A Review on Generating Random Numbers in Decentralised Environments

Srđan Daniel Simić, Robert Šajina, Nikola Tanković, Darko Etinger
Faculty of Informatics
Juraj Dobrila University of Pula
Pula, Croatia
{ssimic, robert.sajina, nikola.tankovic, darko.etinger}@unipu.hr

***Abstract*—Recent advances in blockchain gained significant social attention, mainly due to substantial price fluctuations of Bitcoin and Ethereum cryptocurrencies. By its design, blockchain is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way, providing solutions for many complex tasks without third party involvement. To achieve that, they employ a set of Byzantine Fault-tolerant consensus algorithms that require the implemented logic to be deterministic. The lacking source of randomness is a consequential limitation since many application domains, like games, lotteries, or random elections, require random sources. Given the Byzantine Fault-tolerance, generating random numbers should also be publicly-verifiable and tamper-resistant, but still hold the premises of being unpredictable.**

In this paper, we will provide an overview of the current research surrounding pseudo-random number generation on a decentralized network that satisfies those requirements.

***Index Terms*—Random number generation, Blockchain**

I. Introduction

Publicly-verified randomness is probably the most fundamental technical problem on a decentralized network. Its applications range from random elections, lottery games, to securing the network itself. Thus, a careful investigation into scalability and the security of the available solutions is crucial. The process of generating random numbers from an entropy source is called a Random Number Generator - RNG. The security of the system corresponds to the degree of the quality of the entropy source. There are two main requirements for the entropy source: **RQ1** - uniform distribution, **RQ2** - unpredictability of the result.

Random beacons like the NIST beacon [14] use the hardware entropy sources based on the different quantum-mechanical phenomena. These beacons are centrally governed and require an ongoing trust towards their publishers.

When the sources for random numbers are applied to the decentralized systems, additional requirements arise: **RQ3** - the characteristic of being publicly-verifiable and **RQ4** tamper-resistant which implies that the process of generating random number should not rely on any single actor. Those additional requirements led to many solutions of distributed RNGs - dRNGs that we cover in this review.

Prooving **RQ1** requires observing the process in the long run. A well known sufficient and agreed-upon approximation for achieving that requirement are cryptographic hashes [11]. Prooving the **RQ2** requirement is much harder because the traces of predictability are manifested by a statistically unlikely series of events that demonstrate the advantage for some actors. Generally, no one actor should be able to predict or extract any information in advance. The amount of information on future generated numbers should be the same for each actor.

RQ3 ensures that the generated random number should be verifiable by each actor - correctness. Because of a possible conflict of interest, actors must be able to verify if the received number is indeed the correct one and generated by a specific predetermined procedure.

We analysed the current state of the art and identified several unique ways in ensuring these requirements where most of them are based on a commit-and-reveal scheme and use public distributed ledgers.

II. Methodology

Studies included in this review are the most recent ones which tackled the problem of random number generation given restrictions of decentralised environment. The goal for this review is to enumerate the proposed techniques in decentralized random number generation and to answer the following question about each approach:

- Who are the actors in the proposed method?
- Which algorithmic techniques were used?
- What are the main characteristics of the algorithm in use?
- What is the most appropriate domain of use?

To identify relevant studies we queried Scopus database in March 2020 using **blockchain**,**random beacon**,**random number** keywords ,after which we iterated and converged on a relevant search query that captured all of the relevant aspects and used forward snowballing technique. After reading the abstracts, we selected the promising publications and found the final set of publications included in this short review.

III. Background

In this section, we give a brief overview of a set of methods that were applied in different stages of proposed approaches in order to satisfy the requirements **RQ1-4**.

Publicly verifiable secret sharing. Verifiable secret sharing scheme has the property that the validity of the shares distributed by the dealer can be verified by any party [17]. It is based on classic secret sharing scheme which distinguishes two protocol phases: *distribution* phase where a secret is

distributed (splitted) by a dealer among the participants, and a *reconstruction* phase in which the secret is recovered by fetching the shares, usually in a (t, n) scheme where at least t out of n participants can together reconstruct the secret. Verifiable secret sharing (VSS) resists malicious players, including the dealer itself. Public verification enables that any third party can verify that participants received correct shares.

