SRNG: An Efficient Decentralized Approach for Secret Random Number Generation

***Abstract*—Many blockchain protocols and applications require access to a reliable source of distributed random numbers. This has led to the recent interest in the study of distributed random number generation (RNG) and randomness beacons. Numerous approaches have been proposed in the literature, using different cryptographic techniques and working under different assumptions. A problem that has recently been studied is that of generating *secret* random numbers. There is a natural use-case for this. Suppose a casino CASSIE wishes to offer its gambling games as a smart contract. It is not viable to generate a fresh distributed random number for each bet. Instead, a secret random number should be generated at predefined intervals, e.g. each day, and used as a seed to create the randomness for the whole day. This seed should only be known to CASSIE. Moreover, at the end of the day, CASSIE should be able to disclose the seed and prove that there was no tampering.**

In this short paper, we propose a simple and novel distributed random beacon protocol that generates distributed random numbers while preserving secrecy. The generated random number can be used in DeFi applications, such as decentralized casinos, for some time, until it is published along with proof that it is indeed the output of our random beacon. In addition to achieving the desired secrecy property, our approach is also efficient and requires the same amount of computation and communication as non-secret random beacons. Our protocol can easily be implemented as a smart contract.

***Index Terms*—Random Beacon, Secret Randomness, Blockchains**

I. Introduction

Motivation for Random Beacons. Many applications taking advantage of recent advancements in blockchain rely on randomness, namely on the decentralized generation of random numbers. One notable example is Proof-of-Stake (PoS), a widely adopted energy-efficient blockchain consensus protocol. In PoS, each node is selected as the validator/miner for the next block with a probability proportional to their stake [1]–[3]. Random beacons are also important in emerging blockchain-based gambling applications, such as casinos and lotteries.

Existing Random Beacons. Due to the deterministic nature of blockchain protocols, generating random numbers is a challenging task, attracting active research in this area [4]–[9]. Some approaches use the blockchain state, such as a block hash or timestamp, as the source of distributed randomness [2]. However, these methods are susceptible to miner manipulation and give miners an unfair advantage over other participants of the network. In other approaches, all nodes or a random subset of the nodes form a committee to jointly run a distributed protocol to generate fresh random numbers. These approaches assume that one or the majority of nodes will honestly follow the protocol, and defend against the malicious nodes using cryptographic techniques such as commitment schemes [10], [11], publicly verifiable secret sharing (PVSS) [7], [12]–[15] or verifiable delay functions (VDF) [10].

Motivation for Secret Randomness. Generating a *secret* random number on the blockchain was recently considered in [16]. While previous random number generation protocols succeed in providing bias-resistant distributed randomness, using commitment schemes, PVSS, or VDFs, there has not been much research on applications that require the random number to be secret while being used. For example, consider an online casino CASSIE who wishes to provide gambling and lottery games as a smart contract. Generating a fresh random number for each bet is both too time-consuming and far too expensive, since it uses both gas and rewards to incentivize participation in the RNG. Thus, it would be much more natural to have a decentralized protocol that generates a single fresh random number at the beginning of each day and then keep this random number private to CASSIE and use it as a seed to generate further random numbers for each bet during the day. Of course, the random seed should be publicly verifiable later, at the end of the day, in order to prove to the players that the casino did not manipulate the games' results. See [16] for more on this point. While [16] provided an approach for generating secret random numbers on the blockchain, their method is inefficient and has a quadratic communication complexity, i.e. $\Theta(n^2)$ where n is the number of RNG participants. This is in contrast to non-secret decentralized RNG protocols that only require linear communication. In this short paper, we close this gap and provide simple and novel decentralized RNG protocols that ensure secrecy without compromising on security or efficiency. Our protocols have a communication complexity of $O(n)$.

Our Contributions. In this work, we propose two Secret Random Number Generation (SRNG) protocols, SRNG1 and SRNG2. Both protocols use a novel approach to generate random numbers using a public key encryption scheme, verifiable delay functions (VDFs) and commitment schemes. The former requires only one round of communication from the participants, whereas the latter needs two rounds. SRNG2 provides stronger tamper-resistance guarantees than SRNG1. Both approaches are efficient, requiring only $O(n)$ overall communication for an RNG committee of n members.

