Game-theoretic Randomness for Proof-of-Stake

Abstract. Many protocols in distributed computing rely on a source of randomness, usually called a random beacon, both for their applicability and security. This is especially true for proof-of-stake blockchain protocols in which the next miner or set of miners have to be chosen randomly and each party's likelihood to be selected is in proportion to their stake in the cryptocurrency. The chosen miner is then allowed to add a block to the chain.
Current random beacons used in proof-of-stake protocols, such as Ouroboros and Algorand, have two fundamental limitations: Either (i) they rely on pseudorandomness, e.g. assuming that the output of a hash function is uniform, which is an unproven assumption, or (ii) they generate their randomness using a distributed protocol in which several participants are required to submit random numbers which are then used in the generation of a final random result. However, in this case, there is no guarantee that the numbers provided by the parties are truly random and there is no incentive for the parties to honestly generate uniform randomness. Most random beacons have both limitations.
In this work, we provide a protocol for distributed generation of randomness. Our protocol does not rely on pseudorandomness at all. Similar to some of the previous approaches, it uses random inputs by different participants to generate a final random result. However, the crucial difference is that we provide a game-theoretic guarantee showing that it is in everyone's best interest to submit truly uniform random numbers. Hence, our approach is the first to incentivize honest behavior instead of just assuming it. Moreover, the approach is trustless and generates unbiased random numbers. It is also tamper-proof and no party can change the output or affect its distribution. Finally, it is designed with modularity in mind and can be easily plugged into existing distributed protocols such as proof-of-stake blockchains.

Keywords: Distributed Randomness · Proof-of-stake · Blockchain · Mechanism Design

1 Introduction

Proof of Work. Bitcoin, the first blockchain protocol, was proposed by Satoshi Nakamoto to achieve consensus in a decentralized peer-to-peer electronic payment system [15]. In Bitcoin and many other cryptocurrencies, the miners are selected by a proof of work (PoW) mechanism to add blocks of transactions to the public ledger, i.e. they have to compete in solving a mathematical puzzle and each miner's chance of adding the next block is proportional to their computational (hash) power. Security guarantees are then proven with the assumption that more than half of computational power is in the hands of honest miners. Proof of work is known to be highly energy-inefficient [2] and also prone to centralization due to large mining pools [3]. Currently, the three largest mining pools have more than half of the Bitcoin mining power.

Proof of Stake [12]. Proof of Stake (PoS) is the main alternative consensus mechanism proposed to replace PoW in blockchain protocols. In a PoS protocol, miners are chosen randomly and each miner's chance of being allowed to add the next block is often proportional to their stake in the currency. Hence, instead of relying on the assumption that

a majority of the computational power on the network is owned by honest participants, the security claims of proof-of-stake protocols rely on the assumption that a majority, or a high percentage, of the stake is owned by honest participants. Despite their differences, all proof-of-stake protocols require a random beacon to randomly select the next miners in an unpredictable manner.

Distributed Randomness. A random beacon is an ideal oracle used in a distributed protocol, e.g. a proof-of-stake blockchain, that emits a fresh random number in predetermined intervals. Designing random beacons is an active research topic in the context of distributed and decentralized computation [20, 18, 21, 13]. The desired properties of a random beacon are as follows:

- *Bias-resistance:* The output should always be sampled according to a fixed underlying distribution δ, which is usually the uniform distribution. No party should be able to bias the output or change the distribution δ.
- *Unpredictability:* No party should be able to predict the output before it is publicized. Moreover, no party should even be able to have any extra information about the output, other than the fact that it will be sampled from δ.
- *Availability:* Each execution of the beacon must successfully terminate and produce a random value.
- *Verifiability:* Each execution of the beacon should provide a "proof" such that any third parties who were not involved in the random beacon are able to verify both the output and the fact that the random beacon executed successfully.

Reliable Participants. Almost all distributed randomness protocols have several participants and create a random output based on random numbers submitted by participants of the protocol. Usually, the final value is simply defined by the modular sum of all input values by participants modulo some large number p, i.e. $s := \sum_{i=1}^{n} s_i \pmod p$. If the protocol generates only a single random bit, then $p = 2$ and the modular sum is equivalent to the `xor` operation. Using the summation formula above, if the input values of different participants are chosen independently and if at least one of the participants submits a truly uniform random value in the range $[0, p-1]$, then the final output is also a uniform random value. We call such a participant *reliable*. Note that it is enough to have only one reliable participant for the final output to have the uniform distribution. Therefore, the distributed randomness protocols typically assume that at least one of the participants is reliable. We distinguish between *reliable* and *honest* participants.

Honest Participants. An honest participant is a participant who correctly follows the protocol, e.g. submits their random number s_i in time. Distributed randomness protocols often assume and require that a large proportion of participants are honest and obey the communication rules to complete and produce final values. For example, PBFT achieves byzantine agreement in a partially-synchronous network by requiring that more than two-thirds of all participants be honest [7].

Commitment Schemes. Using the formula above for random number generation, since the participants cannot broadcast their values in a distributed network in a perfectly simultaneous way, the last participant has an advantage and can dominate the final output. The classical cryptographic primitive used to avoid such a scenario is a commitment scheme. A commitment scheme runs in two phases: a commit phase and a reveal phase. In the commit phase, instead of broadcasting the value s_i directly, each party p_i broadcasts $h(s_i, r_i)$, where h is a cryptographic hash function and r_i is a randomly chosen nonce. In the reveal phase, each party broadcasts the values of s_i and r_i and everyone on

the network can verify that the broadcast values have the right hash hence the party has not changed their choice s_i since the commit phase. However, a commitment scheme does not ensure availability, since malicious parties might only commit but not reveal their values.

PVSS. Publicly verifiable secret sharing (PVSS) is a powerful cryptographic tool to ensure the revelation of values s_i even if some malicious parties stop participating in the reveal phase of a commitment scheme [19]. PVSS adds a protection layer to traditional secret sharing schemes in the presence of malicious participants. In a PVSS scheme, a dealer is required to provide a non-interactive zero-knowledge proof (NIZK) along with encrypted secret shares $E_i(s_i)$ to guarantee the validity of secret shares. During the reconstruction phase, a participant sends its secret share to other participants along with a NIZK proof to guarantee the correctness of secret share. The NIZK proofs can be verified by any party, including third parties who are not taking part in the PVSS scheme.

