Adding All Flavors: A Hybrid Random Number Generator for dApps and Web3

Ranjith Chodavarapu* Rabimba Karanjai†
Xinxin Fan‡ Weidong Shi§ Lei Xu¶

Abstract

Random numbers play a vital role in many decentralized applications (dApps), such as gaming and decentralized finance (DeFi) applications. Existing random number provision mechanisms can be roughly divided into two categories, on-chain, and off-chain. On-chain approaches usually rely on the blockchain as the major input and all computations are done by blockchain nodes. The major risk for this type of method is that the input itself is susceptible to the adversary's influence. Off-chain approaches, as the name suggested, complete the generation without the involvement of blockchain nodes and share the result directly with a dApp. These mechanisms usually have a strong security assumption and high complexity. To mitigate these limitations and provide a framework that allows a dApp to balance different factors involved in random number generation, we propose a hybrid random number generation solution that leverages IoT devices equipped with trusted execution environment (TEE) as the randomness sources, and then utilizes a set of cryptographic tools to aggregate the multiple sources and obtain the final random number that can be consumed by the dApp. The new approach only needs one honest random source to guarantee the unbiasedness of the final random number and a user can configure the system to tolerate malicious participants who can refuse to respond to avoid unfavored results. We also provide a concrete construction that can further reduce the on-chain computation complexity to lower the cost of the solution in practice. We evaluate the computation and gas costs to demonstrate the effectiveness of the improvement.

1 Introduction

Random numbers play an important role in a wide range of applications, such as machine learning, simulation, and cryptography. When it comes to blockchain, random numbers are critical for not only the blockchain construction itself (e.g., implementation of proof-of-stake), but also dApps deployed on top of it (e.g., gaming and NFT). An adversary can gain extra advantages if it can manipulate or influence the random numbers involved.

A good random number, or a good random number sequence, should meet two requirements: (i) *Unpredictable*. One should not learn the value before it is released or guess future a random number from historical information; and (ii) *Unbiased*. The value should follow the uniform distribution, i.e., each value is selected with the same probability; These two requirements are not independent. It is easy to see that if the random numbers generated are biased, it is easier for the adversary

*Kent State University, Kent OH 44224, USA. Email: rchodava@kent.edu
†University of Houston, Houston TX 77204, USA. Email: rkaranja@cougarnet.uh.edu
‡IoTeX, Menlo Park CA 94025, USA. Email: xinxin@iotex.io
§University of Houston, Houston TX 77204, USA. Email: wshi3@central.uh.edu
¶Kent State University, Kent OH 44224, USA. Email: xuleimath@gmail.com

to predict it. When the random number is generated in a hostile environment, there are extra security related requirements. For instance, an adversary should not be able to influence the value directly/indirectly. In the context of blockchain and dApp, it is also desirable that the random number generation is verifiable by all participants. In this paper, we focus on random number generation used in a decentralized environment, especially dApps.

Random number generation can be classified by various criteria. Based on the original source, random number generators can be divided into *pseudo random number generator (PRNG)* and *true random number generation (TRNG)*. A TRNG utilizes unpredictable physical sources to generate random numbers, and a PRNG uses a mathematical algorithm and a seed value to generate a sequence of numbers. While it is widely believed that TRNG offers random numbers with better quality, most random numbers used in practice are created by PRNGs, because TRNG requires hardware that is not always available. From the perspective of dApps, the random number generation mechanisms are usually divided into two categories, on-chain RNG and off-chain RNG. An on-chain RNG usually uses the blockchain information as a seed to produce random numbers. This approach is convenient for the dApp to verify and consume the generated random number but the seed is suspectable to the influence of an adversary. On the other hand, an off-chain RNG does not rely on the blockchain contents to generate random numbers but the process is usually opaque to the dApp and the dApp participants need to trust that this process is not manipulated.

