Towards True Decentralization: A Blockchain Consensus Protocol Based on Game Theory and Randomness

Naif Alzahrani and Nirupama Bulusu

Portland State University, Portland OR 97207, USA
{nalza2,nbulusu}@pdx.edu

Abstract. One of the fundamental characteristics of blockchain technology is the consensus protocol. Most of the current consensus protocols are PoW (Proof of Work) based, or fixed-validators based. Nevertheless, PoW requires massive computational effort, which results in high energy and computing resources consumption. Alternatively, fixed-validators protocols rely on fixed, static validators responsible for validating all newly proposed blocks, which opens the door for adversaries to launch several attacks on these validators such as DDoS and eclipse attacks. In this paper, we propose a truly decentralized consensus protocol that does not require PoW and randomly employs a different set of different size of validators on each block's proposal. Additionally, our protocol utilizes a game theoretical model to enforce the honest validators' behavior by rewarding honest validators and penalizing dishonest ones. We have analyzed our protocol and shown that it mitigates various attacks that current protocols suffer from.

Keywords: Blockchain · Consensus Protocol · Game Theory · Randomness.

1 Introduction

Over the last few years, blockchain technology has been an attractive solution for many different industries. The reasoning behind this is the transparency, security, quality assurance, global peer-to-peer transactions, and decentralization that blockchain technology provides [17]. Despite its potential to elevate security, as with all new technologies, security risks can be found beneath the hype [19]. Moreover, blockchain technology has introduced new kinds of attacks such as *block withholding* and *selfish mining* attacks. Such attacks occur for various incentives, mostly financial. To defend against such attacks and to strengthen blockchain security, *game theory* stands out as a potentially powerful means.

Fundamentally, a blockchain is a public, distributed ledger that contains chained blocks, each of which is made up of several transactions. These blocks are validated globally and transparently to guarantee security. This validation has to be executed without the need for a central authority. Instead, the blocks are validated, shared and synchronized across nodes via a peer-to-peer, distributed, and decentralized consensus mechanism [15].

One of the fundamental characteristics of blockchain technology is the consensus protocol. In a blockchain, a consensus protocol ensures that all nodes in the blockchain network agree on the validity of a block to be included in the public ledger. It also guarantees that all nodes have the same order of blocks in their blockchains. This is of significance because blockchains are trustless distributed nodes which need a way to synchronize their copies of stored data. The nodes responsible for executing consensus protocols are the validators (or miners in some blockchains). There are a considerable number of existing consensus protocols. Nonetheless, not all of them guarantee the *true decentralization*, in which the blocks' validation is executed by anonymous, variable sets of validators to strengthen the protocol's robustness. Instead, they rely on fixed, known validators selected at the genesis state. This opens the door for various risk threats which will be discussed shortly. Besides, most of the current consensus protocols do not take the number of validators or how to select them into consideration, as will be discussed in Section 2. The number of validators in a blockchain network influences its security and efficiency substantially, especially in a fully decentralized blockchain, in which there are no special nodes, and all nodes are trustless.

In this paper, we mainly address the problem of validators' selection in terms of *how to select them* and *how many to select* to achieve a satisfiable trade-off between security and efficiency. Also, we study the incentives of malicious nodes to deviate from the consensus protocols, and we apply a game theoretical model to enforce honest behavior.

The first and most popular consensus protocol to secure and decentralize blockchains is the **Proof of Work (PoW)**. This protocol requires powerful nodes known as *miners* to validate the blocks. This consensus approach, however, demands massive computational effort from the miners, which ultimately results in high consumption in energy and computing resources. Additionally, the PoW protocol relies on a few mining pools (often just 2 or 3 mining pools), which raises doubt on the decentralization of PoW-based blockchains [7]. Furthermore, such blockchains frequently fork. As a result, the blockchain nodes are not able to rely on a new block as soon as it appears. Alternatively, they must wait until this block is deep enough in the chain, which results in very high latency [7].

An alternative approach that does not require the expensive PoW computation and, therefore, enhances *efficiency* is **Fixed-Validators Decentralization**. In this approach, a small fixed number of nodes are chosen to be validators. This approach ensures the integrity of the blockchain as long as the majority of the validators are honest. The validators are selected at the genesis state, and they are, usually, selected based on the stake they have. However, the efficiency of such protocols is influenced by the number of selected validators. This is because each validator performs some work to check the validity of a block and communicates with each other validator in the committee to reach a consensus. This incurs computation and communication overhead proportional to the committee size. The validators agree on a block to be included in the chain if the block is digitally signed by a majority of them.

Although the fixed-validators approach is efficient, it has several limitations. First, it relies on an extreme trust assumption that the majority of validators are honest; nevertheless, it is possible for a powerful adversary to corrupt or bribe most of them over time [6]. Second, a fixed committee of validators is vulnerable to adversarial attacks, since they are known and fixed. For example, an adversary can launch a DoS attack against the validators, preventing them from validating new blocks or receiving messages from each other. Third, although this approach is efficient, utilizing a relatively small number of validators in a large network with a massive number of transactions or blocks can bottleneck the performance.

The second alternative approach is **True Decentralization**, in which every node in the system can be chosen to be a member of the validators. In such an approach, a set of validators are selected randomly from the set of "all nodes" in each round of validation. In other words, it does not require a single set of validators to execute all rounds. As a result, the true decentralization approach distributes the validation work among all nodes and can withstand the powerful adversaries. Note that the fixed-validators approach is defined on the same group of validators and do not support validators' replaceability. In response to this, we propose a novel, truly decentralized consensus protocol that selects a different set of random validators on every block proposal.

Despite the security provided by the true decentralization, it does not guarantee that the validators are always honest and do not deviate from the protocol. For example, a dishonest validator might perform a block withholding attack (see Section 4) in favor of a malicious proposing node (i.e., the node which creates and proposes the new block). This attack can result in undermining the consensus process. To overcome such vulnerability, we integrate a game theoretical model into our consensus protocol to reward honest validators and to punish dishonest or lazy validators that do not adhere to the protocol. Additionally, the always-validation (i.e., validators always validate even if the risk likelihood of a block's proposer is low) is a performance shortcoming particularly found in blockchains with low hostility. Thus, utilizing a game theoretical model that enables validators to validate with some probability proportional to the proposers' risk likelihoods would significantly enhance the protocol efficiency.