RANDAO [1]. RANDAO is a family of smart contracts that produce random numbers. Anyone can participate and submit a random value to contribute to the output. RANDAO uses a commitment scheme. Compared to general distributed randomness protocols based on distributed networks, RANDAO's smart contracts run on a blockchain with consensus and directly interact with the underlying cryptocurrency. Therefore, RANDAO naturally enjoys the decentralized consensus provided by the blockchain protocol. Besides, economic incentives can be designed to promote honesty. Cheaters who violate the rules are punished economically, e.g. by having their deposit confiscated. On the other hand, honest participants are rewarded by the income generated from providing the random number generation service to external contracts. However, there is no way to ensure bias-resistance and availability. A malicious party might choose not to reveal their value s_i as it might be beneficial to them to bias the output. So, if a party does not reveal values, the whole random number generation process should be repeated, but even this biases the output as a malicious party can choose not to reveal only when the final result is not to their benefit in an external smart contract. Finally, RANDAO does not incentivize reliability and assumes that a reliable party exists, without arguing why.

VDFs. Verifiable delay functions [6] can be used to ensure bias-resistance in distributed randomness protocols. A VDF is a function whose evaluation takes at least some predetermined number of sequential steps, even with many parallel processors. Once the evaluation is complete, it can provide a publicly verifiable proof for the evaluation result, which can also be checked by any third party efficiently.

VRFs. Verifiable random functions [14, 10] are widely used in PoS blockchain protocols [11, 9]. A party can run a VRF locally, producing a pseudo-random output value based on their secret key and random seed. The VRF also outputs a proof of the output that can be verified by anyone with access to the party's public key and random seed. With the use of VRFs, malicious parties cannot predict who the future miners are before the miners announce their identities themselves.

Algorand. Algorand [11] is a proof-of-stake blockchain protocol based on Byzantine agreement. The random seed for its VRF is based on the VRF of the previous round. While this guarantees most of the desired properties, a major drawback of this randomness beacon is that the generated numbers are not guaranteed to be uniform.

Ouroboros and Ouroboros Praos. Ouroboros [12] was the first provably secure proof-of-stake blockchain protocol. It uses a publicly verifiable secret sharing scheme to gen-

erate a fresh random seed for each epoch. However, in this scheme the participants have no incentive to submit a uniform random value. In other words, there is no incentive to be reliable, but just to be honest. Ouroboros Praos [9] improves over Ouroboros to be provably secure under a semi-synchronous setting. The random seed of Ouroboros Praos is updated every epoch by applying a random oracle hash function to a concatenation of VRF outputs in the previous epoch. Similar to Algorand, the random numbers are not guaranteed to be uniformly random, despite the fact that they are assumed to be uniform in the security analysis.

Our Contribution. Our main contributions are as follows:

- First, we design a novel game-theoretic approach for randomness generation. We call this an RIG (random integer generation) game. RIG efficiently produces a uniform random integer from an arbitrarily large interval. Moreover, we show that the only equilibrium in an RIG is for all participants to choose their s_i uniformly at random. In other words, our RIG ensures that the participants are incentivized not only to be honest, but also to be reliable. This will alleviate the problems with the previous approaches and ensure that all desired properties of distributed randomness are attained.
- We show that our RIG approach can be plugged into common randomness generation protocols with ease. In Section 4, we design protocols to implement RIG as a random beacon on general proof-of-stake blockchains. We describe RIG protocols based on commitment schemes and VDFs in Section 4.1 and RIG protocols based on PVSS in Section 4.2.
- In Section 5, we discuss how RIG can be deployed with minor changes in particular proof-of-stake protocols. We cover Algorand [11] and Ouroboros Praos [9].

Our protocols are the first to incentivize participants to be reliable and submit truly uniform random numbers. In comparison, previous distributed randomness protocols using commitment schemes and PVSS assume that there is at least one reliable participant without incentivizing reliability. In other words, they only reward honesty but assume both honesty and reliability. The reliability assumption is unfounded. Several other randomness protocols, including Algorand and Ouroboros Praos, do not depend on random inputs from participants at all, but instead use real-time data on blockchains and cryptographic hash functions to generate pseudo-random numbers. This pseudo-randomness is not guaranteed to be uniform. Hence, there is no guarantee that miners get elected with probabilities proportional to their stake.

2 Preliminaries

2.1 Games and Equilibria

Probability Distributions. Given a finite set $X = \{x_1, \ldots, x_m\}$, a probability distribution on X is a function $\delta : X \to [0, 1]$ such that $\delta(x_1) + \cdots + \delta(x_m) = 1$. We denote the set of all probability distributions on X by $\Delta(X)$.

One-shot Games [17]. A *one-shot game* with n players is a tuple $G = (S_1, S_2, \ldots, S_n, u_1, u_2, \ldots, u_n)$ where:

- Each S_i is a finite set of *pure strategies* for players i and $S = S_1 \times S_2 \times \cdots \times S_n$ is the set of all *outcomes*; and
- Each u_i is a *utility function* of the form $u_i : S \to \mathbb{R}$.

In a play of the game, each player i chooses one strategy $s_i \in S_i$. The choices are simultaneous and independent. Then each player i is paid a $utility$ of $u_i(s_1, s_2, \ldots, s_n)$ units.

Mixed Strategies [17]. A *mixed strategy* $\sigma_i \in \Delta(S_i)$ for player i is a probability distribution over S_i, that characterizes the probability of playing each pure strategy in S_i. A *mixed strategy profile* is a tuple $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ consisting of one mixed strategy for each player. The *expected utility* $u_i(\sigma)$ of player i in a mixed strategy profile σ is defined as $u_i(\sigma) = \mathbb{E}_{s_i \sim \sigma_i}[u_i(s_1, s_2, \ldots, s_n)]$. Intuitively, in a mixed strategy, the player is not committing to a single pure strategy, but only to the probability of playing each pure strategy.