To better serve the demands of dApps on random numbers, we propose HRNG, a hybrid RNG solution that keeps the advantages of all four types of RNGs while minimizing related limitations. HRNG takes advantage of the emerging DePIN technology (decentralized physical infrastructure network [1]) to solve the problem of introducing physical randomness sources to the blockchain world. The idea of DePIN is to connect a large number of IoT devices to the blockchain and allow them to trade their generated data on the blockchain [2]. When an IoT device is equipped with a hardware-based trusted execution environment (TEE), it can utilize the hardware to work as a TRNG.If all nodes of the blockchain trust the IoT device, the problem of obtaining a random number becomes trivial. For instance, the dApp can request a random number from the IoT device, and the TEE hardware of the IoT device responds with a generated number and a digital signature. As the device is trusted, one only needs to verify whether the provided number and the digital signature are consistent, and does not need to worry that the device colludes with the dApp.

However, as a hardware-aided security enhancement mechanism, TEE is not 100% reliable. In other words, it is possible that an adversary compromises the TEE hardware and manipulates the generation of random numbers [3]. The good thing is that existing attacks against TEE are usually not scalable, i.e., it is hard to compromise a large number of IoT devices equipped with TEE simultaneously. Therefore, HRNG utilizes a group of TEE-enabled IoT devices and aggregates their outputs to provide random numbers to dApps to tolerate the potential risk of compromised hardware. Specifically, HRNG uses multiple TEE devices to create a pool of random numbers, and allows a dApp to combine multiple values from the pool using a preferred method to obtain the final random number it needs as long as it meets the security requirement. Even if some the input random numbers are manipulated by an adversary, the combination mechanism can still guarantee that the final result is not biased. Compared with existing random number generation mechanisms for dApps, HRNG provides the following benefits: **(i)** *TRNG-based*. HRNG utilizes TRNGs as the source to build the final random numbers for dApps, and it enjoys all the advantages of TRNG. **(ii)** *Robustness*. HRNG can not only tolerate IoT devices with compromised TEE, but also other malicious participants in the random number generation process under reasonable security assumptions. **(iii)** *Verifiability*. Except that any third party can verify that the final random number that is consumed by the dApp is obtained following the predefined aggregation method.

In summary, our contributions in the paper include: (i) We provide the details of the design

Table 1: Comparison of HRNG with existing random number generation methods.

Method	**True Random source**	**Verifiability**	**Tolerance of adversaries**
On-chain	○	●	◐
Off-chain	○	◐	◐
HRNG	●	●	●

●: Yes ○: No ◐: Partially

of HRNG that utilizes TRNGs as a building block to generate random numbers for dApps. The design is generic and highly customizable, i.e., one can easily instantiate the system with preferred cryptographic tools based on the demands; and (ii) We propose a concrete construction of HRNG that utilizes the homomorphic feature of the underlying commitment scheme to significantly reduce the on-chain computation complexity, which makes the random numbers more affordable for dApps.

2 Background and Related Works

Existing random number generation mechanisms for dApps are roughly divided into two categories, off-chain random number generation and on-chain random number generation. In this section, we briefly review these mechanisms and compare them with HRNG.

Off-chain random number generation. Using an off-chain source for random number generation is the most convenient approach from an engineering perspective. Randao [4] is a random number generation mechanism used in the beacon chain to provide in-protocol randomness in Ethereum 2.0 [5]. The design of Randao follows the commit-reveal approach to incrementally gather randomness from participants. Motivated by the design of RanShare [6], the Near protocol described a randomness beacon scheme [7] that inherits the randomness properties of Randshare and can tolerate up to 2/3 malicious actors before one can influence the output. Chainlink built a provably fair and verifiable random number generator based on verifiable random functions [8]. Chainlink VRF relies on a decentralized oracle network and can generate one or multiple random numbers together with a cryptographic proof in a single smart contract call. The proof, which can attest to the correctness of the random number generation process, is published and verified on-chain before a dApp consumes the generated random numbers. The off-chain collaborative random number generation schemes like Randao often require multiple participants and communication rounds, thereby incurring significant costs and delays. The VRF-based approach, on the other hand, relies on strong protection of private keys used for computing VRFs.