The contribution of this paper is a *new consensus* protocol that deals with the problem of selecting validators and has the following advantages:

1. It achieves the true decentralization by selecting a *different set of validators* on every block proposal at random.
2. The *number of the selected validators* is dynamic and variable. Hence, instead of selecting a fixed static number of validators, our protocol utilizes game theory to select a different number of validators (on every block proposal) proportional to the *risk likelihood* of the proposing node.
3. The game theoretical approach exploited by our protocol accomplishes the following further benefits:
   (a) It enforces the honest validators behavior by rewarding honest validators and penalizing dishonest ones.

(b) It enhances the efficiency by eliminating the *always-validation* mode. Thus, the validators validate with probability proportional to the *risk likelihood* of the proposing node.

2 Related Work

In this section, we will examine some related existing works. The related literature falls into two general camps: (1) current, widely-used consensus protocols, and (2) works that integrate blockchain and game theory.

2.1 Consensus Protocols

In this section, we present only the BFT (Byzantine Fault Tolerance) protocols, since they are more relevant to our proposed protocol.

Tendermint [3, 12] is a consensus BFT protocol that can work even if up to one-third of nodes in the network fail in arbitrary ways. It does not require the PoW mining, which overcomes the energy and resources consumption issues. Instead, it relies on a fixed, static set of validators (i.e., fixed-validators decentralization) selected at the genesis state to validate the new block and vote on them. In Tendermint, a proposer *propose*s a new block, then the validators *pre-vote* on the block and only proceed to *pre-commit* if they receive more than 2/3 of *pre-vote*s. Validators only accept the block if more than 2/3 of *pre-commit*s are received. Tendermint is notable for its simplicity, performance, and fork-accountability [13]. Our protocol is based on Tendermint and inherits all the features offered by Tendermint. However, it deals with the validators' selection issue by selecting a different random set of validators on each block proposal (i.e., true decentralization).

Hyperledger Fabric is a BFT consensus algorithm [9], which can tolerate up to one-third byzantine nodes in a blockchain network. In Fabric v0.6, there exists a fixed number of *validation peers* responsible for executing the consensus protocol. A proposer can submit a transaction to any of them. Then, the chosen peer broadcasts this transaction to the other peers. One of the validation peers is selected as a *leader*. When generating a block, the leader broadcasts it to all peers. When a validation peer receives this block, it hashes it, broadcasts the hash to all other peers, and begins counting their responses. If two-thirds of the responses were received with the same hash, it commits the new block to its local ledger. Hyperledger Fabric, like Tendermint, employs a fixed known number of validation peers.

In our previous work [2], we presented a protocol based on Tendermint and, hence, can tolerate up to one-third of Byzantine faulty nodes. This protocol overcomes the fixed set of validators that Tendermint suffers from and utilizes the randomness to select a different set of *log* n validators each time a new block is proposed (where n is the number of nodes in the network). This protocol outperformed Tendermint and achieved a remarkable performance with a satisfactory level of security. Nevertheless, this protocol is vulnerable to attacks such as block withholding. This paper aims to overcome the limitations presented in our previous work.

2.2 Blockchain and Game Theory

Although blockchain technology has gained considerable attention from the computer science and economics communities, the use of game theory methods in this technology is limited [16]. In this section, we present the most relevant and recent works that utilize game theory in blockchain technology.

Xu et al. [20] proposed a game theoretical approach to suppress the attack motivation on a blockchain that consists of mobile devices and edge servers. The game is formulated between a mobile device and an edge server, where the mobile device can send a request to the server to acquire a real-time service or launch an attack. On the other hand, the server chooses to either provide the service or to attack the mobile device. The authors introduced a punishment mechanism according to the action record to mitigate the attacks on the blockchain. They have concluded that both players tend to behave finely when the punishment weight is significant. The proposed approach was designed to deal with attacks like zero-day attack, DDOS attacks, and password-based attacks.

Johnson et al. [10] employed a game theoretical model to analyze the incentives for a mining pool to launch a DDoS attack against another mining pool. The players in the game are two competing mining pools, where each one may utilize additional computing resources to increase the chance of winning the mining race, or to trigger a DDoS attack to lower the expected success of the other competing mining pool.

Luu et al. [14] studied the block withholding attack on mining pools using a game theoretical approach by formulating the Bitcoin mining as a game. They analyzed the block withholding attack and concluded that the attack is profitable and well-incentivized in the long-term. The authors derived the game equilibrium state, which is a mixed strategy where all clients are incentivized to attack rather than participate honestly to maximize their payoffs. Finally, the authors concluded that the PoW protocol is vulnerable to such an attack.

In a paper entitled 'The Miner's Dilemma,' Eyal [5] studies the scenario when pools attack each other. Open pools (i.e., pools of miners that allow any miner to join the mining work) are vulnerable to block withholding attacks performed by infiltrated miners from competing pools. This paper defines a game where pools recruit some of their participants to infiltrate other pools to diminish their mining capabilities. This game is called the miners dilemma where players are two pools, and their strategies are whether or not to attack each other. The author observes that attacking is the dominant strategy for each player.

All the above works have introduced game theoretical approaches to the PoW mining protocol. As previously discussed in Section 1, PoW is not an attractive approach for blockchains that are efficient-sensitive due to its massive computation demands. In a more relevant work presented by Kiayias et al. [11], Ouroboros consensus protocol was proposed. Similar to our protocol, Ouroboros eliminates the need for an energy-hungry PoW protocol. Ouroboros is based on the Proof of Stake (PoS) protocol. It works by dividing the time into rounds called slots in which each slot is assigned to a leader. The leaders are picked based on the stake they have. A chosen leader is responsible for producing a block for its time slot. The authors utilized game theory to introduce a reward mechanism to incentivize the participants in the blockchain. By means of the game theoretical design, attacks such as selfish-mining and block withholding are mitigated. The rewarding mechanism works by awarding a positive payoff for participants who do not diverge from the protocol.