Nash Equilibria [16]. A *Nash Equilibrium* of a game G is a mixed strategy profile σ, such that no player has an incentive to change their mixed strategy σ_i, assuming they are aware of the mixed strategies played by all the other players. Let σ_{-i} be a tuple consisting of all the mixed strategies in σ except σ_i. Formally, σ is a Nash equilibrium if and only if for all $\tilde{\sigma}_i \in \Delta(S_i)$ we have $u_i(\sigma) \geq u_i(\tilde{\sigma}_i, \sigma_{-i})$. A seminal result by Nash is that every finite game G has a Nash equilibrium [16]. Nash equilibria are the central concept of stability and self-enforceability for non-cooperative games [17], in which each player maximizes their own utility, i.e. when a game is in a Nash equilibrium, no party has an incentive to change their strategy and hence the game remains in the Nash equilibrium.

In distributed randomness generation, especially for proof-of-stake protocols, we aim to have a committee that plays a game whose output is our random number. Since the players/parties are pseudonymous on a blockchain network and only participate using their public keys, in our committee we might have multiple accounts that are actually controlled by the same person or are in an alliance. Therefore, we need a stronger concept of equilibrium that does not assume a lack of cooperation between any pair of players. Thus, we rely on strong and alliance-resistant equilibria as defined below.

Strong Nash Equilibria [4, 5]. A *strong Nash equilibrium* is a mixed strategy profile in which no group of players have a way to cooperate and change their mixed strategies such that the utility of every member of the group is increased. Formally, σ is a strong Nash equilibrium if for any non-empty set P of players and any strategy profile $\tilde{\sigma}_P$ over P, there exists a player $p \in P$ such that $u_p(\sigma) \geq u_p(\tilde{\sigma}_P, \sigma_{-P})$. In strong equilibria, the assumption is that the players cannot share or transfer their utilities, so a player agrees to a change of strategies in the alliance P if and only if their own utility is strictly increased. However, if the players can share and redistribute utilities, or if they are indeed controlled by the same person, then a group is willing to defect as long as their *total* utility increases, which leads to an even stronger notion of equilibrium:

Alliance-resistant Nash Equilibria [8]. An *alliance-resistant Nash equilibrium* is a mixed strategy profile σ such that for any non-empty set P of players and any strategy profile $\tilde{\sigma}_P$, it holds that $u_P(\sigma) \geq u_P(\tilde{\sigma}_P, \sigma_{-P})$, where u_P is the sum of utilities of all member of P. In our setting, especially in PoS blockchain protocols, alliance-resistant equilibria are the effective notion to justify stability and self-enforceability, because a person with a large stake is likely to control multiple players in the randomly selected committee and only care about the overall revenue.

2.2 Publicly Verifiable Secret Sharing

We follow [19] in our description of PVSS. In a PVSS scheme, a dealer D wants to share a secret s with a group of n participants $P_1, P_2, \ldots, P_n$. The goal is to have a (t, n)-threshold scheme, i.e. any subset of t participants can collaborate to recover the secret s, while any smaller subset of participants cannot recover the secret or obtain any information about it. Moreover, anyone on the network, even those who are not participating, should be able to verify that the dealer is acting honestly and following the protocol.

Initialization. We assume that a multiplicative group $\mathbb{Z}_q^*$ and two generators g, G of this group are selected using an appropriate public procedure. Here, q is a large prime number and all calculations are done modulo q. Each participant P_i generates a non-zero private key $x_i \in \mathbb{Z}_q^*$ and announces $y_i = G^{x_i}$ as their public key. Suppose the secret to be shared is s, the dealer first chooses a random number r and publishes $U = s + h(G^r)$, where h is a pre-selected cryptographic hash function. The dealer then runs the main protocol below to distribute the shares that can reveal G^r. The main protocol consists of two steps: (1) distribution, and (2) reconstruction, each of which has two substeps.

Distribution. This consists of the following:

- *Distribution of the shares*. The dealer picks a random polynomial p of degree at most $t-1$ with coefficients in $\mathbb{Z}_q$ of the form $p(x) = \sum_{j=0}^{t-1} \alpha_j \cdot x^j$. Here, we have $\alpha_0 = G^r$, i.e. the number r is encoded in the first coefficient of the polynomial and every other α_j is a random number from $\mathbb{Z}_q$. The dealer then publishes the following:
  - *Commitment:* $C_j = g^{\alpha_j}$, for $0 \leq j < t$. This ensures that the dealer is committing to the polynomial and cannot change it later.
  - *Encrypted shares:* For each player P_i, the dealer computes and publishes $Y_i = y_i^{p(i)}$, for $1 \leq i < n$. Intuitively, the dealer is taking the value $p(i)$ of the polynomial p at point i and encrypting it using y_i so that only the i-th player can decrypt it. This encrypted value is then published.
  - Proof of correctness: The dealer provides a non-interactive zero-knowledge proof ensuring that the encrypted shares above are valid. See [19] for details.
- *Verification of the shares.* Anyone on the network, be it a player P_i or a non-participant third-party, can verify the proof and encrypted shares provided by the dealer to ensure that the dealer is acting honestly, i.e. following the protocol above, and not giving out invalid shares.

Reconstruction. This step consists of:

- *Decryption of the shares.* Each party P_i knows $Y_i = y_i^{p(i)}$ and their secret key x_i. Recall that $y_i = G^{x_i}$. Hence, the i-th party can compute $Y_i^{1/x_i} = y_i^{p(i)/x_i} = G^{p(i)}$. They publish G^{p_i} along with a non-interactive zero-knowledge proof of its correctness.
- *Pooling the shares.* Any t participants $P_{i_1}, P_{i_2}, \ldots, P_{i_t}$ can compute the G^r by Lagrange interpolation. More specifically, they know t points $(i_j, p(i_j))$ of the polynomial p that is of degree $t-1$. So, they can find the unique polynomial that goes through these points. Note that after all the shares are decrypted, anyone on the network can use t of the shares to compute the polynomial p and then G^r is simply $p(0)$. However, before the decryption of the shares in the reconstruction step,

finding G^r requires the collaboration of at least t participants and no set of $t-1$ participants can obtain any information about G^r. Finally, knowing G^r and U, it is easy to find the secret s, i.e. $s = U - h(G^r)$.