On-chain random number generation. Creating random numbers on-chain is another option. One straightforward approach for on-chain random number generation is using the block contents (e.g., block headers, block heights, timestamps, smart contract states, etc.) as the random source [9, 10]. The ERC721R [11, 12] standard describes the process for producing random numbers using the aforementioned on-chain metadata. While it is easy for a smart contract to access various on-chain data, it is extremely difficult, if not impossible, to prevent on-chain random number generators from being exploited by attackers [9, 10].

This work. Compared with existing on-chain and off-chain random number provision methods, HRNG provides a new way to split the on-chain and off-chain workload, and utilizes TEE technology to incorporate TRNG into the process of random number generation. Furthermore, HRNG also considers the associated centralization risk in the design. Table 1 summarizes the comparison of

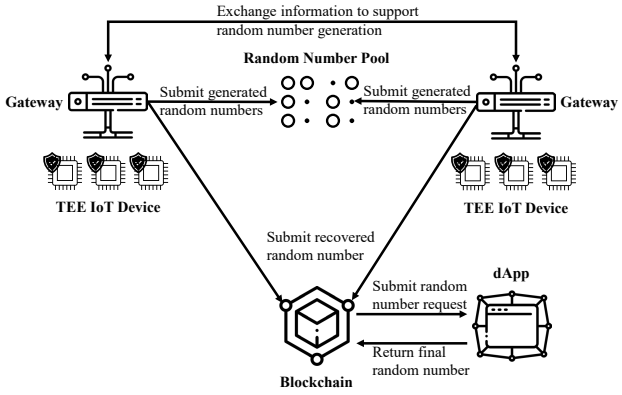

Figure 1: Overview of HRNG. Each gateway connects with multiple IoT devices equipped with TEE to provide random sources to the system, and the final random number is generated in a verifiable manner on the blockchain. Although the dApp is demonstrated as a different entity, it is actually deployed on the same blockchain and maintained by the same set of blockchain nodes.

these methods, and more detailed analyses are provided in later sections.

3 System Overview, Security Assumptions and Problem Statement

In this section, we first present the overview of the decentralized pseudo-random number generation with TEE technology, and then explain the technique challenges.

Overview of HRNG. Figure 1 shows the major participants involved in the random number generation process. **(i)** *IoT device.* An IoT device utilizes equipped TEE to create random numbers to contribute to the provision of final random numbers to dApps. An IoT device only interacts with the gateway it connects with. **(ii)** *Gateway.* A gateway works as the proxy of a group of IoT devices. The gateway has more computation/communication capacity. It can relay messages generated by connected IoT devices to other destinations and process received IoT messages before sending them out. **(iii)** *Random number pool.* The random pool is a storage service that holds information submitted by gateways and allows the blockchain nodes/gateways to retrieve information. **(iv)** *dApp.* A dApp is an application deployed on the blockchain, and its execution requires a random number as input. Different random numbers may lead to completely different execution results. **(v)** *Blockchain.* The blockchain serves as an immutable ledger to store the necessary information and it is also responsible for the execution of certain computations to support the operation of the whole protocol.

The execution of the protocol consists of four major steps: **(i)** *Publish the random numbers pool.* During this step, each gateway collects random numbers generated by connected IoT devices, processes them, and publishes processing results to the pool; **(ii)** *Obtain instructions from the dApp.* The dApp creates a request for a random number, which includes specifications such as the number of sources and the aggregation method; **(iii)** *Respond to the dApp request on a random number.* The gateways obtain the request of the dApp, and work together to generate responses, which are published to the blockchain; and **(iv)** *Construct the final random number.* The blockchain (through the smart contract deployed on the blockchain) follows the request provided by the dApp to build the final random number, which can be consumed by the dApp. Note that the execution of the protocol follows the sequence strictly, i.e., one step only starts when the previous one finishes.

Security assumptions. The major security concern is that an adversary influences the random number generation process and forces the protocol to generate a biased number based on its

Table 2: Summary of notations.