II. Preliminaries

Aggregation of multiple inputs. Previous random number generation protocols, such as [5]–[7], mostly share the same simple idea to ensure decentralization: to allow each member of a group of participants to contribute a part of randomness and use an aggregation function to combine multiple input local random numbers into a decentralized random output. Popular choices of aggregation functions include xor or modular summation. Assume the RNG protocol is designed to generate an m-bit integer v uniformly at random from $\mathcal{S} = \{0,1\}^m$. Each participant P_i of the protocol should independently sample an m-bit integer s_i uniformly at random using her own random source, for example, by rolling a fair coin m times. Then, the modular summation formula defines v as $v := \sum_{i=1}^{n} s_i \pmod{2^m}$, where n is the number of participants. Using the modular summation aggregation, the distribution of v is uniform if at least one out of the n participants honestly samples her value uniformly at random. For simplicity, sometimes we do not explicitly mention $(\bmod\ 2^m)$. We also denote 2^m by M. While the protocol looks simple at first glance, implementing it in a decentralized setting is complicated by the existence of malicious participants who violate the protocol. To ensure such parties cannot abort the protocol, we can use commitment schemes and VDFs or PVSS schemes. In this work, our protocol does not use PVSS to avoid the overall quadratic communication complexity.

Commitment schemes. One attack that a malicious participant can perform is to delay choosing her value until all other participants submit their values. Based on all other values s_i, the malicious participant can successfully bias the RNG output by carefully choosing her own value. A commitment scheme can help prevent this attack using a two-stage protocol. In the first stage, every participant sends a commitment $c_i = hash(s_i||nonce_i)$, where $hash$ is a cryptographic hash function and $nonce_i$ is a random value so that the malicious participant does not know s_i from c_i. In the second stage, everyone reveals s_i and $nonce_i$ to match c_i. A malicious participant can still try to tamper with the final output by not revealing her value in the second stage. However, this can be disincentivized by deposits and further prevented by making the final output unpredictable using verifiable delay functions (VDFs) [10].

VDF. Informally, a verifiable delay function [18] is a function f, that given an input x takes at least time t to compute the output $y = f(x)$, even if using many processors in parallel. However, a proof π can be efficiently computed along with the computation of y, such that a third party can verify whether y is indeed $f(x)$ in a much shorter time using only one processor, with the help of π. In RNG protocols using commitment schemes, a participant can be requested to provide a VDF that evaluates to her contribution to the RNG, so that in case of non-revelation, this value can be restored. The values cannot be revealed by evaluating VDF during the commitment stage, since the parameter t shall be chosen to ensure it takes a long time to evaluate the VDF.

RNG smart contracts and open protocols. On blockchains, it is common to deploy RNG protocols as a published smart contract. Firstly, the smart contract can make use of the consensus mechanism of the underlying blockchain to easily achieve synchronization. Secondly, the smart contract allows interaction with the other participants of the blockchain, which can either join the process of generating random numbers or get fresh distributed random numbers from the smart contract via function calls. The smart contract allows the RNG protocol to be an open protocol. This implies that any participant can join the RNG protocol to contribute to the decentralized community and also earn some reward. A participant should usually pay some deposit, which is required to disincentivize malicious behavior. In the rest of the paper, we call the owner of the smart contract CASSIE and refer to n contributors of randomness as participants, parties or players.

Desired Properties. In SRNG, we aim to achieve the following desired properties:

- *Secrecy*. The final random value should be known only to CASSIE. Other parties can know the value only if CASSIE reveals it to them. Secrecy must be satisfied even in case of a collusion among all players but one.
- *Tamper-resistance*. As long as at least one party is honest, no party should be able to affect the resulting random value v or its distribution, even if there is malicious activity by some of the participants (including CASSIE).
- *Auditability*. CASSIE should be able to disclose the final random number v and prove to the other participants that the disclosed value was truly generated by the protocol and not changed by CASSIE.

III. Our Approach

In this section, we present our SRNG protocols. Both protocols are efficient and secure while guaranteeing the secrecy of the generated random numbers. SRNG2 provides a stronger guarantee of tamper-resistance than SRNG1.

A. SRNG1

An instance of the SRNG1 protocol generating one secret distributed random number consists of the following steps. See 1 for an illustration.

Step 1. Initialization. CASSIE calls the `initial` function of the smart contract, asking for the generation of a new random integer in the range $\{0, 1, 2, \ldots, 2^m - 1\}$. Recall that $M := 2^m$. She should specify the amount of reward R that she is willing to pay to the players, and the amount of deposit d_c that she will lose if she is found to have violated the protocol later. In total, CASSIE deposits $R + d_c$ with the contract. Additionally, CASSIE generates a new public/private key pair (pk, sk). She keeps sk secret but submits pk to the contract, so that it is public and visible for the participants who can send their contributions s_i in encrypted form. Finally, CASSIE selects a VDF with evaluation time t_{VDF}. CASSIE sets a time limit t_{submit} that is significantly smaller than the VDF evaluation time t_{VDF}. Every participant must send her value in time t_{submit}. For example, a real-world casino might set t_{submit} to be 10 minutes and t_{VDF} as several hours.