Collective Signing - CoSi [18] is a scheme based on a variation of well-known Schnorr multi-signature algorithm [10]. Threshold model (t, n) is applied where at least t out of n participants are required in order to decipher the secret message. The secrets are usually shared via the aforementioned PVSS system so that an honest threshold of users can later recover them and form a publicly verifiable proof of their validity.

CoSi is used in [19] for *RandHound*, *RandShare* and *RandHerd* algorithms. RandHound uses a commit-then-reveal approach implemented via PVSS and CoSi to prevent client equivocation (explanation needed). The requirement is that at least f of $3f+1$ contributors are honest. RandHound is similar to RandShare, except that contributors are divided in smaller groups to enable scalability.

Homomorphic encryption (HE) is a kind of encryption scheme that allows a third party to perform certain computable functions on the encrypted data while preserving the features of the function and format of the encrypted data. An HE is primarily characterized by four operations : *KeyGen*, *Enc*, *Dec*, and *Eval*. While *KeyGen*, *Enc* and *Dec* are no different from their classical tasks in conventional encryption schemes, *Eval* is HE specific operation which takes ciphertexts as input, performs the function $f()$ over ciphertexts without seeing the messages and outputs evaluated ciphertexts. The most crucial point in HE is that the format of the ciphertexts after an evaluation process must be preserved in order to be decrypted correctly. To support an unlimited number of operations the size of the ciphertext should also be constant otherwise the increase in the ciphertext size will require more resources [1]

- Partially homomorphic encryption - allows only one type of operation with an unlimited number of times [1]
  - RSA- only homomorphic over multiplication
  - ElGamal- multiplicatively homomorphic
- Somewhat homomorphic encryption - allows some types of operations a limited number of times [1]
- Fully homomorphic encryption - allows an unlimited number of evaluation operations on the encrypted data and resulting output is within the ciphertext space [1]

Zero-knowledge proofs. ZK-proof [7] is a scheme in which a *prover* can prove to any interested party (*verifier*) that he knows a secret value x without revealing any information on the x, except to the fact that he knows the value. It comes in the interactive flavor where messages are exchanged between prover and verifier, or in non-interactive flavor where no such interaction is necessary, but requires a bit-string to be shared between prover and verifier that is **guaranteed** to be random.

Verifiable random function (VRF). VRF [12] is a pseudo-random function that given an input x and a secret key (SK) produces a pseudo-random number $f_{SK}(x) = y$ together with a non-interactive publicly verifiable proof π_y for y so that everyone with the knowledge of π_y and the public key (PK) can verify whether y was generated correctly, but still cannot recover SK. VRFs can be regarded as an asymmetric version of keyed hash functions. They are useful for mapping low entropy inputs ($x \in X$ where $H(X)$ is low) in order to provide preimage resistance: it is difficult to find x such that $h(x) = y$. Ordinary cryptography hashes and encryption are not preimage resistant.

BLS signature/scheme. BLS signature uses a group of *N* members that collectively hold private key *S*, with each member only holding a part of that key denoted s_i. Public key corresponding to *S* is marked as *P* and messages signed by *S* are marked as *SIG*. In the process of signing, each member signs a message with their key s_i resulting in sig_i. *SIG* is calculated only after receiving *k* signatures [15]. BLS scheme utilizes these properties by dividing nodes in the network into h groups. A member of each group produces and releases its share for a BLS signature of the message (e.g., a round number), sharing it with other members of the group. Each member of a given group waits until a threshold number *k* of BLS signature shares are available for that group and then forms the BLS signature SIG_i for this group. The first member who collected *k* shares announces or gossips these signatures with members of other groups. The next group is chosen based on SIG_1. This chosen group repeats the same process as the first group but uses SIG_1 as a signing message. Process of combining this groups signatures continues until a global BLS signature is available [15] [19] [16]. Only the last group can manipulate SIG_h if the group has more than *k* malicious members, although it is not predictable which group will do the SIG_h generation task [15].