Notation	Description
n_g	Number of gateways in the system
n_i	Number of IoT devices that connect to a gateway
n_r	Number of random numbers an IoT device generates in one round of protocol
n_{mg}	Number of malicious gateways in the system, and $m < n$
n_{mi}	Number of malicious IoT devices connected to the same gateway, and $t < k$
ℓ	Number of source random numbers selected by the dApp for final random number generation

preference, which may cause various consequences depending on the nature of the dApp. In the extreme case, the adversary controls all components of the system (i.e., all gateways and connected IoT devices), and there is no way to prevent it from choosing a preferred random number. In this work, we consider a more practical scenario where the adversary can only compromise a fraction of each type of participant. Specifically, **(i)** *Compromised IoT devices.* We assume hardware-aided TEE is breakable but hard to compromise at a large scale. Although we assume each IoT device is equipped with TEE, we consider the case that the TEE hardware can be compromised and the IoT device provides biased random numbers or refuses to. At the same time, we assume that only a subset of IoT devices in the system are compromised. **(ii)** *Compromised gateways.* We assume a subset of gateways in the system are malicious but not all of them. A malicious gateway may refuse to respond to the system and/or deviate from the protocol to maximize the benefit of the colluded dApp. **(iii)** *Trusted random number pool.* We assume the random number pool is a reliable storage system, i.e., it is always available and can safely store received information. **(iv)** *Trusted blockchain.* Following the common assumption of a blockchain system, we assume that some blockchain nodes can be malicious but the blockchain is trusted as a whole. This also applies to the dApp deployed on the blockchain.

For ease of description, we summarize notations that are used in the rest of the paper in Table 2.

Problem statement. HRNG aims at guaranteeing generated numbers are uniformly distributed and unpredictable by an adversary under the security assumptions. Specifically, HRNG needs to address the following risks: **(i)** Compromised gateways. A gateway controlled by an adversary can abandon an unfavored random number generated by a connected IoT device, and refuse to respond according to the protocol based on the preference of the adversary; **(ii)** Compromised IoT devices. An IoT device (and the associated TEE) controlled by an adversary can generate biased random numbers, or discard a generated random number if the adversary cannot take advantage of it.

4 Detailed Random Number Generation Protocol

We present the detailed design of HRNG in this section.

4.1 Random Number Generation on IoT Devices

we assume each of the IoT devices d_i creates a random number n_i. To facilitate the verification of the authenticity of n_i, the device d_i also calculates a digital signature of the random number $\sigma_{d_i}(n_i)$.

We assume that the IoT device is equipped with TEE, both the random number generation and signature generation are done inside the TEE and it is not likely that an attacker can manipulate these operations.

As the IoT device d_i does not connect to the blockchain network directly, the generated message $(n_i, \sigma_{d_i}(n_i))$ is sent to the corresponding gateway for further processing.

4.2 Random Number Processing on Gateways

After receiving a message of a random number from a connected IoT device, the gateway cannot put it into the random number pool directly as the dApp may take advantage of it. In other words, the numbers should only be visible to the dApp when it cannot make any choices. To achieve this goal, HRNG utilizes a commitment scheme, which satisfies two features: **(i)** Hiding. Hiding prevents one from learning information about the original message by observing the commitment. **(ii)** Binding. Binding guarantees that it is hard/impossible to open a commitment to a different value. These two security features can be formalized in different flavors, such as computation hiding/binding and perfect hiding/binding. The selection of a commitment scheme will not affect HRNG, as long as the commitment scheme's security features are acceptable. The only requirement is that the commitment scheme is non-interactive [13]. Specifically, the gateway utilizes a *commitment scheme* to hide the received random number before sending it to the pool. Each gateway runs the `Setup` when it joins the system. During this stage, the gateway runs the `Commit` algorithm on the generated random numbers one by one and submits all committed values to the pool. The *hiding* feature of the commitment scheme guarantees that no one can extract information from the original random number. At the same time, the *binding* feature of the commitment scheme prevents the gateway from modifying the committed value at a later stage.

The committed value has to be opened in a later stage when the dApp uses this particular random number. The gateway prepares for the opening operation with a threshold secret-sharing scheme. A secure (k, t)-threshold secret sharing scheme guarantees that with t or more splits of the message, one can reconstruct the message. But with less than t splits, it is hard/impossible to rebuild the original message. For each committed random number, the corresponding gateway uses the (k, t)-threshold secret sharing scheme to break the committed value into k pieces, which are distributed to gateways in the system. The values k and t are related to the security assumptions and system configuration, which will be discussed in Section 5.