3 The Proposed Consensus Protocol

In this paper, we propose a new consensus protocol that exploits randomness and game theory to achieve true decentralization security with respect to efficiency. Our protocol is based on Tendermint and exploits its capability to overcome up to one-third of Byzantine faults. Unlike other protocols that rely on a fixed, static set of validators responsible for validating all proposed blocks, our protocol randomly selects a different set of different size of validators each time a new block is proposed. Thus, it improves the security, since the validators are not known before proposing the new block. Further, the number of validators employed in the consensus process is also unknown. These two factors make the job more difficult for an adversary to attack or bribe the set of validators. In respect to efficiency, our protocol distributes the validation work among nodes by selecting different sets of validators for different blocks instead of relying on the same static fixed set of validators for all proposed blocks. This is of significant concern, especially in a blockchain with a small number of validators and a massive number of transactions, or blocks. Additionally, the efficiency is enhanced, as not all selected validators upon proposing a new block will validate that block. Instead, a validator validates with a probability based on the outcomes of a game played between the proposing node and this validator. This saves a substantial computational cost, particular, in a low hostility blockchain environment.

Each node in the blockchain has a unique pair of keys (public pk and secret sk) and is identified by its public key. Moreover, each node has a public trust (reputation) value R where this value affect the selection of a node to be validator over time. There are four types of nodes in our protocol:

1. **Proposing (proposer):** This is the node which creates, proposes, and broadcasts to the network the new block.
2. **Validation-leader:** This is the node responsible for selecting the random set of validators for the proposing node.
3. **Validator:** This node is responsible for validating the newly proposed block. Moreover, validators communicate their votes on the block to reach consensus.
4. **Idle:** This node does nothing except wait for the decision to be made by validators on whether to accept or reject the block. All other nodes in the network are idle.

Our protocol works in two phases: the initialization phase, and the verification (validation) phase. The blockchain initiator executes the first phase at the genesis state, in which it randomly maps each proposer to its validation-leaders. In the second phase, each node becomes a proposer in a round-robin fashion. When a node is a proposer, it proposes a block, broadcasts it to all nodes, and its corresponding validation-leaders randomly select the validators to verify (validate) this block. In this phase, a two-stage attacker-defender game is proposed, where the proposer is the potential attacker (i.e., player x) in both stages. The defenders (i.e, player y) in the first-stage are the validation-leaders. The defenders in the second-stage are the validators (i.e., player z) that have been selected by the validation-leaders from the first-stage. Next two subsections present an in-depth description of how these two phases are executed.

3.1 Initialization Phase

This phase's main task is *mapping proposers to validation-leaders*. At the genesis state (i.e., when the genesis block is proposed), the blockchain initiator randomly maps four validation-leaders to each proposer in the network. The reasoning behind this choice is that four is the minimum number to provide tolerance to a single Byzantine fault [3]. This is because our protocol is based on Tendermint, and it is assumed that a Tendermint network has two-thirds of non-Byzantine nodes. A simple approach is to employ only one validation-leader per a proposer, however, to ensure safety and liveness of the consensus process, we need to utilize more. It is worth noting that this number (i.e., four) can be changed based on factors like the network's size and hostility, or the blockchain application that utilizes our protocol. Our approach works with any number of validation-leaders per proposer other than four, but we utilizes the minimum in favor of efficiency. Additionally, this number can be a random number to further increase robustness.

The mapping is executed randomly according to the nodes weights (reputations R). As shown in Algorithm 1, we use the Weighted Random Sampling (WRS) algorithm [4]. The weights in our algorithm are the nodes' reputation values. Furthermore, this mapping is done blindly; that is, no proposer knows its corresponding validation-leaders and no validation-leader knows its proposer until executing the consensus protocol. This way, we prevent a malicious proposer from corrupting or bribing its validation-leaders and vise versa.

To accomplish the anonymous mapping, the blockchain initiator, first, includes a secret S_1 in every node's genesis block, so it uses this secret when the node becomes a proposer. S_1 is a hash that includes the proposer's public key $pr.pk$, all the four selected the validation-leaders' public keys $[vl_1.pk - vl_4.pk]$, the blockchain ID $blockchainID$, and a random number $Rand_1$ as flows:

$$S_1 \leftarrow hash(pr.pk||vl_1.pk||vl_2.pk||vl_3.pk||vl_4.pk||blockcahinID||Rand_1)$$

Note that there is only one proposer secret S_1. Each proposer in the network has its own S_1. This secret is checked by each of the four validation-leaders.

Algorithm 1: Proposers to leaders Mapping

```
Input : A population $A$ of $n$ nodes having reputation values
foreach $pr \in A$ do
    for $k \leftarrow 1$ to $4$ do
        Try:
        $p_i(k) \leftarrow \frac{R_i}{\Sigma_{s_j \in A-B} R_j}$
        Randomly select $vl_i$ with probability $p_i(k)$ from $A - B$
        if $C_i.size > 4$ then
            Go to Try
        else
            $B.add(vl_i.pk)$
        end
    end
    Randomly generate $Rand_1$
    $S_1 \leftarrow hash(pr.pk||vl_1.pk||vl_2.pk||vl_3.pk||vl_4.pk||blockcahinID||Rand_1)$
    Append $S_1$ to the $pr$'s genesis block
    foreach $vl_i \in B$ do
        Randomly generate $Rand_2$
        $S_2 \leftarrow hash(S_1||Rand_2)$
        $\pi \leftarrow Sign_{in.sk}(pr.pk||vl_i.pk||blockcahinID)$
        $g \leftarrow [((i-1).\frac{n}{4}) + 1 \ , \ i.\frac{n}{4}]$
        $C_i.add(S_2||Rand_2||\pi||g)$
    end
    Flush $B$
end
```

Second, blockchain initiator generates a validation-leader's secret S_2. S_2 is a hash that includes the proposer's secret S_1, and a random number $Rand_2$ as flows:

$$S_2 \leftarrow hash(S_1||Rand_2)$$

Here, we use different $Rand_2$ for each validation-leader to make S_2 different for each one of them. Note that $Rand_2$ is private and is only known to its particular validation-leader node.