A PVSS scheme can be used to generate random numbers. To do so, we use a separate PVSS scheme for each participant P_i. All n PVSS schemes run in parallel. In the i-th scheme, P_i is the dealer and everyone else is a normal participant. P_i first chooses a random number s_i and then performs the protocol above as the dealer. At the end of the process, all the s_i's are revealed by pooling the shares and we can use $s = \sum_{i=1}^{n} s_i$ as our random number. The upside is that no party can avoid revealing their s_i and hence the protocol satisfies availability. The downside is that every set of t parties can unmask everyone else's choices and hence bias the result. Therefore, in random number generation using PVSS we have to assume that there are at most $t-1$ dishonest participants.

2.3 Verifiable Delay Functions

We follow [6] in our treatment of verifiable delay functions. A verifiable delay function (VDF) is a tuple $V = (\mathbf{Setup}, \mathbf{Eval}, \mathbf{Verify})$ parameterized by a security parameter λ and a desired puzzle difficulty t. Suppose our input space is $\mathcal{X}$ and our output space is $\mathcal{Y}$. V is a triplet of algorithms as follows:

- $\mathbf{Setup}(\lambda, \mathbf{t}) \to (ek, vk)$. This function generates an evaluation key ek and a verification key vk in polynomial time with respect to λ.
- $\mathbf{Eval}(ek, x) \to (y, \pi)$ takes an input $x \in \mathcal{X}$ and produces an output $y \in \mathcal{Y}$ and a proof π. **Eval** may use randomness in computing the proof π but not in the computation of y. It must run in parallel time t with $poly(\log(t), \lambda)$ processors.
- $\mathbf{Verify}(vk, x, y, \pi) \to \{\text{Yes, No}\}$ is a deterministic algorithm that verifies the correctness of evaluation in sequential time $poly(\log(t), \lambda)$.

See [6] for more details and desired properties. Intuitively, anyone can evaluate the VDF using the evaluation key. However, this takes a long time, i.e. at least t steps, even when using parallelization. Even if a malicious participant has strong parallel computational power, they cannot evaluate the VDF significantly faster than an ordinary participant that owns only a single processor. However, after the evaluation is done, verifying the result is easy and much faster and anyone can do the verification using the verification key vk.

The use case of verifiable delay functions in random number generation is to again defend against dishonest participants who do not reveal their choice in a commitment scheme. We can require every participant to provide a VDF whose evaluation is their choice s_i. Then, even if the participant is dishonest and does not reveal their own choice, other participants can evaluate the VDF and obtain the s_i, hence ensuring availability for our random number generation protocol. However, evaluation takes a long time, and hence the choice will not be revealed while in the commit phase.

Note that both PVSS and VDF methods above can be used to ensure availability and defend against dishonest parties who do not reveal their choices. However, they do not incentivize the parties to be reliable and choose their s_i uniformly at random. This is our main contribution in the next section.

3 Random Integer Generation Game (RIG)

We now provide the main component of our approach, i.e. a game to incentivize reliability in random number generation.

3.1 Overview of RIG

RIG. Suppose that we have n players and n is even. A *Random Integer Generation game* (RIG) with n players and $m \geq 3$ strategies is a game G in which:

- For every player $i \in \{1, \ldots, n\}$, we have m pure strategies $S_i = \{0, 1, \ldots, m-1\}$;
- We pair the players such that every even player is paired with the previous odd player and every odd player is paired with the next even player. In other words, $pair(2 \cdot k) = 2 \cdot k - 1$ and $pair(2 \cdot k - 1) = 2 \cdot k$.
- At an outcome $s = (s_1, s_2, \ldots, s_n)$ of the game, the payoff of player i is defined as $u_i(s) \coloneqq f(s_i, s_j)$ where $j = pair(i)$, and

$$f(s_i, s_j) \coloneqq \begin{cases} 1 & \text{if } s_i - s_j \equiv 1 \pmod{m} \\ -1 & \text{if } s_i - s_j \equiv -1 \pmod{m} \\ 0 & \text{otherwise} \end{cases}$$

Essentially, we assume that any adjacent pair of even player and odd player play a zero-sum symmetric one-shot game with each other. Their payoffs are independent of other $n - 2$ players. For each pair $(2 \cdot k - 1, 2 \cdot k)$ of players, this is a zero-sum matrix game with the following payoff matrix:

$$A = \begin{pmatrix} 0 & -1 & 0 & \cdots & 0 \\ 1 & 0 & -1 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

3.2 Analysis of Alliance-Resistant Nash Equilibria

Theorem 1. *(Alliance-Resistant Nash Equilibrium of an RIG.) Let G be an RIG game with n players and m strategies, where n is an even number and $m \geq 3$. Let $\bar{\sigma}$ be a mixed strategy profile defined by $\bar{\sigma}_i = (1/m, 1/m, \ldots, 1/m)$ for all i, i.e. the mixed strategy profile in which each player i chooses a strategy in S_i uniformly at random. Then, $\bar{\sigma}$ is the only Nash equilibrium of G. Further, it is also alliance-resistant.*

Proof. First, we prove that $\bar{\sigma}$ is an alliance-resistant Nash Equilibrium. Under the mixed strategy profile, the expected payoff of each one of the players is 0. Let G be a subset of players, then the overall utility of all players in G is $\sum_{i \in G} u_i(\bar{\sigma}_{-G}, \sigma)$ if players in G play another strategy profile σ. Each player i is effectively playing against its adjacent player. If both player i and player $pair(i)$ are in G, then $u_i(\bar{\sigma}_{-G}, \sigma) = -u_{pair(i)}(\bar{\sigma}_{-G}, \sigma)$. The utilities of these two players always sum up to zero, so that the overall utility of G is not influenced by them. Similarly, if both player i and player $pair(i)$ are not in G, they do not influence the overall utility of G either. The only non-trivial part consists of those players in G who play against players outside G. For

each such player i, since the player $pair(i)$ plays mixed strategy $\bar{\sigma}_{pair(i)}$, the utility is $u_i = \sigma_i^T \cdot A \cdot \bar{\sigma}_{pair(i)} = \sigma_i^T \cdot (0, 0, \ldots, 0) = 0$. Therefore, the overall utility of G is 0 and changing the strategy has no benefit.

We now prove that $\bar{\sigma}$ is the unique Nash equilibrium of this game. Suppose there is another strategy profile $\tilde{\sigma}$ that is also a Nash equilibrium. Then for any player i, since it is effectively only playing with its adjacent player $j = pair(i)$, it follows that $(\tilde{\sigma}_i, \tilde{\sigma}_j)$ forms a Nash equilibrium for the zero-sum bimatrix game defined by A.