4.3 Random Number Request by dApp

The dApp is responsible for generating a request for a random number, which is aggregated using multiple inputs from the pool for security purposes. The request must include two major parts that define the aggregation process, i.e., the way to select numbers in the pool and the method to aggregate selected numbers. This request is distributed to all gateways in the system. For security purposes, HRNG verifies the request based on two criteria: **(i)** The request involves enough number of gateways. The minimum number of gateways needed in the request is determined by the security assumption, i.e., the number of gateways that are compromised and collude with the dApp in the system. **(ii)** The aggregation algorithm only needs a few unbiased inputs to guarantee the output is unbiased. This criterion essentially means the aggregation algorithm needs to be able to tolerate biased inputs as much as possible. There are different ways to achieve this goal. The simplest one is treating each input as a binary string and using `XOR` operator to connect all operands, which only requires one unbiased operand to guarantee the output is unbiased. In most cases, a dApp does not have special requirements on the final random number except unbiasedness, and this simple

XOR-based aggregation algorithm is enough.

4.4 Response to the Request on Gateways

After a request is created and submitted by the dApp, the gateways collaborate to help the dApp to compute the final random number. The final random number is calculated through two steps:

- Revealing committed random numbers in the pool. For a committed value, the opening information is shared with multiple gateways, and a subset of gateways can collaborate to open the committed random numbers. Because of the threshold feature, committed numbers can always be opened even if there is a small number of malicious gateways.
- Aggregating numbers in the pool to produce the final random number. This operation is straightforward when all involved elements in the pool are correctly opened. When the pool is accessible by the public, every blockchain node can easily verify the correctness of the final random number as the aggregation method is usually simple.

4.5 Some Implementation Considerations

We discuss the implementation of some components of HRNG. These design and implementation choices do not affect the architecture of HRNG, though they may affect its performance and reliability.

Construction of the random number pool. The random number pool serves as a temporary information storage system to hold generated random numbers of the current round (in the form of commitments). It can also be used to store splits of commitments open information (each split should be cryptographically protected and only the corresponding). There are several options to build this random number pool. For instance, a centralized FTP server can work as a pool. It is also possible to let each gateway to store everything, or utilize a decentralized storage system such as IPFS or other peer-to-peer storage mechanism [14] as the random number pool. Each option for a random number pool has its pros and cons, and can be chosen based on the actual environment.

The blockchain system. HRNG can work with all types of blockchain systems and dApps deployed atop. As the blockchain is involved in the random number generation process (i.e., verification/opening of commitments and aggregation), its security features affect the reliability of these operations. For instance, if the blockchain is constructed using a BFT style consensus protocol [15], an adversary controls more than 1/3 nodes participating the consensus can disable HRNG or manipulate the final random numbers arbitrarily.

5 Security and Performance Analysis

In this section, we analyze the generic construction of HRNG from security and performance perspectives.

5.1 Security Analysis

There are two major strategies for the dApp and its colluders to manipulate the random number generation to maximize its benefits:

- Active attack. For this type of attack, the dApp and its colluders actively modify the random number generation process to obtain preferred random numbers. Major active attacks include:

(i) An IoT device always generates random numbers that the dApp prefers; and (ii) The dApp always selects gateways that collude with it in the request for a random number.

- Passive attack. For this type of attack, the dApp and its colluders do not actively change any information but skip certain numbers or ignore unfavorable requests. Major passive attacks include: (i) An IoT device discards a generated random number that is unfavorable for the dApp. While this attack requires the IoT device to actively skip generated numbers, the device does not need to modify anything. Therefore, we still classify this attack as a passive attack; (ii) A gateway refuses to collaborate to reveal committed numbers that are unfavorable for the dApp to force the protocol to terminate.

HRNG can prevent both types of attacks under the security assumptions discussed earlier.