To ensure that a validation-leader is *legitimate*, and that it has been elected by the blockchain initiator, we need to utilize a verifiable proof π. This proof is a digital signature signed by the initiator using its private key $in.sk$. The proof π includes the proposer's public key $pr.pk$, the validation-leader's public key $vl.pk$, and the blockchain ID $blockchainID$ as below.

$$\pi \leftarrow Sign_{in.sk}(pr.pk||vl_i.pk||blockcahinID)$$

The validation-leader must submit this proof to its elected validators so that each can verify π using the initiator's public key $in.pk$ prior to involving in the validation and consensus process. This protects against malicious nodes claiming that they are validation-leaders for a proposer.

As mentioned, for one proposer, there exists four leaders responsible for selecting the validators for the block proposed by this particular proposer. This arises a new problem of selection conflict, since each validation-leader selects the validators blindly without knowing its peer leaders. Consequently, the four leaders perform the validators' selection from the same pool of nodes without any communication or agreement between them. This can result in selecting a validator more than once by different leaders. Our protocol overcomes this problem by dividing the pool of nodes into four pools, each of which is assigned to a leader. Specifically, each validation-leader will have a range g to choose from determined at the genesis state. Note, we assume that all the nodes in the network have the same set of nodes in the same order. As shown in Algorithm 1, g is predetermined by the blockchain initiator and is defined as below:

$$g \leftarrow [((i-1).\tfrac{n}{4}) + 1 \ , \ i.\tfrac{n}{4}]$$

Where $1 \leq i \leq 4$ and is the index of a validation-leader among its peers.

In Algorithm 1, there are three lists. The first one (A) is a population of n nodes each of which has a reputation value R. The second list (B) is a temporary list for a proposer to hold the public keys for the selected validation-leaders. This list is flushed after selecting the validation-leaders and initializing their secrets and proofs. The last list (C) is for a validation-leader. There exists four corresponding proposers for each validation-leader. Thus, C stores four tuples, and each of them corresponds to one proposer. Each tuple includes the secret S_2, the random number $Rand_2$, the proof π, and the range g. By the end of executing Algorithm 1, each node in the network will have exactly one proposer's secret S_1 used when the node becomes a proposing node, and a list C used whenever this node becomes a validation-leader for one of its four proposing nodes.

3.2 Verification (Validation) Phase

This phase is executed upon proposing a new block. It is carried out by three parties (proposer, validation-leaders, and validators). The main purpose of this phase to decide the validity of the newly proposed block and to reach a consensus on this decision.

When a node becomes a proposer, it broadcasts its secret S_1 to all nodes in the network. Every other node checks if it is a validation-leader for this proposer by looping through its list C and hashing the received S_1 and each private random number $Rand_2$ it has. If the resulting hash matches its secret S_2, then this node is a validation-leader for this proposer as shown in Algorithm 2.

After a node decides that it is a leader for the proposer, this leader plays the first-stage game with the proposer to decide how many validators (m) to select.

First-Stage Game This game takes place between the proposer (i.e., player x) and each of its validation-leaders (i.e., player y). The validation-leader determines the number of validators based on the outcome of the game. There are two strategies for the validation-leader from which to choose. The first one

Algorithm 2: Validation-leader checking

```
Input  : The node's list $C$, and the received proposing node's secret $S_1$
Output: A decision of weather or not this node is a validation-leader
decision ← false
foreach $tuple_i \in C$ do
    if $S_2^i = hash(S_1^i || Rand_2^i)$ then
        decision ← True
    end
end
Return decision
```

is to *UseMinimumValidators* where the minimum is four validators. The second strategy is to *AddMoreValidators* where the number of validators varies based on the outcome of the game, which is proportional to the risk likelihood of the proposer. The strategy profile for the second player (i.e., the proposer) is a) $Cheat$, and b) $NotCheat$. A proposer could be of two types: *malicious* or *regular*. Our game is considered to be a one-to-four game where each of the four leaders has no cooperation with the other leaders, so, we consider each game between a leader and the proposer as an independent event. Since the validation-leader do not know the type of player x (i.e., regular or malicious), we model our game as a Bayesian game. This is because the leader node (player y) in our model has incomplete information about the game. Player x, however, has this private information about its type known only to it.

Strategic Form of First-Stage Bayesian Game First, we model our game as a strategic form as shown in table 2 and 3. Table 1 shows the notation used in our game theoretical approach. It is worth mentioning the importance of the proposer (β), and how it is obtained is not discussed in this paper due to space limitation. Table 2 shows the payoff matrix of the game when player x is of type *malicious*. For each cell in the payoff matrix, the first payoff is for player x and the second one is for player y. Table 3 shows the payoff matrix of the game when player x is of type *regular*. The goal of both players x and y is to maximize their payoffs. We assume that the players are rational.

Extensive Form of First-Stage Bayesian game The Bayesian game introduces a third player called Nature (denoted by N), which determines the type of player x by assigning a probability (μ) to player x of being *malicious*. Figure 1 represents the the Bayesian game extensive form. μ can be assigned according to the environment of the network, which can be learned dynamically by multi-stage games. A higher value of μ is given when the environment is hostile.

Bayesian Nash Equilibrium (BNE) Analysis

A. Game Pure-Strategy BNE: In this section, we analyze BNE assuming that player x knows player y's belief of μ. If player x plays his pure strategy

Table 1. The first-stage game notation.

Symbol	**Definition**
β	Importance of the proposer. We assume that some proposing nodes in the blockchain network have higher criticality than others.
γ	A reward that player y can get if it maintains the performance of the consensus process under a cretin threshold by playing $UseMinimumValidators$. However, player y can loose γ (i.e. deducted from his gain g_y) if it plays $AddMoreValidators$ and the performance violates the specified threshold. We assume that player y will not win γ in case of a successful attack (i.e. player x plays $Cheat$ and player y plays $UseMinimumValidators$).
w_x	Work done by the proposing node (player x) to play $Cheat$.
g_x	The gain for player x from a successful attack.
c_x	The cost (risk) for player x if captured.
w_y	Work done by the validation-leader (player y) to play $AddMoreValidators$.
g_y	The gain for player y from capturing a cheater, in case the validation-leader employed more validators.
c_y	The cost (risk) for player y if fails to capture a cheater.
μ	The probability of player x being malicious.
N	The nature node, which determines the type of player x.

