**Distributed Consensus**

# Simulator for Sleepy Consensus Protocol

SJTU 2017 Cornell Summer Workshop

**Instructor**

*Elaine Shi*

*Cornell University*

## Our Group:


- **Framework Team Members**

| | |
|---|---|
| Junxiang Huang | 841450297@qq.com |
| Yifei Pu | pkq2006@gmail.com |

- **Honest Node Team Members**

| | |
|---|---|
| Tiancheng Xie | wjxtcsgx@hotmail.com |
| Jiaheng Zhang | ZHANGJIAHENG@sjtu.edu.cn |
| Xiaotian You | youxiaotian@hotmail.com |
| Shuyang Tang | tangshuyang25@163.com |
| Chengyao Li | cyli2014@sjtu.edu.cn |

- **Adversary Members**

| | |
|---|---|
| Qingrong Chen | chenqingrong@sjtu.edu.cn |
| Ruisheng Cao | 211314@sjtu.edu.cn |
| Shiquan Zhang | zsq007@sjtu.edu.cn |
| Haochen Huang | hhc98598@189.cn |

- **Integrators**

| | |
|---|---|
| Feiyang Qiu | st.yeah@gmail.com |
| Lingkun Kong | klk316980786@sjtu.edu.cn |
| Lanqing Liu | sunnysunny@sjtu.edu.cn |
| Jialu Li | 790359064@qq.com |


- **Where to find our project?**

https://github.com/initc3/sleepysim SleepySim

# Table of Contents

## 1 Introduction

Consensus protocol serves as the core of distributed computing and also provides a foundational building block for cryptocurrency protocols. In traditional cryptocurrency schemes, proof-of-work (PoW) is leveraged to provide consistency and chain quality for the underlying blockchain. However, proof-of-work is notoriously indifferent for its waste of energy and the potential of the computing power centralization. To face this issue, proof-of-stake (PoS) is proposed to replace proof-of-work. In Pass and Shi's *sleepy consensus* article (eprint 2016/918) [1], a PoS protocol is constructed to realize consensus on the linearly ordered log abstraction – often referred to as state machine replication or linearizability in the distributed systems literature. This scheme is named as sleepy consensus protocol, which respects two important resiliency properties, i.e., consistency and liveness. Moreover, in sleepy consensus model, players can be either online (alert) or offline (asleep), and their online status may change at any point during the protocol execution.

Algorithm 1 presents how sleepy consensus protocol works. The protocol takes a parameter $p$ as input, where $p$ denotes the probability each node is elected leader in a single time step. All nodes that just spawned will invoke the init entry point. During initialization, a node generates a signature key pair and registers the public key with the public-key infrastructure $F_{CA}$.

**Algorithm 1** Sleepy Consensus Protocol

**If On Initialization:**
Let $(pk, sk) := \sum.\text{gen}()$
Register $pk$ with $F_{CA}$
Let $chain := genesis$
**If On Received *chain'*:**
Assert $|chain'| > |chain|$ and *chain'* is valid w.r.t. eligible and the current time $t$
$chain := chain'$ and gossip *chain*
**Every Time Step:**
Receive input transactions(txs)
Let $t$ be the current time
**if** eligible$^t(P)$ where $P$ is the current node's party identifier **then**
    Let $\sigma := \sum.\text{sign}(sk, chain[1].h, \text{txs}, t), h' := d(chain[1].h, \text{txs}, t, P, \sigma)$
    Let $B := (chain[1].h, \text{txs}, t, P, \sigma, h')$, let $chain := chain||B$ and gossip *chain*
**end if**
Output $extract(chain)$ to $Z$ where $extract()$ is the function outputs an ordered list containing the txs extracted from each block in *chain*
***Subroutine*** **eligible**$^t(P)$**:**
**if** $H(P, t) < D_p$ and $P$ is a valid party of this protocol. **then**
    **return** 1
**else**
    **return** 0
**end if**

Now, the sleepy protocol proceeds very much like a proof-of-work blockchain, except that instead of solving computational puzzles, in this protocol a node can extend the chain at time $t$ iff it is elected leader at time $t$. To extend the chain with a block, a leader of time $t$ simply signs a tuple containing the previous blocks hash, the nodes own party identifier, the current time $t$, as well as a set of transactions to be confirmed. Leader election can be achieved through a public hash function $H$ that is modeled as a random oracle. The difficulty parameter $D_p$ is defined such that the hash outcome is less than $D_p$ with probability $p$. For simplicity, here we describe the scheme with a random oracle $H$ – however as we explain in this section, $H$ can be removed and replaced with a pseurdorandom function and a common reference string.