Now consider the bimatrix game between player i and player j. Let their utility at Nash equilibrium mixed strategy profile $(\tilde{\sigma}_i, \tilde{\sigma}_j)$ be $(\tilde{u}_i, \tilde{u}_j)$. Since it is a zero-sum matrix game, $\tilde{u}_i + \tilde{u}_j = 0$. Without loss of generality, assume that $\tilde{u}_i \leq \tilde{u}_j$, then $\tilde{u}_i \leq 0$. By the definition of Nash equilibrium, player i cannot increase its utility by changing its strategy $\tilde{\sigma}_i$ to any other strategy σ_i, while player j keeps playing the same strategy $\tilde{\sigma}_j$. This indicates that every coordinate of the vector $A \cdot \tilde{\sigma}_j$ is no more than $\tilde{u}_i$, which is at most 0. Let $\tilde{\sigma}_j = (p_0, p_1, \ldots, p_{m-1})$, then $\delta_k = p_k - p_{k-2 \pmod m} \leq \tilde{u}_i \leq 0$, for each k in $\{0, 1, \ldots, m-1\}$. However, $\sum_{k=0}^{m-1} \delta_k = \sum_{k=0}^{m-1} p_k - \sum_{k=0}^{m-1} p_k = 0$. So it must hold that $\tilde{\sigma}_j = (1/m, 1/m, \ldots, 1/m)$ and $\tilde{u}_i = \tilde{u}_j = 0$. Since $\tilde{u}_j = 0 \leq 0$, similar analysis can show that $\tilde{\sigma}_i = (1/m, 1/m, \ldots, 1/m)$. This proves that $\bar{\sigma}$ is the only Nash equilibrium for the Random Integer Generation game.

The theorem above shows that it is in every player's best interest to play uniformly at random, i.e. choose each pure strategy in S_i with probability exactly $1/m$. Moreover, this equilibrium is self-enforcing even in the presence of alliances. Hence, we can plug this game into a distributed random number generation protocol and give participants rewards that are based on their payoffs in this game. This ensures that every participant is incentivized to provide a uniformly random s_i. As mentioned before, even if one participant is reliable and submits a uniformly random s_i, then the entire result $s = \sum s_i$ of the random number generation protocol is guaranteed to be unbiased. Hence, instead of assuming that a reliable party exists, we incentivize every party to be reliable.

3.3 Dense RIG bimatrix game

The matrix A is sparse if the strategy size m is large, which is the case if we want to generate integers from a large range. If the number of players is much smaller than m, then the probability that one party really receives a non-zero payoff is negligible. Therefore, it is desirable to design a matrix B that is dense, as well as provides the same unique alliance-resistant equilibrium property.

For simplicity of notation, suppose m is a power of 2 and $m \geq 2^3 = 8$. Then B is defined such that $B_{i,j} = g(j - i)$, where $g(\cdot)$ is defined as:

$$g(k) \coloneqq \begin{cases} 1 & \text{if } 1 \leq k \leq m/4 \pmod m \\ -1 & \text{if } 3m/4 \leq k \leq m-1 \pmod m \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to check that $g(-k) = -g(k)$, which indicates $B^T = -B$. It is easy to check that the mixed strategy profile $\bar{\sigma}$ is an alliance-resistant Nash equilibrium. To show that it is the only Nash equilibrium, we follow the analysis we did for A. Suppose there is another Nash equilibrium $(\tilde{\sigma}_i, \tilde{\sigma}_j)$ between player i and player j and $\tilde{u}_i \leq 0$. Let $\tilde{\sigma}_j$ be $(p_0, p_1, \ldots, p_{m-1})$, then every element of $r = B \cdot \tilde{\sigma}_j$ is at most $\tilde{u}_i$, which is

non-positive. However, $\sum_{k=0}^{m-1} r_k = \mathbf{1}^T \cdot r = \mathbf{1}^T \cdot B \cdot \tilde{\sigma}_j = \mathbf{0}^T \cdot \tilde{\sigma}_j = 0$, which requires $r = \mathbf{0}$ and $\tilde{u}_i = 0$. By $r = \mathbf{0}$, we have

$$r_s = \sum_{1 \le k \le m/4} p_{k+s} - \sum_{3m/4 \le k \le m-1} p_{k+s} = 0$$

for every $s \in \{0, 1, \ldots, m-1\}$. For simplicity of notation, assume that $p_{m+t} = p_t$ for any integer t. If we substract r_{s+1} by r_s, we get

$$p_{s+1+m/4} - p_{s+1} = p_{s+m} - p_{s+3 \cdot m/4} = p_s - p_{s-m/4}$$

Let $q(s) = p_s - p_{s-m/4}$, then $q(s+1+m/4) = q(s)$. We also have $q(s+m) = q(s)$. Since $\gcd(1+m/4, m) = 1$, $q(\cdot)$ is constant on integers, from which we can infer that $p_0 = p_1 = \cdots = p_{m-1} = 1/m$. Therefore, $\bar{\sigma}$ is still the only Nash equilibrium.

Remark. Note that simple parallelization of RIG loses the uniqueness property of Nash equilibrium, hence we cannot simply have an RIG game for $m = 2$, use it to generate a random bit, and parallelize it k times to generate k random bits. Instead, we must set $m = 2^k$ and have a single non-parallel RIG game. As an example, consider the simplified case of two players and m bits. If each player only uniformly randomly set their first bit, and then copy the same bit to all other bits, then this also forms a Nash equilibrium. However, this Nash equilibrium does not produce a uniformly random output. Instead, the output is 0 with 50% probability and $2^m - 1$ with 50% probability. More generally, any $\sigma = (\sigma_1, \sigma_2)$ such that $\sigma_i(j\text{-th bit is } 0) = 1/2$ for all $1 \le j \le m$ is a Nash equilibrium. The existence of these undesirable Nash equilibria breaks the guarantee of uniformly random distribution of final output value. Hence, parallelization should not be used and a game on 2^k strategies should be played in the first place.