Step 2. Value Submission. Every participant of the blockchain network can choose a random value s_i and a random nonce $nonce_i$. Participant P_i sends the encrypted $c_i = Enc_{pk}(s_i)$ by calling the smart contract's `submit` function. The smart

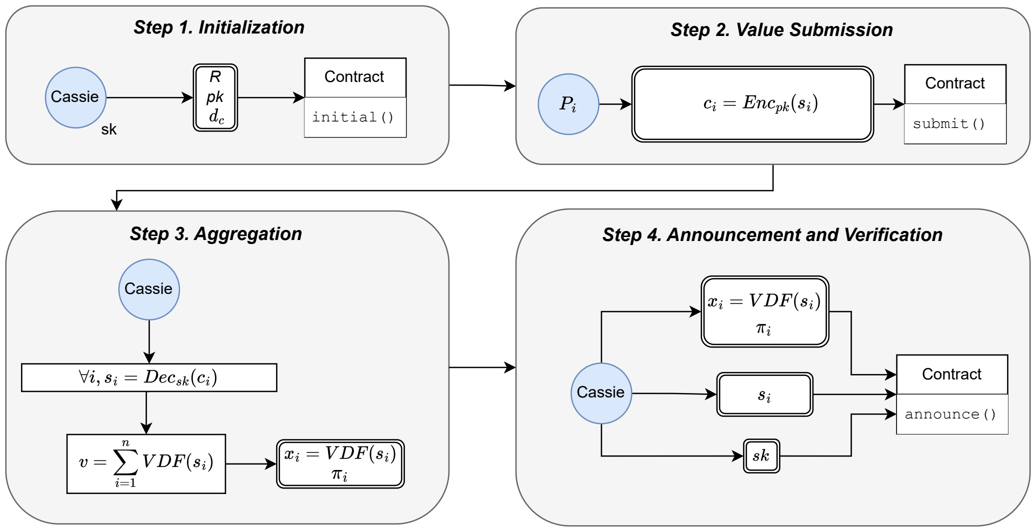

Fig. 1. Illustration of the steps in our SRNG1 protocol.

contract also keeps track of id_i, i.e. P_i's identity/address on the blockchain for future reward payments.

Step 3. Aggregation. Upon completion of the previous phase, CASSIE can decrypt all the submissions and obtain each s_i. Next, she computes $v = \sum_{i=1}^{n} VDF(s_i) \pmod M$. It takes time t_{VDF} for CASSIE to evaluate VDF on n values using n processors in parallel. For each evaluation of $VDF(s_i)$, CASSIE obtains the result $x_i = VDF(s_i)$ and a proof π_i of its correctness. CASSIE can use $v = \sum_{i=1}^{n} x_i$ as the desired secret random number.

Step 4. Announcement and Verification. At the end of the gambling day, CASSIE has a deadline to call the `announce` function of the smart contract. This publishes her secret key sk, the decrypted values $\{s_i\}$, the VDF results $\{x_i = VDF(s_i)\}$ and the proofs $\{\pi_i\}$ for efficient public verification, and the final value v. Each participant is also paid a reward of R/n.

When handling the `announce` transaction, the smart contract checks its validity: 1) the revealed secret key sk matches the public key pk, 2) $\{s_i\}$ match the decryption of $\{c_i\}$ under the secret key sk, and 3) $\{x_i\}$ is the result of evaluating the VDF on $\{s_i\}$, with the proof $\{\pi_i\}$. If any check fails, the contract will claim the deposit d_c that was put down by CASSIE by not refunding it.

Since the smart contract knows the value of v at this point, it is also possible to verify that CASSIE has not cheated in any of the bets that were placed during the day or to penalize her for any cheating. This is an orthogonal issue that depends on the details of the casino games being played and is handled exactly as in the previous secret random number generation method of [16].

B. SRNG2

In SRNG1, each participant only sends one transaction to the contract, which saves transaction fees. This protocol's security lies in the impossibility of evaluating the VDF within t_{submit} time so that a malicious party cannot find a suitable value to tamper with the result during the submission phase. However, it is possible to bias the output if CASSIE herself is malicious. For example, suppose there are two participants P_1 and P_2. While P_1 is honest and sends encryption of s_1 to the smart contract, P_2 is malicious and controlled by CASSIE. As a result, P_2 can decrypt s_1. Although P_2 does not know $x_1 = VDF(s_1)$, she can set s_2 to be the same as s_1, so that the output $v = 2 \cdot x_1$ is guaranteed to be an even number. This is considered a violation of tamper-resistance. Specifically, if the downstream application chooses a winner based on whether the random number is odd or even, then CASSIE can manipulate the game.