In this document, we build a simulator for monitoring the real-world performance of sleepy consensus protocol by constructing a framework which implements Algorithm 1, as well as imitating behaviors of honest players and corrupted/adversarial players in the meanwhile. After analyzing the simulating results, **we know ... add text here**

This document is organized as follows. In Section 2, we introduce the framework of simulator. In Section 3, we present how honest players work while simulating. And in Section 4, we imitate the adversarial players' behavior and attack the sleepy consensus protocol by several algorithms. We give the analysis of simulating results in Section 5. Finally, we draw conclusions in Section 6.

## 2 The Framework of Simulator

In this section, we will illustrate the construction of the framework of our simulator. We will give a detailed introduction of how the framework work.

### 2.1 Protocol of the Framework

There are four underlying parts of our framework. They are controller, network, honest nodes and adversary. The algorithm of our protocol has been described in the last section. In this section, I will give a more detailed introdution of our protocol and framework.

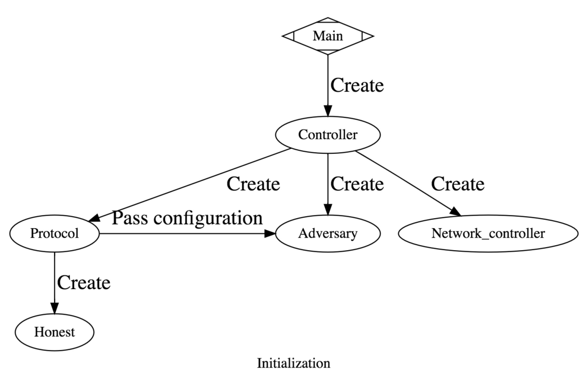
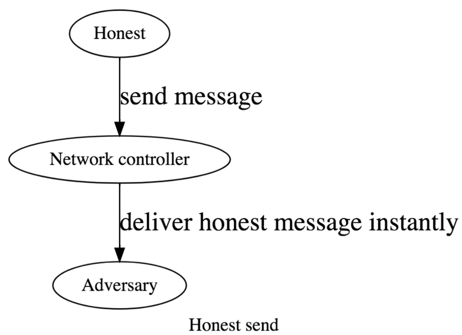



**Fig. 1.** This figure represents the initialization of the whole system.

The network part defines some basic behaviors of the system including sending message and receiving message. When there are some messaging sending demands, the network will receive the messages from the nodes . And after some judgments and operating, the messages will be sent to the destination nodes from the framwork. In the network, there are a buffer storaging the messages needed to be sent to each nodes including honest nodes and corrupted nodes and a buffer storaging the messages sent to the adversary.

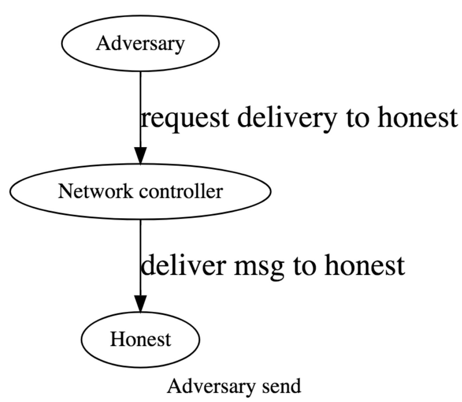
Each messages sent by the honest nodes will be intercepted by the adversary. The honest nodes send messages to the network controller, and the network controller storage will send the these messages to the adversary at once.




send.jpg

**Fig. 2.** This figure represents how the system works if there is a honest node's message demand.

Meanwhile, the adversary make the messages-sending demand and send them to the network controller. The network controller will package the messages such as include the destination of one message. And the network will send these messages to the honest nodes.




send.jpg

**Fig. 3.** This figure represents how the system works if there is a adversary message demand.

The controller part is just like the center of the system. It defines the behavior of the honest nodes, network and adversary. It will determine how to run the hole system in each round and when to stop the running. In each round, the controller will check if the block chain is consistent. If the last $T + 1$ blocks are generated by the corrupted nodes, it means that the adversary break the consistency of the chain and the controller will tell the hole system the chain is inconsistency, and then the system will stop to run and output the chain.

