em} \textbf{while} (true) do \\
3: & \hspace{2em} \textbf{if} (trace \leftrightarrow \text{sample (Hardware)}) \hspace{5em} \triangleright \text{propose a block} \\
4: & \hspace{2em} \textbf{while} (true) do \\
5: & \hspace{3em} \textbf{end while} \\
6: & \textbf{end while} \\
7: & \textbf{procedure Verify(sealer,)} \hspace{5em} \triangleright \text{verify a block} \\
8: & \hspace{1em} \textbf{if} (trace \leftrightarrow \text{sample (Hardware)}) \hspace{5em} \triangleright \text{propose a block} \\
9: & \hspace{2em} \textbf{end procedure} \\
10: & \hspace{1em} \textbf{if} (diff \leftrightarrow \text{nil}) \hspace{5em} \triangleright \text{not nil} \\
11: & \hspace{2em} \textbf{return} \hspace{5em} \triangleright \text{return true} \\
12: & \hspace{2em} \textbf{end procedure} \\
13: & \hspace{1em} \textbf{if} (diff \leftrightarrow \text{nil}) \hspace{5em} \triangleright \text{return false} \\
14: & \hspace{2em} \textbf{else if} (diff \leftrightarrow \text{nil}) \hspace{5em} \triangleright \text{return false} \\
15: & \hspace{2em} \textbf{else if} (diff \leftrightarrow \text{nil}) \hspace{5em} \triangleright \text{patch} \\
16: & \hspace{2em} \textbf{end procedure} \\
17: & \hspace{1em} \textbf{end procedure}\end{align*}
Table 3: Notations, consistent with [31]

| Symbol | Role          | Alias                  | Functionalities                                                                 |
|--------|---------------|------------------------|---------------------------------------------------------------------------------|
| $N_i$  | Sealer        | Out-of-turn sealer     | Committee members, cannot produce blocks                                         |
| $N_L$  | Leader        | In-turn sealer         | Produce the block in the highest priority                                        |
| $N_e$  | Sealer        | Edge-turn sealer       | Produce the block in the secondary priority                                       |
| $f(.)$ | Formula       |                        | Rules deciding how to select the leader                                           |
| $B_i$  | Out-of-turn block |                        | Blocks created by out-of-turn sealers                                           |
| $B_L$  | In-of-turn block |                        | Blocks produced by in/edge-turn sealers                                         |

- Honest sealer: The sealer who obey predefined rules, but can act arbitrary for out-of-scopes rules, such as being profitable.
- Benign sealer: The sealer who act honestly without own willingness, faithfully following the predefined rules.

the outputs with proofs matches its associated inputs. The owner who holds the secret key can generate the proof from an input string, while others can verify two types of facts by using the public key and the proof: (i) whether the output is created by the owner who holds the secret key; (ii) whether the output is correctly calculated from the associated input. To be noted, Algorand [81] has also adopted VRF in its cryptographic sortition algorithm to randomly select the sealer while additionally hiding her identity. Here, we present the main steps of constructing VRF-based sealer rotation. Notations are defined as follows: $s$ is a negotiated seed for which corresponds to a specific block height; $pk$ and $sk$ are the sealer’s key pair; $hs$ is the random hash generated by the incoming sealer; $\pi$ is the evidence for the generated hash.

- $\text{VRFKeyGen}(\lambda) \rightarrow (sk, pk)$: The algorithm takes as input a security parameter $\lambda$, and outputs the secret/public key pair $(sk, pk)$.
- $\text{VRFHash}(sk, s) \rightarrow hs$: The algorithm takes as input $sk$ and a seed $s$, and outputs a fixed-length hash $hs$.
- $\text{VRFP proves}(sk, s) \rightarrow \pi$: The algorithm takes as input $sk$, $s$, and outputs the proof $\pi$ for $hs$.
- $\text{VRFVerify}(pk, s, hs, \pi) \rightarrow 0/1$: The algorithm takes as input $(pk, m, hs, \pi)$, and outputs the verification result, a boolean variable 0 or 1 (0/1 for invalid/valid, respectively).

Based on the properties of VRF, the proposed confidential scheme obtains additional benefits from three aspects. Firstly, VRF is pseudo-randomness, where an adversary cannot effectively distinguish the output of $\text{VRFHash}$ from a random string without the knowledge of corresponding $pk$ and $\pi$. This enables the sealer to be hidden from the public. Secondly, VRF is unique where one input can only produce a unique output, guaranteeing that attackers cannot falsify the identity of the sealer. Thirdly, VRF is collision-resistance where each output cannot be produced by two different inputs, ensuring adversaries cannot compromise benign sealers.

APPENDIX D. SCREENSHOTS OF TYPE-II ATTACK IN OUR EXPERIMENTS

Fig.4 and Fig.5 show the screenshot of a normal case on running the normal $Clique$ clients and a malicious client, respectively. In the normal case, the address 0xf719... is assumed to be owned by a benign sealer. In the attacking case, such a benign sealer changes her role to an honest-but-profitable sealer with the same address. In the first experiment, such a sealer operates the program based on a normal Geth client. We observe that only one block has been produced after the in-turn sealer abandons to mine, which means that this sealer cannot frontrun the block. In contrast, in the second experiment, the sealer is supposed to be a profitable sealer and starts to conduct the arbitrage strategy based on a maliciously modified client. We can see that the address 0xf719... in Fig.5 (with difficulty of 1) appears four times, frontrunning other edge-turn sealers in each rotation round. The results demonstrate that the profit-pursuing sealer has successfully launched the Type-II attack.

Here, we still keep the leader (with difficulty of 2) in the network, but can omit it when counting the result.

Figure 4: Screenshot: A normal case on running the $Clique$ client

Figure 5: Screenshot: Launching the Type-II attack on $Clique$ client