The tampering issue above only exists if CASSIE herself takes part in the tampering. It is not serious in most real-world applications. It might be alleviated by using an alternative aggregation formula instead of modular summation or by carefully using the random number in the downstream application. Another solution would be to apply a hash function one more time before using the random number, i.e. use $hash(v)$ instead of v. However, without the random oracle assumption, hash functions are not proven to preserve the uniform distribution. Therefore, we propose SRNG2, which uses a commit-reveal strategy to ensure a stronger tamper-resistance. This comes at a cost of one extra message per participant.

Step 1. Initialization. This step is similar to SRNG1. In addition to SRNG1's initialization, CASSIE sets a value d_p, which is the deposit every participant is required to post with the smart contract. She also specifies the duration of the commitment stage and revealing stages as t_{commit} and t_{reveal}, respectively. t_{reveal} is chosen to be much smaller than t_{VDF}.

Step 2. Commitment. Each participant P_i submits the commitment $c'_i = hash(Enc_{pk}(s_i)||nonce_i)$ and pays the deposit d_p by calling the `commit` function of the smart contract. Here,

$nonce_i$ is a nonce chosen by P_i and used as a salt for the hash function.

Step 3. Revealing. P_i calls the `reveal` function of the smart contract to open the hash. More specifically, she reveals $c_i = Enc_{pk}(s_i)$ and $nonce_i$. The smart contract checks whether $hash(Enc_{pk}(s_i)||nonce_i) = c'_i$ and rejects the transaction otherwise.

Step 4. Aggregation. CASSIE decrypts the values to obtain each $s_i = Dec_{sk}(c_i)$ and excludes the shares from malicious participants who committed to two different values or failed to reveal the committed value. She can now compute $v = \sum_{i=1}^{n} VDF(s_i) \mod M$ in sequential time t_{VDF} along with a VDF proof π_i for each $VDF(s_i)$.

Step 5. Announcement and Verification. At the end of the gambling period, CASSIE calls the `announce` function to publish the secret key sk, decrypted values $\{s_i\}$, VDF outputs $\{x_i = VDF(s_i)\}$ and VDF proofs π_i. This function also returns the deposit d_p and pays a reward R/n to each honest participant who correctly and punctually revealed their value.

The smart contract checks that in the `announce` transaction: 1) the revealed secret key sk matches the published pk, 2) $\{s_i\}$ are the decryption results of $\{c_i\}$, 3) The VDF values are computed correctly as in SRNG1.

All other details are exactly as in SRNG1.

IV. Discussion and Security Analysis

In this section, we show that our protocols both satisfy the requirements of distributed secret random beacons.

A. SRNG1

Secrecy. Assume the n participants are not controlled by the same party and at least one of them is honest. Then, under the protection of public-key encryption, unless CASSIE wants to release the final random number v, no party can know every s_i, thus cannot know v. By the aggregation formula, even if a party knows all values except one s_i, v is still unpredictable and uniformly-distributed as long as s_i is uniformly-distributed.

Tamper-resistance. a) Every participant is only required to send one message throughout the protocol, that is, during the submission stage. At the submission stage, every participant except CASSIE does not know other participants' values. b) Even CASSIE does not know any information about $VDF(s_i)$ if P_i is an honest participant so she cannot dominate v arbitrarily by submitting a complement value herself. However, a tampering similar to the one used above to motivate SRNG2 is possible.

Auditability. The secret key sk should be publicized so that each s_i will be publicized. VDFs allow efficient public verification given that CASSIE should evaluate $VDF(s_i)$ and provide proof π_i. If CASSIE fails to publish this data, she will lose her deposit.

Communication complexity. Each participant only sends one transaction of $O(1)$ length. In total, there are $n+2$ transactions and the overall length is $O(n)$. Thus, our approach is asymptotically optimal in terms of communication complexity.

Liveness. Firstly, if CASSIE is dishonest and does not call the `announce` function, any blockchain account can call the `withdraw` function to send rewards to participants and also earn some deposit. The remaining deposit of CASSIE is burnt. Therefore, it is not likely that CASSIE does not call the `announce` function. Secondly, a participant cannot prematurely terminate the protocol or cause it to fail, because she only interacts with the smart contract once.