If the block chain is consistent in a round, the controller will run each honest nodes and use the adversary to run the corrupted nodes. The detailed behavior is determined by the honest nodes and the adversary, and will be explained in Section 3 and Section 4.

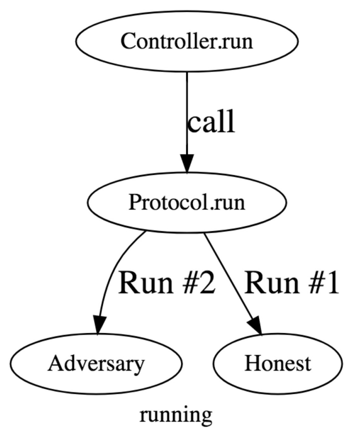



**Fig. 4.** This figure represents how the controller runs the system.

The controller also determine which node is the leader. To choose the leader, it use a hash function $H$. And it will compare $H$ to $D_p$. If $H$ of a node is smaller than the $D_p$, this node will build a block.

## 3 The Imitation of Honest Players

In this section, we will introduce the imitation of honest players using sleepy consensus protocol.

Sleepy consensus protocol is a consensus protocol that allow honest nodes to be either online or offline. Every node can choose to sleep or not at every round the protocol runs. This protocol will keep the consistency of the whole distributed system as long as the majority of online nodes are honest. Compared with other consensus protocol, the sleepy consensus protocol remain resilience under sporadic participation, which is a more realistic scenario in practice.

Every honest nodes follows the sleepy consensus protocol will choose to sleep or not at every round. If they are online, they will send and receive message from the framework. Since it's a distributed system, they will no be aware of whether the message has been successfully sent to all other honest nodes, but the framework will ensure them that the message will be sent to everyone in $\delta$ rounds, which is the internet transmission delay time.

### 3.1 Algorithm for Honest Players

The specific description of sleepy consensus has been shown in the introduction section. The following is a brief restatement of the protocol. First, the nodes will elect a leader using the hash function of identity and current time. If

$$H(identity, currenttime) < D$$

where D denotes difficulty, then Node[identity] will be elected to be a leader. Second, the leader can sign the block using the hash value of previous block, transactions and time, then broadcasts.

$$Block = sign(sk, block, Trans, time)$$

When one honest node receive a new chain, if the time in block is strictly increasing and the time in the blocks is not in the future, it will update its chain with the new chain.

### 3.2 The process to simulate sleepy consensus

There are several steps to simulate the algorithm of sleepy consensus in our program.

First of all, in every round, the controller will let every node to run.Second, the honest node will ask network controller for messages.Third, the network controller will send related message to every node.Then every honest node will ask the controller whether it has been elected a leader.Next, if one honest node is not elected to be a leader, it will do nothing. However, if it is a leader, it will sign one new block using its secret key with the hash value of previous block, transactions and time stamp. Then it will broadcast it. Finally, whatever one honest node has done, it will give some feedback to controller and network controller.

## 4 The Imitation of Adversarial Players

The previous section introduces the implementation of the honest players' behavior under the sleepy consensus protocol in simulator. This section, in contrast, will present the imitatation of the adversarial players' behavior, which aims to hinder the normal functioning of sleepy consensus protocol under current framework structure. And we design four different attacking algorithms to try to break the consensus between different nodes, i.e., players in the distributed system. What should be noticed is that the adversaries cannot betray the rules

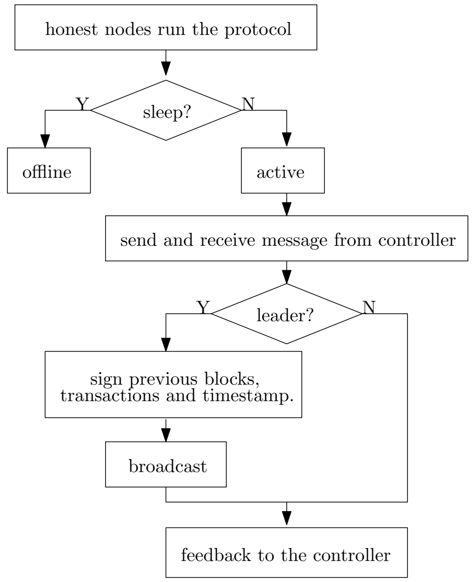



**Fig. 5.** This figure represents how the honest nodes perform in this simulator using sleepy consensus protocol.

established by the framework, while they can only control the network message transportation, i.e., intercepting and delaying the message from honest players. Also, an adversary can manipulate several corrupted nodes in the system, by which adversary impose damage to the system under sleepy consensus protocol.