Prevention of Active Attack. It does not matter what strategy the adversary takes, its ultimate goal is to manipulate one or more steps in the random number generation process to make the final value biased in a preferred manner. HRNG can thwart this type of attacks because it enforces a checking on the dApp's random number request, which guarantees two features: **(i)** The adversary cannot control all inputs. According to the security assumptions, only a subset of IoT devices/gateways can be compromised. HRNG only needs to ensure that the aggregation algorithm involves more inputs than the maximum number values the adversary can control. **(ii)** The aggregation output cannot be influenced by a subset of inputs. Given an arbitrary aggregation algorithm, it is a hard problem to determine whether any single input can guarantee unbiasedness. But considering the normal requirements of the dApp on the final random number, HRNG only needs to support `XOR` operation for aggregation, and this feature is preserved.

Prevention of Passive Attacks. The nature of a passive attack is "denial-of-service", i.e., when the adversary expects that an operation will lead to non-preferred results, it controls compromised entities to stop responding. This type of attack is usually hard to prevent as it is not easy to force an entity to do certain things. Most existing decentralized systems use an incentive mechanism to discourage such behaviors (e.g., offering rewards for taking actions and enforcing penalties for not taking actions). HRNG adopts a different strategy to deal with this type of attack. **(i)** Prevention of passive attacks in the process of random number generation on the IoT device. From a high level, HRNG divides the random generation into two stages. In the first stage, IoT devices/gateways contribute random numbers to a pool and the adversary does not know which will be included in aggregation. Therefore, the adversary does not have the motivation to control compromised IoT devices/gateways to refuse to participate in the protocol. **(ii)** Prevention of passive attacks in the process of final random number generation. HRNG prevents passive attacks in the aggregation stage by utilizing the threshold secret-sharing scheme to distribute the response capability to multiple entities. Based on security assumptions, there is always a subset of honest gateways. As long as the number of honest gateways exceeds the threshold, random numbers in the pool can be recovered.

5.2 Performance Analysis

In this section, we consider the performance of HRNG with a generic commitment scheme and threshold secret sharing scheme. A dApp has two performance requirements for HRNG: (i) Low latency. The latency is the period between the submission of the random number request and the receiving of the result. The dApp expects a quick response (i.e., low latency), especially when the dApp is time-sensitive. (ii) Low cost. The on-chain computation is usually expensive and the cost is proportional to the amount of computation. To reduce the execution cost, the dApp expects to minimize the on-chain computation involved in the random number generation protocol.

Computation cost analysis. The generation of the final random number includes both on-chain and off-chain works. While on-chain computation cost is more sensitive, it is better to minimize both on-chain and off-chain computation.

Off-chain computation. HRNG keeps most of the computation off-chain, which includes three components: (i) Random number commitment (`Commit`). Gateways convert all random numbers collected from IoT devices to the form of commitments. The total number of commit operations is $n_g \times n_i \times n_r$. (ii) Secret sharing (`Split`). Each gateway splits the opening information of each commitment and distributes to other $n_g - 1$ gateways. The total number of secret sharing operations is $n_g \times n_i \times n_r$. (iii) Secret recovering (`Reconstruct`). Only ℓ committed random numbers need to be recovered, so it involves ℓ `Reconstruct` operations.

On-chain computation. The on-chain part is mainly related to the aggregation phase, i.e., each blockchain node needs which is further divided into two parts: (i) Verification of numbers in the pool. This operation is equivalent to the opening of a set of committed values and the cost is determined by the commitment scheme, i.e., ℓ `Open` operations of the selected commitment scheme. (ii) Verification of aggregation process. The simplest way to verify the aggregation result is by repeating the process. If `XOR` is the only operator used for aggregation, the computation cost is $\ell - 1$ `XOR` operations.

Communication cost analysis. There are two communication expensive steps in HRNG, and both are related to the (k, t)-threshold secret sharing. (i) Distribution of shares of committed random numbers. Each committed number is split into k pieces, so the overall communication complexity is $\mathcal{O}(n_g \times n_i \times n_r \times k)$. (ii) Reconstruction of shares of committed random numbers. As only ℓ values need to be reconstructed, the overall communication complexity is $\mathcal{O}(\ell \times t \times k)$. Here we consider a worse situation, where each gateway distributes its splits to all k other gateways. If all gateways are collaborative, this complexity can be reduced to $\mathcal{O}(\ell \times t)$.