4 Designing a Random Beacon Based on RIG

In this section, we discuss how to use a single execution of the Random Integer Generation game in a distributed random number generation beacon. The major challenge is to execute the game, in which the parties have to move simultaneously, in a decentralized environment. We propose two schemes to implement the RIG game: (1) using commitment schemes and verifiable delay functions, and (2) using publicly verifiable secret sharing. We assume that the set of players is already fixed. Usually, only a small subset of users (or blockchain miners) are selected from all the users in the system to join a single execution of the game (generation of a random number). The selection rule is determined by the specific application. The design and amount of reward/penalty and deposits are also subject to the specific application. We will address these adjustable configurations in Section 4.3. Finally, we focus on the case where our protocol is used in conjunction with a blockchain protocol, including gossip protocols for announcements.

4.1 Commitment Scheme and VDF approach

As mentioned above, commitment schemes are already widely used in random number generation in distributed systems. As expected, our approach has two phases: commit and reveal. The execution starts with the commit phase, which lasts for T_{commit} time

slots. In a blockchain ecosystem, we can use the block number to keep track of time. After the commit phase ends, the execution enters the reveal phase, which lasts for T_{reveal} time slots. The RIG game is executed when the reveal phase completes.

In the commit phase, each participant p_i broadcasts a signed commit message: $(session\_id, h_i, proof_i)_i$, where $session\_id$ is the session id of the execution and $h_i = hash(v_i|nonce_i)$ is the commitment. v_i is the value the participant chooses and $nonce_i$ is a random nonce. $proof_i$ is a publicly verifiable proof that p_i is an eligible participant, applicable when only a subset of selected users are allowed to join the game. A commit message is valid if: (1) the message is properly signed by p_i, (2) the message has a valid proof of participation, (3) there is no other different valid commit message from p_i in the network, and (4) the message is received during the commit phase.

In the reveal phase, each participant p_i broadcasts a signed reveal message: $(session\_id, v_i, nonce_i)_i$. A reveal message is valid if (1) the message is received during the reveal phase, (2) p_i has exactly one valid commit message, and (3) $hash(v_i|nonce_i)$ matches the commitment h_i of the participant p_i.

After the reveal phase completes, we can compute the payoffs of the RIG game. We describe in subsection 4.3 the details of computing results. Assume the outcome of the game is $(s_1, \ldots, s_n)$, where $s_i \in \{0, 1, \ldots, m-1\}$ is the strategy played by each player. We set $r := \sum s_i \pmod m$ and output it as the result of the random number generation protocol.

The value of r can be biased by malicious participants who might choose not to reveal their values/strategies. If a participant does not reveal the values after completing the commit phase correctly, they will lose their deposit. However, the participant might benefit from a biased output of the random beacon in the downstream application, for which they might be willing to cheat even at the cost of losing the deposit. In order to prevent this possibility of adversarial manipulation on the game result, we make use of a verifiable delay function $\mathrm{VDF}(\cdot)$ as in [6] and require the participant to provide all necessary parameters for the evaluation of the VDF as part of the commit message. We then check that the provided VDF really evaluates to the strategy s_i of the player and otherwise, simply remove player i from the game. Of course, the VDF evaluation time should be long enough to ensure it cannot be evaluated until the reveal phase is over.

Using this technique, any adversary cannot have any information about the final output by the end of the reveal phase. Therefore, revealing values honestly is always a strictly better strategy than not revealing values for all participants. The game is then executed when all the values are revealed and all the VDFs are evaluated and it is ensured that cheating participants are excluded.

Note that, even if v conforms to a uniformly random distribution, the output of VDF on v is not guaranteed to be uniformly random. In existing constructions of random beacons that rely on VDF, a hash function is applied to the output of VDF to get random numbers, under the random oracle assumption. However, with the novel RIG game, we can guarantee the delivery of uniformly random output values, if we define the final output as $\tilde{v} = v_1 + \mathrm{VDF}(v_2)$, where v_1 is the higher half bits of v and v_2 is the lower half bits of v. Since v is uniformly random, then v_1 and v_2 are independent uniformly random integers. Whatever distribution $\mathrm{VDF}(v_2)$ has, the sum $\tilde{v}$ is a uniformly random integer.

Finally, note that this approach works as long as at least one of the participants is honest. So, in a proof-of-stake scenario, if we choose n participants for each random number generation, we need to ensure that honest miners have much more than $1/n$

fraction of the stake in the cryptocurrency, so as to ensure that at least one honest participant is chosen with high probability. We also assume that at least one participant is reliable, but this is already incentivized by our game.

4.2 PVSS approach

The drawback of the commitment scheme in random number generators is the possibility of adversarial manipulation in the reveal phase by not revealing values. We have already seen a solution using VDFs. A publicly verifiable secret sharing scheme solves the same issue differently, i.e. by forcing the opening of commitments, at the cost of increased communication complexity. We follow the PVSS scheme in [19].

An execution of the RIG game in a PVSS scheme consists of three phases: prepare, distribute and reconstruct. The first phase is prepare, which lasts for $T_{prepare}$ time slots. The second phase, distribute, lasts for $T_{distribute}$ time slots and reconstruct for $T_{reconstruct}$ slots.

In the prepare phase, all eligible participants inform each other that they are selected. Under a synchronous communication setting, all honest participants can reach a consensus on the list of participants. More specifically, we assume a blockchain protocol that a transaction will be added to a finalized block within known bounded time slots after it is broadcasted. Each participant firstly broadcasts a signed prepare message $(session\_id, proof_i)_i$ to announce its identity along with eligibility proof for the current session of execution. By the end of prepare phase, all prepare messages should be included in the blockchain and are synchronized across all nodes. Suppose the list of participants is $\{P_i\}_{i=1}^n$.

In the distribute and reconstruct phases, each participant P_i runs a PVSS scheme to share their value s_i to the other $n-1$ participants. This is exactly as described in Section 2.2. In the distribute phase, every participant should send valid $(n-1,t)$-threshold secret shares to others along with a proof of commitment and consistency. The shares are publicly verifiable so that a dishonest participant who distributes invalid shares can be discovered and excluded from the game. Hence, by the end of distribute phase, all honest participants release their correct shares and receive correct shares from other honest participants. If a dishonest participant distributes some invalid shares or does not distribute part of the shares, they will be reported and deleted from the list of participants. As long as the number of dishonest participants is less than t, they cannot decrypt any secret from honest participants in the distribute phase.