Table 2. Strategic form of the first-stage Bayesian game (Player x is malicious)

Game Matrix		**player *y* (Validation-leader)**	
		AddMoreValidators	*UseMinimumValidators*
player *x*	$Cheat$	$(\beta.c_x) - w_x$, $[(\beta.g_y) - \gamma] - w_y$	$(\beta.g_x) - w_x$, $\beta.c_y$
	$NotCheat$	0, $-w_y - \gamma$	0, γ

Table 3. Strategic form of of the first-stage Bayesian game (Player x is regular)

Game Matrix		**player *y* (Validation-leader)**	
		$AddMoreValidators$	$UseMinimumValidators$
player *x*	$NotCheat$	0, $-w_y - \gamma$	0, γ

($Cheat$ if malicious, $NotCheat$ if regular), then, the expected payoff of player y playing his pure strategy $AddMoreValidators$ is:

$$E\mu_y(AddMoreValidators) = \{\mu.[((\beta.g_y) - \gamma) - w_y]\} + \{(1-\mu).(-w_y - \gamma)\}$$

Similarly, the expected payoff of player y playing his pure strategy *UseMinimumValidators* is:

$$E\mu_y(UseMinimumValidators) = [\mu.(\beta.c_y)] + [(1-\mu).\gamma]$$

So, if $E\mu_y$ $(AddMoreValidators) > E\mu_y$ $(UseMinimumValidators)$ Or,

$$\{\mu.[((\beta.g_y) - \gamma) - w_y]\} + \{(1-\mu).(-w_y - \gamma)\} > [\mu.(\beta.c_y)] + [(1-\mu).\gamma]$$

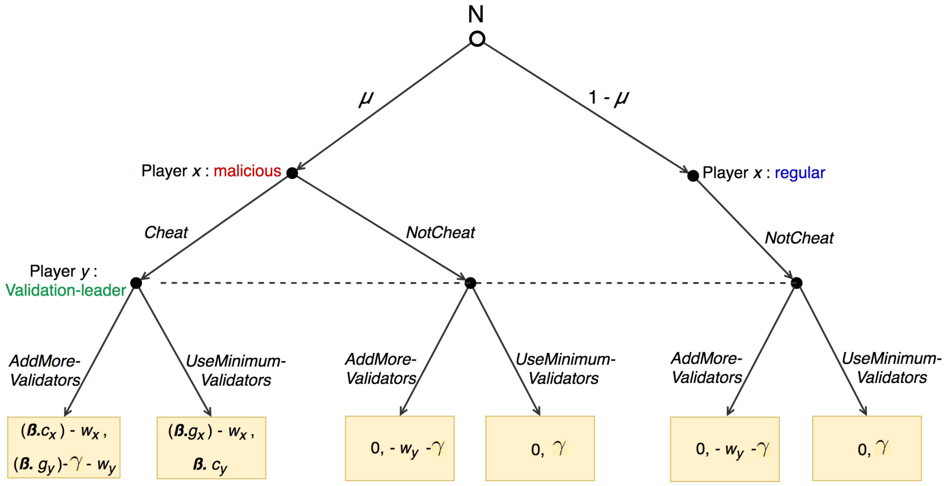

Fig. 1. Extensive form of first-stage Bayesian game.

Which can be simplified to:

$$\mu > \frac{w_y + 2\gamma}{\beta(g_y - c_y) + \gamma} \tag{1}$$

Then, the best response of player y is to play $AddMoreValidators$. Nevertheless, if player y chooses to play $AddMoreValidators$, $Cheat$ will no longer is the best response for player x type *malicious* and, instead, will choose to play $NotCheat$. As a result, (($Cheat$ if malicious, $NotCheat$ if regular), *AddMoreValidators*, μ) is not a Bayesian Nash Equilibrium (**BNE**). However, if $E\mu_y(AddMoreValidators) < E\mu_y$ $(UseMinimumValidators)$ Or,

$$\mu < \frac{w_y + 2\gamma}{\beta(g_y - c_y) + \gamma} \tag{2}$$

Then, the best response for player y is to play $UseMinimumValidators$ and thus (($Cheat$ if malicious, $NotCheat$ if regular), $UseMinimumValidators$, μ) is a *pure-strategy* **BNE**.

If player x type *malicious* chooses to play the pure strategy $NotCheat$, player y's dominant strategy is $UseMinimumValidators$, regardless of μ. Nevertheless, if player y plays $UseMinimumValidators$, the best response for player x type *Malicious* is $Cheat$, which reduces to the above case. Hence, (($NotCheat$ if *Malicious*, $NotCheat$ if *Regular*), $UseMinimumValidators$) is not a **BNE**.

B. Game Mixed-Strategy BNE: We previously showed that when equation 1 is true, there is no *pure-strategy* **BNE** exists. So, we have to find *mixed-strategy* **BNE**. Let p be the probability with that player x plays $Cheat$. Let q be the probability with player y plays $AddMoreValidators$. The expected payoff of player y playing $AddMoreValidators$ is:

$$E\mu_y(AddMoreValidators) = \{p.\mu.[((\beta.g_y) - \gamma) - w_y]\} + \{(1-p).\mu.(-w_y - \gamma)\} \\ +\{(1-\mu).(-w_y - \gamma)\}$$

The expected payoff of y playing $UseMinimumValidators$ is.

$$E\mu_y(UseMinimumValidators) = \{p.\mu.(\beta.c_y)\} + \{(1-p).\mu.\gamma\} + \{(1-\mu).\gamma\}$$

So, player y plays $AddMoreValidators$, if $E\mu_y(AddMoreValidators) >$ $E\mu_y(UseMinimumValidators)$. Or,

$$p > \frac{w_y + 2\gamma}{\mu\beta(g_y - c_y) + \mu\gamma} \qquad (3)$$

Likewise, we calculate the expected payoffs player x. The expected payoff of x playing $Cheat$ is:

$$E\mu_x(Cheat) = \{q.\mu.[(\beta.c_x) - w_x]\} + \{(1-q).\mu.[(\beta.g_x) - w_x]\}$$

The expected payoff of x playing $NotCheat$ is:

$$E\mu_x(NotCheat) = 0$$

As a result, player x plays $Cheat$, if $E\mu_x(Cheat) > E\mu_x(NotCheat)$, or:

$$q > \frac{w_x - (\beta g_x)}{\mu\beta(g_y - c_y)} \qquad (4)$$

Now, we derive our game's mixed-strategy **BNE** as: ((q if malicious, $NotCheat$ if regular), p, μ).