IV. Commit and reveal approaches

These approaches are based on generating a random number from a series of contributions from multiple agents. Multiple agents are used to satisfy **RQ4**. The base assumption here is pretty realistic: at least one agent must be honest and not in malicious coordination with other agents. If that is the case, **RQ2** is also satisfied. The protocol is divided into time epochs where each epoch generates a single random number. Each epoch is further divided into two main phases: (1) the collection phase, where all the contributions are gathered, but hidden in a way that no one agent can gain insight from them, and (2) the reveal phase in which the agents have to reveal their contribution to others satisfying **RQ3**. There are two essential rules to adhere to: the second phase does not start until the first phase is finished, and all the contributions must be revealed in the second phase for the epoch to be considered valid.

A known challenge in the commit-reveal approach is that the last agent to reveal its commitment has an unfair advantage of knowing the final output before revealing their secret and thus can withhold revealing if it considers for that outcome

not to be favorable. This can be regarded as a form of Denial of Service attack, and there are separate agreements on techniques on how to enforce the final revealing. In this chapter, we will see different methods authors used to tackle that challenge.

A. Dealing with non-reveals

To address the flaw in a commit reveal scheme authors [20] proposed protocol where they provided means of hiding the randomness output from any subset of participants that are not abiding by the rules of the protocol while also maintaining the public verifiability which is necessary for randomness source. Their protocol has five steps where a new step can only be started when all the participants have finished the previous one: (1) Commitment, (2) Revealing, (3) Distribution, (4) Validation, (5) Reconstruction. To ensure that no participant has an unfair advantage in (2), (1) was designed so that each participant computes $y_i = g^{x_i}$, where $x_i \in Z_q$ is their random value and g is an element of a cyclic multiplicative group G, and together with another random value r_i with at least $log_2(||Z_q||)$ random bits calculates a secure cryptographic hash function $H_i = H(y_i, r_i)$ which is then recorded on a blockchain. Authors argue even though $y = g^x$ is publicly known, nobody can extract x unless he is capable of calculating the discrete logarithm in G, which is assumed to be a hard problem. The last step in (2) is extracting randomness R from blocks of the blockchain. Authors state that this step stops any attacker, who does not control at least k participants, from manipulating the block randomness in their favor. In step (3), participants distribute random value x from a uniform distribution using verifiable secret sharing protocol. In (4) each participant P_t decrypts and verifies $s_{i,t}$ of others P_i from (3) using predefined equation. If the equation results in non-equality, that will imply that P_i had been cheating. Finally in (5) all participants submit their decrypted $s_{i,t}$ and by solving the system of linear equations $s_{i,t}$ any party can calculate values of x_i and therefore $x = \sum_{i=1}^{n} x_i$. Public randomness r is finally achieved with randomness extraction function (secure cryptographic hash algorithm), which processes the value x concatenated with R, to ensure **RQ1**. Some of the possible security issues in this approach are 51% attack and having k corrupted participants.