In the reconstruct phase, each participant can reveal their value and share the decrypted secret shares they received. If the number of honest participants is at least t, then the pooling mechanism is successful and anyone can find all the secrets from valid secret shares, without the help of any dishonest participant. The dishonest participants cannot mislead honest participants by providing wrong decryption of secret shares in reconstruction, because the decryption for reconstruction also requires a publicly verifiable proof.

Suppose the number of dishonest participants is f. PVSS requires $f < t \leq n - f$. Therefore, we can assume that $n \geq 2 \cdot f + 1$ and let $t = \lceil n/2 \rceil$. In other words, we need to assume that more than half of the participants are honest. In a proof-of-stake use case, the set of participants for each execution are sampled from the population of miners based on their stake. If we want $n \geq 2 \cdot f + 1$ to hold with overwhelming probability, then the dishonest stake ratio should be strictly smaller than $1/2$ and preferably much

smaller. Moreover, n should not be too small. In other words, this approach assumes that most of the stake, strictly more than half and preferably much more than half, is owned by honest participants. This is in contrast to the previous approach that only needed more than $1/n$.

4.3 Further Details of the Approach

Participant Selection rules. Proof-of-stake blockchain protocols are important applications of random beacons. To prevent Sybil attacks and enforce proof-of-stake, it is common to sample a small subset of participants based on stake ratios for the random beacon. Verifiable random functions (VRF)[14] are popular for the purpose of selecting participants. VRF requires a random seed, which can be the output of RIG game in the previous round. Similar to the treatment for VDF outputs to ensure uniformly random distribution, we can also use the trick of separating the bits of the random seed r to two parts r_1 and r_2 and using $r_1 + VRF(r_2)$ instead of $VRF(r)$.

Sorting rules. In contrast to RANDAO and many other random number generators, our RIG game is sensitive to the order of participants. The result of the RIG game is not only the output value, which is the sum of all valid values submitted by the participants, but also the payoffs. The honest participants who reveal their values faithfully might receive different rewards/penalties depending on the ordering of participants. As before, we can use the output of the previous RIG round to generate a random ordering for the current round. Finally, the RIG game requires an even number of participants, so if the number of valid participants is odd, we will remove one participant arbitrarily. To make sure this does not have an effect on consensus, we can remove the participant for whom $h(\texttt{commit message})$ has the largest value.

Design of the Incentives. Every participant puts down a deposit d at the same time they send their commit message. The value of d is fixed by the protocol. After collecting all the valid values s_i and ordering of the participants P_i, $i \in [1, n]$, the result has the format $(s, \{P_i, u_i\})$, where u_i is the payoff of participant P_i. The values s, s_i are in $\{0, 1, \ldots, m\}$, where m is a parameter of RIG game, i.e. the number of strategies for each player. The output random number is computed as $s = \sum_{i=1}^{n} s_i \pmod m$. Note that all dishonest players are excluded from the sum. If a player does not release their value or otherwise cheats, then they will be punished by confiscating their deposit and they will not be included in the game. Each honest participant P_i receives a payoff of the form $r_i = u_i(s_1, \ldots, s_n) + c$. Recall that $u_i(s_1, \ldots, s_n)$ is the payoff of P_i defined by the game matrix, which sums up to 0 among valid participants. The number c is a constant defined by the specific application. Generally, c should be positive and high enough to motivate honest participants to join the RIG game and perform its steps. When we require the participants to use blockchain transactions for communication, c should at least cover the transaction fees. The deposit amount d should also be larger than any reward that a participant can possibly obtain in the game in order to discourage dishonest behavior.

4.4 Assumptions and Limits to Applicability

Network Assumptions. The most important assumptions are the network assumptions. Our RIG game relies on a synchronous communication network. All real-world blockchain networks use the internet and are effectively synchronous.

δ-synchrony. A broadcast network is δ-synchronous if a message broadcasted by some user at time t will be received by all other users by time $t + \delta$.

When applied to blockchains, blockchain consensus protocols guarantee public ledgers with different levels of synchrony. In this paper, we rely on blockchain consensus protocols to achieve a synchronized view of RIG execution. In detail, we require Δ-synchrony for blockchains:

Δ-synchronous blockchains. A blockchain is Δ-synchronous if any valid transaction broadcasted by some user at time t will become part of the stable chain at time $t + \Delta$ in the view of all honest nodes.

We assume that Δ is known to all nodes and use it to design the duration of commitment scheme approach and PVSS approach. Specifically, in the commitment scheme approach we must have: (i) $T_{commit}, T_{reveal} > \Delta$; and (ii) $T_{wait} > T_{Eval}$. After the reveal phase ends, each participant requires extra T_{wait} time slots to compute the final output. Similarly, in the PVSS approach we must have $T_{prepare}, T_{distribute}, T_{reconstruct} > \Delta$.

If the PVSS approach is implemented using off-chain communication, then we can derive lower bounds for the durations in terms of δ. In any case, the approach will not work if the network/blockchain is not synchronous or if the time limits are too tight and messages are not guaranteed to be delivered before the beginning of the next phase.

Rationality Assumption. We proved that any rational party or parties, i.e. a party/parties interested only in maximizing their own payoff, would play uniformly at random in an RIG game and would therefore be reliable. This is because playing uniformly at random is the only alliance-resistant equilibrium in the game. Moreover, the uniformity of the output random number depends on having at least one reliable player. Therefore, we must assume that at least one player is rational and our approach would not work if none of the players are rational. However, this case is unlikely to happen in practice as we would normally expect all parties to be rational.

5 RIG in Proof of Stake Protocols

We now show how our RIG random beacon can supplant standard PoS protocols. In general, the RIG random beacon, be it implemented by the commitment scheme approach or the PVSS approach, is applicable to any PoS protocol that requires an evolving random seed to select miners. Overall, using our RIG as the source of randomness only introduces negligible overhead in terms of transaction throughput.

If we use the RIG random beacon for generating the random seed in a proof-of-stake protocol, a single execution of the RIG game updates the random seed once. Usually, a single execution spans an epoch, which consists of multiple slots where the same random seed is repeatedly used. The blockchain protocol is modified to consecutively run the RIG random beacon to update the random seed in every epoch. The participants of RIG random beacon of each epoch are randomly selected, e.g. based on the RIG result of the previous epoch. Note that our approach can also be applied for every block, instead of every epoch, but this would require more communication complexity.