Thus far, we have obtained the above game's mixed-strategy **BNE**. However, this game is molded for one player x and one player y, and we have four defenders (validation-leaders) and player x knows this fact. Hence, ((q if malicious, $NotCheat$ if regular), p, μ) is no longer a valid *mixed-strategy* **BNE**. Thus, we calculate a new *mixed-strategy* **BNE**. The events of validations are independent. We have four validation-leaders. Therefore, the likelihood that the four validators plays $AddMoreValidators$ is $\hat{p}$ and is calculated as:

$$\hat{p} = (4.p) - p^4 \qquad (5)$$

Where, p is the probability that one validation-leader plays *AddMoreValidators*. Now, the attacker plays $Cheat$ with probability $\hat{q}$ defined as:

$$\hat{q} = q - (\hat{p} - p) \qquad (6)$$

So, our new *mixed-strategy* **BNE** is: (($\hat{q}$ if malicious, $NotCheat$ if regular), p, μ).

Deciding the Number of Validators (M) After executing the first-stage game, each validation-leader decides its number of validators m, of which $m < n$ where n is the total number of nodes in the network. The m value can be: a) four validators if the validation-leader chooses to play $UseMinimumVlaidators$, or b) a fraction of n proportional to p it it plays $AddMoreValidators$.

p is the probability that the proposing node (player x) might attack (plays $Cheat$). In response to this probability, a validation-leader (player y) chooses the appropriate strategy that will maximize its payoff (i.e., whether or not to $AddMoreValidators$). Hence, we consider p as the "*risk likelihood*" of an attack. p is computed with the assumption that the validation-leader is 'risk-neutral,' that is in a fair game each player aims to maximize its expected payoff. In case if a validation-leader chooses to play $AddMoreValidators$, the number of validators (m) will be a random number bounded by the minimum number of validators (i.e. four) and a fraction of $\frac{n}{4}$ proportional to p (we choose $\frac{n}{4}$ because we have four validation-leaders). In other words, a validation-leader select a random number between 5 (the minimum number of validators plus one) and $\frac{p.(n-2)}{4}$ (excluding the proposing and the validation-leader nodes) as flows:

$$m = Random[5, \tfrac{p.(n-2)}{4}]$$

After a validation-leader decides its m, it selects its validators, instructs them, and broadcasts m to all nodes. When a node in the network receive all the ms from the validation-leaders, it calculate the overall number of the validators involving in the protocol (M) as flows:

$$M = \Sigma_{i=1}^{4} m_i$$

Note that our protocol inherits the Byzantine tolerance provided by Tendermint. In other words, the system can work with one faulty leader, of which M is the aggregation of only three ms. In case if more than one leader is faulty, each node in the network waits for a time period named "$leader-time-out$" and then switches to "$all-validate$" mode. In this mode, every node in the network votes on the received M to agree on it (details are not provided due to space limitation). This mode is costly but preserves the consensus liveness.

Selecting validators Each validation-leader selects its set of m validators. The four sets of selected validators will be responsible for validating the proposed block. Our protocol is based on Tendermint which involves two steps of voting (pre-vote and pre-commit). The validators are selected randomly, and each set of selected validators is only known to their validation-leader. A validator is only known, to the other nodes in the network, when it contributes to one of the voting steps. Therefore, an adversary can observe the validators after revealing their identities in executing the first stage of voting (i.e., pre-voting). As a result, a powerful adversary might be able to attack or corrupt a sufficient number of them which can result in not executing the second step of voting (i.e., pre-committing). In response to this issue, our protocol requires each validation-leader to select two sets of nodes of size m. The first set is the validators/pre-voters, and the second one is the pre-committers. The pre-voters are responsible for executing the first step of voting, and the pre-committers execute the second step. As a result, the adversary discovers a participating node in the voting only after

Algorithm 3: Validators' Selection

```
Input : A population $V$ of $\frac{n-2}{4}$ nodes having reputation values, AND the risk
        likelihood $p$
Output: A set of validators/pre-voters $PV$ and a set of pre-committers $PC$ of
        size $m$
if $AddMoreValidators$ then
    $m = Random[5, \frac{p.(n-2)}{4}]$
else
    $m \leftarrow 4$
end
for $k \leftarrow 1$ to $m$ do
    $p_i(k) \leftarrow \frac{R_i}{\Sigma_{v_j \in V-PV} R_j}$
    Randomly select $v_i$ with probability $p_i(k)$ from $V - PV$
    $PV.add(v_i)$
end
for $l \leftarrow 1$ to $m_i$ do
    $p_i(l) \leftarrow \frac{R_i}{\Sigma_{c_j \in V-PC} R_j}$
    Randomly select $c_i$ with probability $p_i(l)$ from $V - PC$
    $PC.add(c_i)$
end
Return $PV$ AND $PC$
```

giving its vote, which is an unuseful knowledge. Algorithm 3 shows the process of selecting the validators/pre-voters and pre-committers.

After selecting the validators and pre-committers nodes, each validation-leader needs to include a proof of eligibility τ for each selected node to prove that a legitimate validation-leader has selected this node. τ is a digital signature signed by the validation-leader's private key *vl.sk* and includes the validation-leader's public key *vl.pk*, the selected node's public key (*pv.pk* for a pre-voter and *pc.pk* for a pre-committer), and the validation-leader's proof π as flows:

$$\tau \leftarrow Sign_{vl.sk}(vl.pk||pv.pk||\pi)$$

A node which receives a vote accompanied by τ from a voting node (i.e, pre-voter or pre-committer) needs to perform two verifications. First, it needs to verify τ using the validation-leader's public key *vl.pk*. Second, after successful verification of τ, the node verifies π using the initiator's public key *in.pk*.