To simulate the attacks, we assume there is an an adversary lurking within the framework, who owns competence to intercept all messages coming from honest nodes and decide which to delay in the transportation. Also, the adversary is able to access useful information from these message to fork blocks right behind the private chain he captures. Based on above setting, we design four attacking methods for adversary to smash the consensus holded by system, which are illustrated in following subsections.

### 4.1 Naïve Adversary Attack

The *Naïve Adversary Attack* corresponds to an attacking method that is quite simple and easy to be came up with when knowing how the sleepy consensus protocol works. Figure 6 illustrates how naïve adversaries to break the consistency of sleepy consensus. In this figure, when honest players add blocks to the main chain which they think is longest (that is the reason why there are several folks in Figure 6 – blocks linked by dashed arrows), adversaries mine their own private chain and check whether the length of longest added blocks is larger than security parameter $T$ in every time step. If the longest chain added by honest nodes has length larger than $T$ yet smaller than the length of private chain mined by adversaries, the attacker can, just like the red arrow in Figure

6, add their private chain to a specific block in public chain and thus damaging the consistency of sleepy consensus, since some blocks which are confirmed be fixed in main chain are replaced by blocks forged by adversaries.

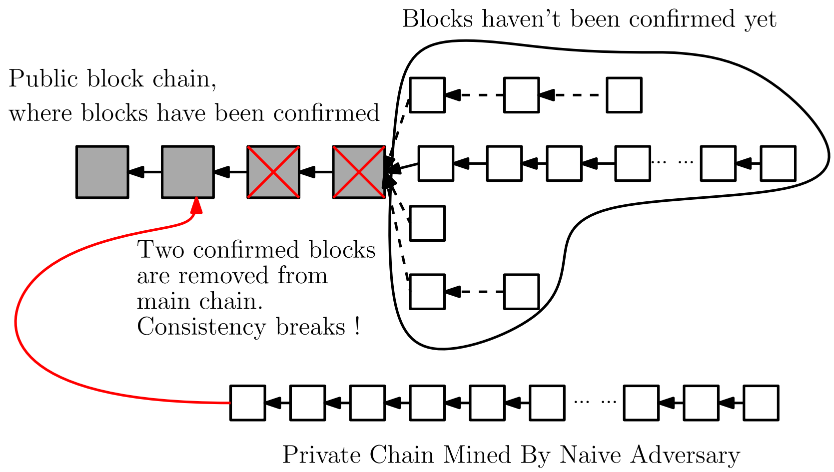
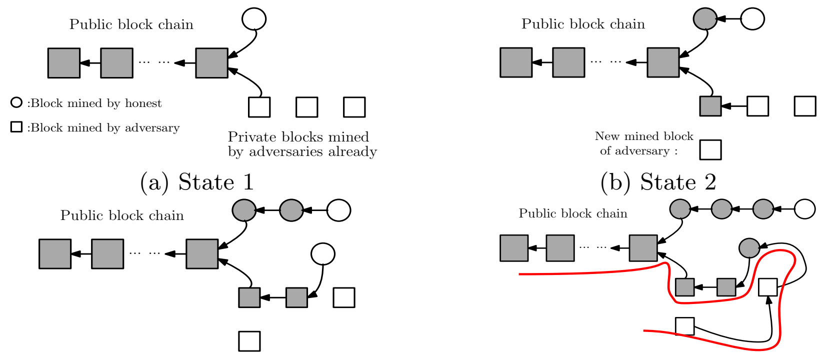



**Fig. 6.** This figure represents the attack which naïve adversaries impose to the sleepy consensus. The consistency of consensus will break when the length of private chain mined by adversaries is larger than the length of the longest private chain mined by honest nodes, and the later one is supposed be larger than security parameter $T$, which denotes the time round number a block can be confirmed as secure, i.e., be eternally fixed in main block chain.

### 4.2 Selfish Adversary Attack

The selfish adversary attack is integrated with the concept of *selfish mining*, which is proposed by Eyal et al. [2]. To illustrate, if everyone adheres to the sleepy consensus protocol, then a node with computation power of 10% should also get 10% profit from mining for expectation. However, the aim of selfish mining is to waste honest nodes' computation power and increase its revenue for mining blocks.