B. Usage withing lottery-alike games

Authors in [9] addressed the issue of unfairness in lottery games and have proposed a blockchain-based smart contract for the fair and efficient lottery. Their approach is based on a commit-reveal scheme that can be described in 6 phases: (1) Deploy, (2) Register, (3) Purchase, (4) Publish, (5) Verify, (6) Award. Phase (1) starts with lottery center generating two pair of keys (LC_k and PL_k public and private) after which they deploy smart contract, which contains ticket value of users and specified $EndTime$, signed with their private LC_k while also providing their public LC_k to protect the confidentiality of communication. In (2) players generate their key pair PL_k for which they receive an address $PL_{address}$ from the blockchain platform. Players buy their tickets in phase (3). The ticket contains ticket value and identification code (v_i, s_i) chosen by the player. For ticket to be valid players have to send $Hash(v_i, s_i)$ to the blockchain before $EndTime_{buy}$, which will generate list of unverified transactions, after which they have to submit $LC_{P_k}(v_i, s_i)$ before $EndTime_{open}$ so that lottery center, using their private LC_k, can verify if the (v_i, s_i) matches the value submitted before $EndTime_{buy}$. Phase (4) is what distinguishes it from other research. To generate a winning number, authors have chosen to use Barycentric Lagrange Interpolation, which is an improvement of Lagrange Interpolation. Lottery center treats all $(Hash(v_i, s_i), v_i)$, submitted by the players, as points (x_i, y_i) that are used to get a polynomial $f(x)$ which goes through all of them. Authors argue that the coefficient of the polynomial is unpredictable and "truly random", therefore ensuring **RQ2** . If a player suspects that a winning number is false, he can verify it in (5) by validating $f(x_i) = y_i$, which provides answer to **RQ3**. The last step is (6) where the winner provides his $PL_{address}$ and (v_i, s_i) to the lottery center which verifies it and rewards the player accordingly. Authors conducted a security analysis of their approach that addressed several attacks on the system, along with solutions for them.

Authors in [5] addressed the problem of data manipulation by the game providers and designed algorithms to prevent human manipulation, which allows all the participants to be involved in the generation of random numbers. It has three actors: players, game providers (GP), and a blockchain. The algorithm can be summarized in a few steps:

- Process starts when GP creates a random number with a public-private key pair and records it on a blockchain
- Players generate and send their own random numbers to GP (**RQ02**)
- GP generates a random number according to a pre-announced algorithm by using players number, chain random number and its own random number
- GP uses this newly generated random number to start a game
- When the game is over GP announces its number and private key which players can use to verify if the random number is correct or if there was cheating involved satisfying **RQ03**

The algorithm was deployed on a consortium chain-based game platform with mixed Byzantine fault tolerance (MBFT) consensus algorithm. Within their experimental environment, the authors concluded that the latency between the player sending its random number and game provider sending response is approximately equal to the load time that a player has to wait to start a game. When there are more than 2000 players, there is a need for more than one consensus group because latency doubles with an increasing number of players.

C. Using beacons

Authors in [8] proposed a protocol used inside beacon that combines properties of blockchain and timed commitment, thus making the probability of adaptive attacks negligible

while also providing a quickly verifiable result. The main advantage of their protocol is in their verifiable timed commitment algorithm. By adapting the flaw of timed commitment method where adversaries can hinder the protocol with false commitments, authors offered a solution in which the commitment can be guaranteed to open correctly by verifying the commitment before accepting a block, including the one. Their protocol can be divided into five steps:

1) Commit
2) Verify commitment
3) Open commitment
4) Verify result
5) Generate public randomness

The evaluation was done on availability, unpredictability, impartiality, and verifiability of their protocol. Chain growth, common prefix and chain quality properties of blockchain are guaranteeing availability, unpredictability (**RQ2**) and impartiality, while verifiability (**RQ3**) has been ensured with verifiable timed commitment.

In [4], the authors propose an approach with participants who send their inputs to the beacon operator. Participants send their inputs combined with all data required for the computation named commitment, **RQ4**. Such a technique improves transparency by allowing any party to compute this randomness alongside the beacon operator, **RQ3**. To decrease the possibility of the operator manipulating the results by trying different commitments before releasing them, the operator must run a verifiable delay function, which ensures that the operator cannot try more than one commitment before running out of time, **RQ4**. The adversary can not bias the outcome as long as there is at least one honest party. As the authors state, there are two options for implementing this approach: (1) using a smart contract as a beacon operator, (2) beacon operator runs separately, but publishes some of its artifacts on the blockchain. While the first solution offers benefits in terms of decentralization, each computation executed on a smart contract consumes virtual currency, requiring many users to compensate for the high on-chain computation cost. Authors further break down the second option in four options, each providing different trade-offs of decentralization, security, and costs.