Second-Stage Game After selecting the validators by their leaders, and after proposing and broadcasting the new block by the proposer, the second-stage game takes place between the proposer (player x) and each of the validators (player z). The strategy profile for a validator is a) $Validate$, and b) $NotValidate$. This game is modeled similarly to the first stage game. Table 4 and 5 show the strategic form of the second-stage Bayesian game. The extensive form of this game is similar to the one in the first-stage game which was illus-

Table 4. Strategic form of the second-stage Bayesian game (Player x is malicious)

Game Matrix		**player z (Validator)**	
		$Validate$	$NotValidate$
player x	$Cheat$	$(\beta.c_x) - w_x$, $(\beta.g_z) - w_z$	$(\beta.g_x) - w_x$, $\beta.c_z$
	$NotCheat$	0, $-w_z$	0, 0

Table 5. Strategic form of of the second-stage Bayesian game (Player x is regular)

Game Matrix		**player z (Validator)**	
		$Validate$	$NotValidate$
player x	$NotCheat$	0, $-w_z$	0, 0

trated previously in Figure 1. We use the same notations presented in Table 1 with following additional notations. w_z is the work done by the validator (player z) to play $Validate$. g_z is the gain for player z from capturing a cheater. c_z is the cost for player z if fails capturing a cheater.

A. Game Pure-Strategy BNE: We follow similar analysis that we presented in the first-stage game. If player x plays his pure strategy ($Cheat$ if malicious, $NotCheat$ if regular), then, the expected payoff of player z playing his pure strategy $Validate$ is:

$$E\mu_z(Validate) = \{\mu.[(\beta.g_z) - w_z]\} + \{(1-\mu). - w_z\}$$

The expected payoff of player z playing his pure strategy $NotValidate$ is:

$$E\mu_z(NotValidate) = \mu.(\beta.c_z)$$

As a result if, $E\mu_z(Validate) > E\mu_z(NotValidate)$ Or,

$$\mu > \frac{w_z}{\beta(g_z - c_z)} \tag{7}$$

Then, the best response of player z is to play $Validate$. Therefore, (($Cheat$ if malicious, $NotCheat$ if regular), $Validate$, μ) is not a (**BNE**). However, if $E\mu_z$ ($Validate$) $<$ $E\mu_z$ ($NotValidate$) Or,

$$\mu < \frac{w_z}{\beta(g_z - c_z)} \tag{8}$$

Then, the best response for player z is to play $NotValidate$ and thus (($Cheat$ if malicious, $NotCheat$ if regular), $NotValidate$, μ) is a *pure-strategy* **BNE**. Nevertheless, similar to the first-stage game, (($NotCheat$ if *Malicious*, $NotCheat$ if *Regular*), $NotValidate$) is not a **BNE**.

B. Game Mixed-Strategy BNE: Let p' be the probability with which player x plays $Cheat$. Let q' be the probability with player z plays $Validate$. The expected payoff of z playing $Validate$ is:

$$E\mu_z(Validate) = \{p'.\mu.[(\beta.g_z) - w_z]\} + \{(1-p').\mu. - w_z\} + \{(1-\mu). - w_z\}$$

The expected payoff of z playing $NotValidate$ is:

$$E\mu_y(NotValidate) = p'.\mu.(\beta.c_z)$$

So, the defender (player z) plays $Validate$ when:

$$p' > \frac{w_z}{\mu\beta(g_z - c_z)} \tag{9}$$

Similarly, we acquire the expected payoffs the attacker (player x). The expected payoff of x playing $Cheat$ is:

$$E\mu_x(Cheat) = \{q'.\mu.[(\beta.c_x) - w_x]\} + \{(1-q').\mu.[(\beta.g_x) - w_x]\}$$

The expected payoff of x playing $NotCheat$ is:

$$E\mu_x(NotCheat) = 0$$

As a result, the attacker (player x) plays $Cheat$ when:

$$q' > \frac{w_x - (\beta g_x)}{\mu\beta(g_z - c_z)} \tag{10}$$

As in the first-stage game, this game is 1-to-M game, where M is the number of validators (player z). Hence, our new *mixed-strategy* **BNE** is: ((q'' if malicious, $NotCheat$ if regular), p', μ), where:

$$q'' = q' - (p'' - p') \tag{11}$$

And:

$$p'' = (M.p') - p'^M \tag{12}$$

Consensus Voting We use Tendermint's voting mechanism, where there exists two steps of voting. The first one is 'pre-vote.' In this step, the validators/pre-voters validate the block and pre-vote on it. There are three types of pre-votes according to the outcome of the validation process: a) *valid* if the block is valid, 2) *invalid* if it is not, and c) *timed-out* if it is not received in a particular time window. Each validator validates with the probability p'. If a validator chooses to play $Validate$, then it contributes to the validation process; otherwise, it pre-votes *nil* and remains idle. The second step of voting is 'pre-commit.' A pre-committer advances to this step only if it receives more than two-thirds of M pre-votes (i.e., $> \frac{2.M}{3}$). The type of pre-commit (i.e., *valid*, *invalid*, or *timed-out*) depends on the type of the received pre-votes. Likewise, the block is committed or rejected if more than two-thirds of M pre-commits are received.

4 Security Analysis

In this section, we briefly present a threat model and demonstrate how our protocol protects against it. Utilizing the randomness and blind assignment protects from various attacks. Besides, exploiting game theory motivates the defenders in our protocol to adhere to the protocol, and disincentivizes malicious parties.

4.1 Randomness and Anonymity of Validators' Selection

As mentioned in Section 3, each proposer is blindly and randomly mapped to four leaders. Moreover, on each new block proposal, M number of validators are selected at random. The M value is also generated randomly proportional to the risk likelihood of the proposer. This selection approach protects against the following attacks:

DDoS attacks: The DDoS attack is more likely to happen if the set of validators is known in advance. Such an attack can happen to undermine the blockchain and can be launched from inside or outside the network. Validators' replaceability and randomizing their selection can significantly mitigate this attack. This is because of the set of validators changes randomly, and their identities remain anonymous until they participate in the consensus voting. Besides, each step of voting has a different set of voters. Thus, launching a DDoS attack is almost impossible and require to attack all the nodes in the network to undermine the system. Similarly, attacking the validation-leaders is hard too since leaders are known only after completing their tasks (i.e., broadcasting the m value and instructing their selected validators/pre-voters and pre-commiters). Note that we only aim to protect the validation and consensus process form DDoS attacks.