The operation of this attack are quite simple. Firstly, adversaries will mine a private chain and hide it from the rest. If the honest node mines a block from the public main chain, then adversaries will publish their private block immediately and delibrately creating a fork in the network. Otherwise, adversary will keep mining on their private chain. To be more specific, if currently the private chain has led the public chain by length of two, then if a new public block is discovered, adversaries would likely to publish all of their two private blocks, thus making their chain the longest chain in the network. If the private chain has led the public chain by length more than two, then for every newly discovered public block, adversaries will publish one of their private blocks to create forks in the network. We can imagine that through making forks in the network, computation is dispersed and also wasted. Thus, the adversary can gain more than he deserves.




(a) State 1 (b) State 2

(c) State 3 (d) State 4

**Fig. 7.** This figure shows an example to illustrate how selfish adversary algorithm works. Four subfigures respectively correspond to four different states in one selfish attack.

Figure 7 presents an instance of selfish adversary attack. First of all, in Figure 7(a), the system is initialized with one public block chain, and both honest and adversary players want to add block to the public block chain. In state one, adversaries have already mined three blocks. And according to selfish attack algorithm, they will make a folk chain with same length of longest chain in current state, when one honest player adds his mined block to public chain. Figure 7(b) shows the state 2 when new honest player adds block to the public chain branch he selects. And adversaries need to release another private block he holds to keep the chain he works on has the same length of the longest chain in system. What should be noticed is in state 2, adversaries mined a new private block. In Figure 7(c), i.e., state 3, two honest players come in and add blocks. However, in this state, one player select the branch where adversaries work on as the longest chain while another does not, which causes two longest branches are still with same length. Therefore, in state 3, adversaries just keep mining while add no block to the main chain. In Figure 7(d), i.e., state 4, one honest node comes in and chooses the longest chain he prefers. And adversaries still need to add block to the chain they works on to keep the equivalence of length. However, in this case, adversaries only have one block left. Thus in line with algorithm, they need to release that block and add it to their branch to make sure their chain is now the sole longest chain in system – just as red line in Figure 7(d) represents. And from then on, both honest nodes and adverseries will select the chain underlined in red color as public chain to keep mining.

### 4.3 Stubborn Adversary Attack

The stubborn adversary attack, which is proposed by Kartik et al. [3], is developed from *selfish adversary attack*. The *selfish mining* strategy withholds blocks when it is ”in the lead” but cooperates with the honest network when it falls behind. However the new *stubborn mining* strategies is that *the attacker should not give up so easily!* In this case the attacker can increase profits by mining on its private chain more often, even under circumstances where a selfish-mining attacker would acquiesce to the public chain.

We integrate *Equal Fork stubborn* and *Trail stubborn* mining strategies in our simulator. Both attacks are similar to *selfish adversary attack* and modify a certain part of it. For *Equal Fork stubborn*, adversaries will mine a private chain and hide it from the rest. If the honest node mines a block from the public main chain, the F-stubborn miner would conceal her new block and continue mining on it privately instead of hurry to reveal her new block to the public. In this way the adversaries can make the honest waste more than one block than *selfish adversaries*.

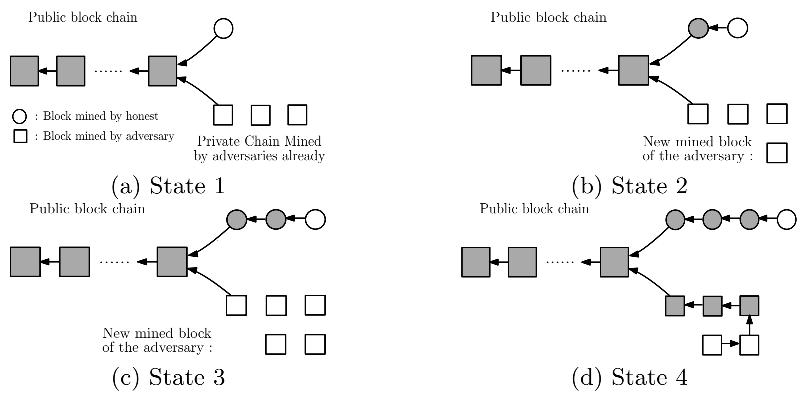
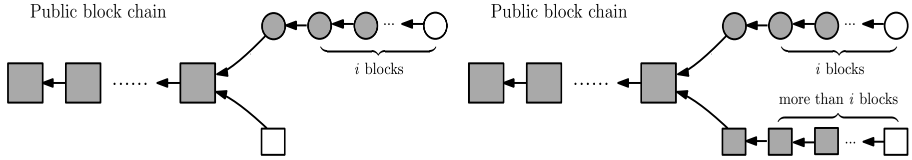