5.1 RIG in Ouroboros Praos

Ouroboros Praos is the second proof-of-stake protocol in the Ouroboros family and the underlying protocol of Cardano cryptocurrency [9]. Ouroboros Praos assumes a Δ-

semisynchronous broadcast network, where Δ is a finite bound on the delay of messages unknown to the nodes of the blockchain. Ouroboros Praos also assumes that the adversarial stake ratio is strictly smaller than 50%. Ouroboros Praos satisfies persistence with parameter k and liveness with parameter $u = 8 \cdot k/(1+\epsilon)$ with high probability [9]. Thus, our $8 \cdot k/(1+\epsilon)$-synchronous blockchain assumption holds with high probability.

The selection rule for random seed generation participants in Ouroboros Praos is based on a VRF. The random beacon concatenates the VRF output of the participants and applies a random oracle hash function on the concatenated output. Each participant is also a miner and announces their VRF output along with their new block. The generated random seed is used in the next epoch, which consists of a number of slots. The protocol waits for enough slots until the seed generation is synchronized among all participants for the next epoch.

We can substitute the random beacon of Ouroboros Praos with our RIG. For example, if we use the commitment scheme approach, we can reuse the VRF selection rule and epoch/slot timing system. The major difference is that the execution of RIG consists of two phases of communication. Therefore, it requires $T_{reveal} + T_{wait}$ more slots within an epoch to reach a consensus on the result of RIG. Besides, we improve the VRF selection using the split randomness trick to ensure uniformly random sampling, instead of pseudo-random sampling. In Cardano, 1 epoch lasts for 5 days, and the transaction confirmation time is 20 minutes. When using the commitment scheme, we require more time ($<= 3\times$ transaction confirmation time) within an epoch for the extra reveal phase and VDF computation time to reach a consensus on the result of RIG, which is negligible.

5.2 RIG in Algorand

Algorand [11] executes a byzantine agreement protocol for every block to ensure persistence with parameter 1 and liveness within 1 minute, assuming δ-synchrony for 95% honest users. The no-fork and one-minute-liveness properties guarantee a one-minute-synchronous blockchain for RIG random beacon. Algorand assumes that honest users hold more than two-thirds of the total stake, which satisfies the requirement of the PVSS implementation of RIG.

Algorand requires an evolving random seed for VRF-based selection of miners and Byzantine agreement protocols. The random seed is updated every R rounds by applying the VRF of the current miner to the previous random seed and the current epoch number. This is again pseudo-random and there is no guarantee that the output is uniformly distributed. We can use our RIG random beacon to generate the random seeds for Algorand. If we use on-chain communication in the PVSS approach, then we have to select the participants of RIG separately from the committee for each single round, because we want the participants of RIG to be active for multiple slots ($T_{prepare} + T_{distribute} + T_{reconstruct}$ slots) without obstructing the growth of blockchain. Each single execution of the RIG spans the duration of an epoch, which consists of R rounds. Algorand reaches consensus within 1 round, and updates the random seed once every 1000 rounds, which is sufficient for a PVSS execution. Moreover, assuming that $R = 1000$ and 100 participants join the RIG game, using RIG decreases the transaction-per-second by less than 1%.

The execution of the RIG random beacon is parallel to other parts of Algorand. While other parts require an evolving random seed from the RIG random beacon, they

can reuse the previous random seed until a new seed is computed and synchronized. Compared to the random seed updating procedure in Algorand, our RIG random beacon is bias-resistant and rules out the possibility of adversarial manipulation on the random seed assuming the selected participants satisfy the assumption of honest majority.

6 Conclusion

We presented a game-theoretic beacon for distributed random number generation. We showed that our approach is bias-resistant, unpredictable, available, verifiable and incentivizes every participant to be reliable, i.e. provide a truly uniform random input. Even if only one of the participants is rational and reliable, the output number is guaranteed to be unbiased. Additionally, our approach does not use pseudo-randomness at any point and instead only relies on well-incentivized game-theoretic randomness.

References

1. RANDAO: A DAO working as RNG of Ethereum (2019), https://github.com/randao/randao
2. Cambridge bitcoin electricity consumption index (2022), https://ccaf.io/cbeci/index
3. Arnosti, N., Weinberg, S.M.: Bitcoin: A natural oligopoly. Management Science pp. 4755–4771 (2022)
4. Aumann, R.J.: 16. Acceptable Points in General Cooperative n-Person Games, pp. 287–324. Princeton University Press (2016)
5. Bernheim, B., Peleg, B., Whinston, M.D.: Coalition-proof nash equilibria i. concepts. Journal of Economic Theory pp. 1–12 (1987)
6. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO. pp. 757–788 (2018)
7. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: OSDI. pp. 173–186 (1999)
8. Chatterjee, K., Goharshady, A.K., Pourdamghani, A.: Probabilistic smart contracts: Secure randomness on the blockchain. In: ICBC. pp. 403–412 (2019)
9. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: CRYPTO. pp. 66–98 (2018)
10. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: PKC. pp. 416–431 (2005)
11. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling Byzantine agreements for cryptocurrencies. In: SOSP. pp. 51–68 (2017)
12. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO. pp. 357–388 (2017)
13. Krasnoselskii, M., Melnikov, G., Yanovich, Y.: No-dealer: Byzantine fault-tolerant random number generator. In: INFOCOM. pp. 568–573 (2020)
14. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: FOCS. pp. 120–130 (1999)
15. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review p. 21260 (2008)
16. Nash, J.: Non-cooperative games. Annals of Mathematics pp. 286–95 (1951)
17. Roughgarden, T., Nisan, N.: Algorithmic Game Theory. Cambridge University Press (2007)
18. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.R.: Hydrand: Efficient continuous distributed randomness. In: SP. pp. 73–89 (2020)
19. Schoenmakers, B.: A simple publicly verifiable secret sharing scheme and its application to electronic voting. In: CRYPTO. pp. 148–164 (1999)
20. Syta, E., Jovanovic, P., Kokoris-Kogias, E., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: SP. pp. 444–460 (2017)
21. Wang, G., Nixon, M.: Randchain: Practical scalable decentralized randomness attested by blockchain. In: IEEE Blockchain. pp. 442–449 (2020)