Eclipse attacks: This attack is presented by Heilman et al. [8] and allows an attacker who controls an adequate number of IP addresses to manage all connections to and from a victim node. As a result, the adversary can utilize the victim nodes for attacks on block validation and consensus system. As in the DDoS attack, an adversary mounting this attack needs to know the node participating in the validation and consensus process in advance. Introducing variable random validators on each block's proposal makes the adversary's job more difficult.

Validators' or leaders' bribing or corruption: An example of this attack is when a malicious proposer bribes and convinces other leaders or voters to accept and vote for an invalid block. Performing such an attack requires knowing the identities of the targeted nodes. Our protocol anonymizes the interaction between the consensus and validation parties, which overcomes such an attack.

4.2 Game Theory

Utilizing game theory protects against several real attacks. We model the interaction between a proposer and leaders or validators as an attacker-defender game. This way the defenders will work hard to maximize the utility that each can gain as a reward for excellent work and avoid the punishment (cost) that might incur due to misbehaving or not obeying the protocol. Our theoretical game approach can protect against the following attacks:

Faulty or lazy validation-leader: This attack happens when a validation-leader colludes with its corresponding malicious proposing node. It could result in many problems such as utilizing the minimum number of validators or colluding with other malicious nodes as validators. Another type of this attack is the lazy validation-leading, in which the validation-leader does not execute the protocol or does not obey its requirements. For example, it is possible for a validation-leader node to produce an assignment that is not truly random. Utilizing the reward and punishment payoffs provided by the proposed game mitigates the incentives of such an attack.

Block withholding attack Rosenfeld [18]: In this attack, a dishonest validator does not participate in the validation process or does not reveal the result of the verification in favor of a malicious proposing node. The reward and punishment provided by our game incentivize the validators to avoid this attack.

4.3 Randomness and Game Theory

The main attack that we are defending against is the **adversary proposer (malicious proposed block)**. This attack happens when a proposing node maliciously proposes an invalid new block. This attack occurs for various incentives; double spending is one of them. Integrating random and anonymous validators' selection with a game theoretical model contributes substantially in mitigating this attack. This is because a malicious proposer does not know the nodes that will validate its proposed block, which makes it hard to corrupt or bribe them to agree on its invalid block. Additionally, the punishment enforced by the game model could alleviate the attack motives.

5 Conclusion and Future Work

We have proposed a new true decentralized consensus protocol utilizing game theory and randomness. Our protocol randomly employs a different set of different size of validators each time a new block is proposed to protect against several real attacks mounted by powerful adversaries. Additionally, our protocol enjoys the feature offered by game theory to reward honest adhered parties and punish malicious ones.

This work, however, is in progress and has a few open problems. First, the probability μ that player x is of type *malicious* is static. To overcome this, μ can be determined dynamically, where the defenders y and z update their beliefs about μ at every stage of the game. Second, the proposers-leaders mapping guarantees the anonymous mapping only for one round of proposing. In other words, when a proposer proposes a block for the second time, its leaders' identities are revealed. This protocol was originally designed for a products' supply chain [1] where each node authenticates the product and proposes a block for it only once. Nevertheless, our protocol is suitable for many other blockchain applications, and we plan to make the proposers-leaders mapping dynamic in a way that preserves the anonymity. Third, the detailed evaluation of the protocol's safety and liveness nor the efficiency compared with other consensus protocols such as Ouroboros [11] are not presented in this paper.

References

1. Alzahrani, N., Bulusu, N.: Securing pharmaceutical and high-value products against tag reapplication attacks using nfc tags. In: Smart Computing (SMARTCOMP), 2016 IEEE International Conference on. IEEE (2016)
2. Alzahrani, N., Bulusu, N.: Block-supply chain: A new anti-counterfeiting supply chain using nfc and blockchain. In: Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems. pp. 30–35. ACM (2018)
3. Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains. Ph.D. thesis (2016)
4. Efraimidis, P.S., Spirakis, P.G.: Weighted random sampling with a reservoir. Information Processing Letters **97**(5), 181–185 (2006)
5. Eyal, I.: The miner's dilemma. In: Security and Privacy (SP), 2015 IEEE Symposium on. pp. 89–103. IEEE (2015)
6. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 51–68. ACM (2017)
7. Gorbunov, S.: Pure Proof-of-Stake Blockchains, https://medium.com/algorand
8. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX Security Symposium. pp. 129–144 (2015)
9. Hyperledger: hyperledger/fabric, https://github.com/hyperledger/fabric/tree/v0.6
10. Johnson, B., Laszka, A., Grossklags, J., Vasek, M., Moore, T.: Game-theoretic analysis of ddos attacks against bitcoin mining pools. In: International Conference on Financial Cryptography and Data Security. pp. 72–86. Springer (2014)
11. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual International Cryptology Conference. pp. 357–388. Springer (2017)
12. Kwon, J.: Tendermint: Consensus without mining. Retrieved May **18** (2014)
13. Kwon, J., Buchman, E.: Cosmos: A network of distributed ledgers (2016)
14. Luu, L., Saha, R., Parameshwaran, I., Saxena, P., Hobor, A.: On power splitting games in distributed computation: The case of bitcoin pooled mining. In: Computer Security Foundations Symposium (CSF), 2015 IEEE 28th. pp. 397–411. IEEE (2015)
15. Mackey, T.K., Nayyar, G.: A review of existing and emerging digital technologies to combat the global trade in fake medicines. Expert opinion on drug safety **16**(5) (2017)
16. Nojoumian, M., Golchubian, A., Njilla, L., Kwiat, K., Kamhoua, C.: Incentivizing blockchain miners to avoid dishonest mining strategies by a reputation-based paradigm. In: IEEE Computing Conference (CC). London, UK: IEEE (2018)
17. Pilkington, M.: 11 blockchain technology: principles and applications. Research handbook on digital transformations (2016)
18. Rosenfeld, M.: Analysis of bitcoin pooled mining reward systems. arXiv preprint arXiv:1112.4980 (2011)
19. Sheridan, K.: Blockchain All the Rage But Comes With Numerous Risks, https://www.darkreading.com/vulnerabilities—threats/blockchain-all-the-rage-but-comes-with-numerous-risks/d/d-id/1332038
20. Xu, D., Xiao, L., Sun, L., Lei, M.: Game theoretic study on blockchain based secure edge networks