B. SRNG2

Secrecy. SRNG2 guarantees secrecy by cryptographic hash functions at the commitment stage and public key encryption at the revealing stage. It provides the exact same guarantees as in SRNG1.

Tamper-resistance. Participants cannot manipulate v because they do not know the values s_i of others at either the commitment stage or revealing stage. In contrast to SRNG1, CASSIE cannot tamper with v due to her inability to invert the hash function and know other players' values during the commitment phase. Note that CASSIE, if she takes part as a player, has to already commit to her value in this phase and the deposit d_p is assumed to be large enough to deter any dishonest act of non-revelation.

Auditability. The same argument as in SRNG1 holds.

Communication complexity. Each participant sends one transaction at the commitment stage and another transaction at the revealing stage. Since we have n participants, the total number of transactions would be $2 \cdot n$. Additionally, CASSIE calls `initial` and `announce` functions and sends $O(n)$ bits. Thus, the total communication complexity is $O(n)$ and asymptotically optimal.

V. Conclusion

Secret random beacons are useful in applications such as decentralized casinos, where the casino wants to use a decentralized random number for games but has to keep the random number secret for a predetermined amount of time, e.g. a day, and make it publicly verifiable afterwards. Considering such application scenarios, this work presented a novel efficient decentralized protocol to generate secret random numbers. We designed two variants based on public-key encryption, commitment schemes, and verifiable delay functions. Both protocols are shown to satisfy the requirements of distributed secret random beacons and can be implemented as smart contracts. In comparison with the only previous approach for secret random number generation on the blockchain, i.e. [16], our protocols reduce the communication complexity from quadratic to linear with respect to the number of participants.

Acknowledgments

Removed to preserve anonymity for review.

References

[1] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: A provably secure proof-of-stake blockchain protocol,” in *CRYPTO*, 2017, pp. 357–388.

[2] B. David, P. Gaži, A. Kiayias, and A. Russell, “Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain,” in *CRYPTO*, 2018, pp. 66–98.

[3] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling Byzantine agreements for cryptocurrencies,” in *SOSP*, 2017, pp. 51–68.

[4] A. Kavousi, Z. Wang, and P. Jovanovic, “Sok: Public randomness,” *IACR Cryptol. ePrint Arch.*, p. 1121, 2023.

[5] G. Wang and M. Nixon, “RandChain: Practical scalable decentralized randomness attested by blockchain,” in *IEEE Blockchain*, 2020, pp. 442–449.

[6] M. Krasnoselskii, G. Melnikov, and Y. Yanovich, “No-dealer: Byzantine fault-tolerant random number generator,” in *INFOCOM*, 2020, pp. 568–573.

[7] P. Schindler, A. Judmayer, N. Stifter, and E. R. Weippl, “HydRand: Efficient continuous distributed randomness,” in *S&P*, 2020, pp. 73–89.

[8] K. Choi, A. Manoj, and J. Bonneau, “Sok: Distributed randomness beacons,” in *S&P*, 2023, pp. 75–92.

[9] M. Raikwar and D. Gligoroski, “Sok: Decentralized randomness beacon protocols,” in *ACISP*, 2022, pp. 420–446.

[10] “RANDAO: A DAO working as RNG of Ethereum,” 2019. [Online]. Available: https://github.com/randao/randao

[11] D. Yakira, A. Asayag, I. Grayevsky, and I. Keidar, “Economically viable randomness,” *arXiv preprint arXiv:2007.03531*, 2020.

[12] B. Schoenmakers, “A simple publicly verifiable secret sharing scheme and its application to electronic voting,” in *CRYPTO*, 1999, pp. 148–164.

[13] I. Cascudo and B. David, “SCRAPE: scalable randomness attested by public entities,” in *ACNS*, 2017, pp. 537–556.

[14] ——, “ALBATROSS: publicly attestable batched randomness based on secret sharing,” in *ASIACRYPT*, 2020, pp. 311–341.

[15] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, “Scalable bias-resistant distributed randomness,” in *S&P*, 2017, pp. 444–460.

[16] P. Fatemi and A. Goharshady, “Secure and decentralized generation of secret random numbers on the blockchain,” in *IEEE BCCA*, 2023.

[17] Y. Dodis and A. Yampolskiy, “A verifiable random function with short proofs and keys,” in *PKC*, 2005, pp. 416–431.

[18] B. Wesolowski, “Efficient verifiable delay functions,” in *EUROCRYPT*, 2019, pp. 379–407.