D. Using homomorphic encryption

Authors of [13] proposed a protocol composed of three main components: Requester, Core Layer (consists of many parties responsible for PRNG), and Public Distributed Ledger (PDL). Their protocol works in rounds where each round consists of a few stages. To initialize the process, Requester sends a $nonce$ together with his PK to the PDL for what he receives the ticket T, which is used to determined which parties, called contributors, are eligible to contribute to generating a random number. For each T, an instance of verifiable random function (VRF) had to be executed by Core Layer's parties, after which proof-of-eligibility (PoE) has to be sent to PDL that verifies all received PoE. Each contributor, which is a member of the eligible party, encodes his contribution, chosen value M_i, with Requester's PK, and sign it with their private key after which proof-of-contribution is generated. After all contributions have been made public, they are summed and sent to the Requester, which is possible because the protocol employs El-Gamal encryption. At the final stage, Requester uses his secret key to decrypt the tallied contribution and publishes decrypted value M along with a proof of proper decryption, **RQ3**. Security and complexity analysis has been conducted in which authors showed that their protocol could be implemented for practical use, after which they compared it with other possible solutions on dRNG market. The authors also noted the limitation of their protocol, which is an assumption that the Requester does not share his private key with anyone else, negating **RQ4**.

E. Game-theoretic approaches

In [2] authors proposed a commit-reveal approach based on a random bit generation game (RBG) and implemented their algorithm using a single-instance smart contract called Random Bit Generator Contract (RBGC). In the RBG game, each player is represented as a number and can be either an even-numbered or odd-numbered player. Even-numbered players can choose between playing either 0 or 2, while odd-numbered players can play either 1 or 3. A mini-game is simulated for each *(i, j)* pair where *i* and *j* are of different parity's. The winner is the player with its chosen number greater by one than that of its opponent. This way, players are incentivized to play uniformly at random to maximize their earnings, enforcing **RQ1**. The RBGC contract supports multiple requests for a random (in experiments 176 bits/s) with requesting clients specifying random bit generation due time. This approach produces random numbers as long as one of the participants is honest. The client who requested the first random bit would receive its bit only if all precipitants revealed its bits truthfully. The RGBC contract will fail to deliver random bit in cases where the deadline was to close (time available for generating the random bit is less than a predefined constant t_{min}), there were no participants or some participants failed to reveal its bit. In the latter case, the RGBC contract refunds the client with confiscated deposits. Finally, in the case of the successful reveal of all participants, a random bit is generated as XOR of all bits. Authors argue that generating a multi-bit random number can be achieved by asking participants to submit an N-bit number and running an independent RGB game for each bit. Looking at the implementation code provided by authors, we noticed that running multiple independent RBG games for each bit would be imposed on the participant in the form of transaction execution costs - *gas*.

V. Pure DLT approaches

One of the first approaches in the area of blockchain-based smart contracts used to solve the pseudo-random number generator problem was to use the information on the ledger itself as the source of entropy, hence the name of this section Pure Distributed Ledge Technology approaches (Pure DLT approaches). In the proof-of-work secured ledgers, the primary

source was the computed block *nonce* which comes from a high entropy uniform distribution computed with a delay for securing the ledger itself. Therefor using proof-of-work as primary source of entropy satisfy **RQ1**, **RQ2** and **RQ3**.

There are two main branches in this approach: (1) using the information from previous blocks and (2) using the information from yet unseen future blocks. In the cases where additional security is required, just using the block nonce is not sufficient, because of possible attacks on a blockchain network itself, so additional sources of entropy are required. In this chapter, we will explore how authors used data from blocks spiced with some additional data to produce a reliable pseudo-random number generator.