**Fig. 8.** This figure gives an example to illustrate how equal fork stubborn adversary algorithm works. Four subfigures respectively correspond to four different states in one equal fork stubborn attack.

Figure 8 presents an instance of *Equal Fork stubborn* attack. The initial state is the same as that in *selfish attack*, in which the adversaries have mined three blocks. When the honest node mines a block from the public chain, adversaries just keep mining private blocks. Eventually adversaries can publish all their private blocks and becomes the longest chain, discarding the blocks mined by honest nodes.

*Trail stubborn* can be considered as a generalization of *selfish mining*. In *selfish attack* adversaries will give up mining once the private chain falls behind the public chain. However for *Trail stubborn* adversaries continue mining on it, in the hope of catching up. The private chain can lag behind the public by $j$ blocks before giving up, where in *selfish attack* $j = 1$.

Figure 9 explains how *Trail stubborn* works. Initially the private chain lags behind the public by $i(i < j)$ blocks. But adversaries will keep mining and once they exceed the public chain they will finally publish the private chain and damage the consistency.




(a) State 1 (b) State 2

**Fig. 9.** This figure gives an example to illustrate how trail stubborn adversary algorithm works. Two subfigures respectively correspond to two different states in one trail stubborn attack.

### 4.4 Selfish Eclipse Attack

We use terminology *eclipse attack* to describe network attacks where the adversary essentially partitions the honest miners into the public and the eclipsed as shown in Figure 10. The adversary controls the communication between the public and the eclipsed.

There are two ways for the adversary to attack the eclipsed. The adversary can choose to *destroy the eclipsed victim*, which means that the adversary ignores all of the blocks mined by the eclipsed. This has the effect of destroying the computation power of the eclipsed and therefore increases the adversary's effective gain. The can also *collude with the eclipsed victim* and force her to cooperate. Both the adversary and the eclipsed would maintain a single private blockchain and the adversary would accept all the blocks mined by the eclipsed.

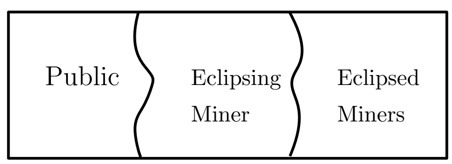



**Fig. 10.** This figure illustrates that selfish eclipse attack can separate the public and the eclipsed to *destroy the eclipsed victim* or *collude with the eclipsed victim*

.

## 5 The Analysis of Simulating Results

Here is the analysis of Simulating Results. In order to find the influence of different parameters on our system, we try some tests on the different parameters in our protocol such as the number of adversary nodes, maximum network delay and difficulty. There are some results of the tests.

First of all, the adversary's success increases as the number of adversary nodes increases. For example, in our protocol, the total number of nodes is one hundred, and difficulty is 0.1, maximum network delay is 1 round, and $T = 10$. If the number of adversary nodes is more than fifty, then it will break consistency with high probability. Otherwise, it can not break consistency within some rounds with high probability.

## 6 Conclusion

To sum up, in this paper, we build a distributed consencus simulator for monitering different consensus protocol's performance. Specifically, we use sleepy consensus protocol as an instance to launch implementation. We introduce both the framework and detailed algorithms for imitaing honest or adversary players' behaviour. Also, we analyze the performance of sleepy consensus protocol that **result ...**

## References

1. Bentov, I., Pass, R., & Shi, E. (2016). The Sleepy Model of Consensus. *IACR Cryptology ePrint Archive*, 2016, 918.
2. Eyal, I., & Sirer, E. G. (2014, March). Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security* (pp. 436-454). Springer, Berlin, Heidelberg.
3. Nayak, K., Kumar, S., Miller, A., & Shi, E. (2016, March). Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In Security and Privacy (EuroS&P), 2016 IEEE European Symposium on (pp. 305-320). IEEE.