A. Previous blocks

In this subsection, we will review [3] in which authors ensured **RQ4** by utilizing data from previous blocks of a blockchain paired with cryptocurrencies price information to produce one-dimensional Cellular Automaton. To create a seed, which will be used to generate a one-dimensional CA, the authors described the formation of RngBlocks. These blocks V_{t_i} consists of sequence of bits of length k, timestamp t_i that denotes when the RngBlock was created and set of statistically independent evidence $E = \{e_1, e_2, ..., e_n\}$. That evidence is data extracted from previous blocks of a blockchain, and they must be represented in a binary form. Authors aligned the population of RngBlocks with the evolution of a blockchain, which is considered to be a continuous sequence of blocks, in a bounded time window $[L, A]$ of block stream S that contains $(V_{t_i}, block.hash_i)$ pairs. The generation of the genesis RngBlock $(V_{t_n}, block.hash_n)$, which is used as the initial seed for 1-d CA creation, is triggered by a self-executing smart contract, with a deployment date L, in a specific point in time A when all execution conditions are met. Configuration of proposed CA, which consists of a Boolean cellular array with $s \in \{0, 1\}$ states, is as follows : initial seed $(V_{t_n}, block.hash_n)$, length of the CA, which is the same as the RngBlock length, transition rules, which are specified in the form of a 3-bit rule table with each entry capturing every possible neighborhood state, and the time of execution q, which sets the finite number of evolution steps for CA. Authors defined q as a decimal form of the last z digits converted from a specific hexadecimal value that has been gathered from a block's hash which is determined by *blockhash(block.number - offset)* where *block.number* is the height of the current block and *offset* is related to Ethereum's price fluctuation normalized within a domain $[0 - 255]$. In each evolutionary step using the genesis RngBlock as the initial seed, their proposed methodology instantiates a CA, while the selection of transition rule, which contributes to the next evolution state of the CA, is set according to binary representation of a normalized integer value that resides within the domain [0-255] derived by $y = (255 * ms)/1000$ where ms is milliseconds collected from A. When the number of steps reaches q, a binary number of length k is produced as the final outcome of the pseudo-random generator process. The authors conducted an experimental evaluation using *Dieharder* and *NIST* statistical test suits that yielded positive results.

B. Future blocks

In [6], the authors defined their protocol with four participants: users, service system, random number generator, and blockchain. The random number generator is implemented as a smart contract and can execute two kinds of queries: Reg, which registers the received data on the blockchain, and $AskNonce$, which is used to obtain the nonce of the future block. Authors implemented their protocol inside a lottery game, and the process for obtaining the winning number can be described as a two-phase process: user and service system registration and lottery phase. During the user and service registration phase, each participant sends their seed and address to the random number generator smart contract G, to ensure **RQ4**. The registration phase must be executed at least once before the lottery phase can begin. During lottery phase G makes identification number sn and executes $Reg(sn, a, b)$, where a is user's and b system service address, after which $AskNonce$ would be executed to get nonce n of the next block. Last step is computing pseudo-random number α by calculating a hash function of sn, n, a, b , r and p, where r and p are user's and system service seed. The authors noted that their protocol has performance and scalability problems that will be addressed in their future work.

VI. Conclusion

Generating pseudo-random numbers on a decentralized network is a non-trivial task. Lack of entropy and transparent ways to secure the data so that no malicious party can manipulate or gain unwanted knowledge from it is still a major bottleneck in that process.

In this paper, we present some of the recent attempts in generating a pseudo-random number on a decentralized network, i.e., blockchain. Although, during our research, we encountered various approaches, based on volume of relevant research we concluded that at the moment, the most promising, and also the most prominent approach for solving this particular problem, is by using a commit reveal scheme.

References

[1] Abbas Acar et al. "A Survey on Homomorphic Encryption Schemes". In: *ACM Computing Surveys* 51.4 (2018), pp. 1–35. ISSN: 03600300. DOI: 10.1145/3214303.

[2] Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Arash Pourdamghani. "Probabilistic smart contracts: Secure randomness on the blockchain". In: *ICBC 2019 - IEEE International Conference on Blockchain and Cryptocurrency* (2019), pp. 403–412. DOI: 10.1109/BLOC.2019.8751326. arXiv: 1902.07986.

[3] Klitos Christodoulou et al. "Randomblocks: A transparent, verifiable blockchain-based system for random numbers". In: *Journal of Cellular Automata* 14.5-6 (2019), pp. 335–349. ISSN: 15575977. DOI: 10.13140/RG.2.2.21366.96328.

[4] Samvid Dharanikot et al. “Breeding Unicorns: Developing Trustworthy and Scalable Randomness Beacons”. In: (2020), pp. 99–106. DOI: 10.1109/blockchain.2019.00022.

[5] Mingxiao Du et al. “A blockchain-based random number generation algorithm and the application in blockchain games”. In: *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics* 2019-Octob (2019), pp. 3498–3503. ISSN: 1062922X. DOI: 10.1109/SMC.2019.8914618.

[6] Yuto Ehara and Mitsuru Tada. “How to generate transparent random numbers using blockchain”. In: *Proceedings of 2018 International Symposium on Information Theory and Its Applications, ISITA 2018* C (2019), pp. 169–173. DOI: 10.23919/ISITA.2018.8664239.

[7] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “Knowledge complexity of interactive proof systems”. In: *SIAM Journal on Computing* (1989). ISSN: 00975397. DOI: 10.1137/0218012.

[8] Yanxue Jia and Lei Fan. “Generate public randomness based on blockchain”. In: *Proceedings - 2018 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovations, SmartWorld/UIC/ATC/ScalCom/CBDCo* (2018), pp. 263–270. DOI: 10.1109/SmartWorld.2018.00080.

[9] Jiasheng Li, Zijian Zhang, and Meng Li. “BanFEL: A Blockchain Based Smart Contract for Fair and Efficient Lottery Scheme”. In: *2019 IEEE Conference on Dependable and Secure Computing, DSC 2019 - Proceedings* (2019), pp. 1–8. DOI: 10.1109/DSC47296.2019.8937559.

[10] Gregory Maxwell et al. “Simple Schnorr multi-signatures with applications to Bitcoin”. In: *Designs, Codes, and Cryptography* 87.9 (2019), pp. 2139–2164. ISSN: 15737586. DOI: 10.1007/s10623-019-00608-x.

[11] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. USA: CRC Press, Inc., 1996. ISBN: 0849385237.

[12] Silvio Micali, Michael Rabin, and Salil Vadhan. “Verifiable random functions”. In: *Annual Symposium on Foundations of Computer Science - Proceedings* (1999), pp. 120–130. ISSN: 02725428. DOI: 10.1109/sffcs.1999.814584.

[13] Thanh Nguyen-Van et al. “Scalable Distributed Random Number Generation Based on Homomorphic Encryption”. In: October (2020), pp. 572–579. DOI: 10.1109/blockchain.2019.00083.

[14] NIST. *Interoperable Randomness Beacons*. 2019 (accessed January, 2020). URL: https://csrc.nist.gov/projects/interoperable-randomness-beacons.

[15] randao.org. “Randao: Verifiable Random Number Generation”. In: 2017. URL: https://www.randao.org/whitepaper/Randao_v0.85_en.pdf.

[16] Philipp Schindler et al. “HydRand: Practical Continuous Distributed Randomness.” In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 319. URL: https://eprint.iacr.org/2018/319.pdf.

[17] Berry Schoenmakers. “A simple publicly verifiable secret sharing scheme and its application to electronic voting”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1666.i (1999), pp. 148–164. ISSN: 16113349. DOI: 10.1007/3-540-48405-1_10.

[18] Ewa Syta et al. “Keeping Authorities "honest or Bust" with Decentralized Witness Cosigning”. In: *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*. 2016. ISBN: 9781509008247. DOI: 10.1109/SP.2016.38. arXiv: 1503.08768.

[19] Ewa Syta et al. “Scalable Bias-Resistant Distributed Randomness”. In: *Proceedings - IEEE Symposium on Security and Privacy* (2017), pp. 444–460. ISSN: 10816011. DOI: 10.1109/SP.2017.45.

[20] Habib Yajam et al. “Improvement on Bitcoin’s Verifiable Public Randomness with Semi-Trusted Delegates”. In: *9th International Symposium on Telecommunication: With Emphasis on Information and Communication Technology, IST 2018* (2019), pp. 53–57. DOI: 10.1109/ISTEL.2018.8661008.