6 Improvement of HRNG

The construction described in Section 4 is generic and does not consider the special features of the building blocks.

Utilization of commitment scheme features. While any secure commitment scheme can meet the security requirement of HRNG, they are not equivalent from a performance perspective. As analyzed in Section 5, the on-chain computation cost of the generic construction includes ℓ commitment open operation, and $\ell - 1$ `XOR` operations which are employed to produce the final random number. If the Pedersen commitment scheme [16] is adopted, each commitment open operation involves two exponents and one multiplication on the given finite field. While this cost is not a big concern for a modern computer, it is not cheap as on-chain computation.

To further reduce the on-chain computation complexity, we utilize the additive homomorphic feature of the Pedersen commitment scheme. Specifically, given two commitments $c_1 = g^{m_1} h^{r_1}$ and $c_2 = g^{m_2} h^{r_2}$, we can build the commitment of $m_1 + m_2$ with one multiplication, i.e.,

$$c_{1+2} = c_1 \cdot c_2 = g^{m_1+m_2} h^{r_1+r_2}.$$

With this feature, we can tweak the design of HRNG, and a blockchain node does not need to open each committed value to verify its correctness before aggregating them to obtain the final random number. Specifically, a blockchain node can aggregate the commitments utilizing the homomorphic feature first, and only verify the validity once. With this strategy, the total on-chain computation cost is reduced to one aggregation of ℓ commitments ($\ell - 1$ multiplications) and one open operation (two exponent operations and one multiplication). Depending on the selection of $\mathbb{G}$, the exponent

and multiplication can mean different arithmetic operations. If $\mathbb{G}$ is a multiplicative group of a finite field, multiplication is a multiplication of two big integers modulo a prime number. If $\mathbb{G}$ is a group of elliptic curve points, multiplication is a scalar multiplication of elliptic curve points. In both cases, the complexity of a typical efficient exponent operation is equivalent to $\mathcal{O}(\log|\mathbb{G}|)$ multiplication operations. In other words, the exponent operation is much more expensive than multiplication, and the new approach can greatly reduce the on-chain computation complexity. We analyze the security and performance of the improved HRNG in the rest of this section.

Security analysis of improved HRNG. With the improved HRNG, the blockchain can only obtain the addition of ℓ random numbers

$$m_1 + \cdots + m_\ell, \tag{1}$$

not the original bitwise `XOR` result

$$m_1 \oplus \cdots \oplus m_\ell. \tag{2}$$

Without loss of generality, we assume each m_i has the same bit length. For Equation (2), as long as one of the input values $m_i, 1 \leq i \leq \ell$ is an unbiased random bit string, the aggregation result is an unbiased random value. The situation of Equation (1) is more complex. If all m_is are defined as integers within a given range and selected randomly, the sum of these values follows a variant of Irwin-Hall distribution [17, 18] other than uniform distribution in that range. Fortunately, m_is are defined in a finite group that is isomorphic to $\mathbb{Z}/p\mathbb{Z}$ where p is a prime number (actually we have $|\mathbb{G}| = p$). Accordingly, the addition given in Equation (1) is addition modulo p. Addition in this structure offers a similar feature as `XOR` on bit strings. Specifically, as long as one element m_i is uniformly selected from $\mathbb{Z}/p\mathbb{Z}$, the result of Equation (1) is uniformly distributed in $\mathbb{Z}/p\mathbb{Z}$.

To prove this feature, we assume m_1 is randomly selected from $\mathbb{Z}/p\mathbb{Z}$, and the adversary manages to set the sum of other elements as a preferred value $a \in \mathbb{Z}/p\mathbb{Z}$. Given an arbitrary value $b \in \mathbb{Z}/p\mathbb{Z}$,

$$\Pr[m_1 + a = b] = \Pr[m_1 = b - a] = \frac{1}{p}. \tag{3}$$

Equation (3) states that it does not matter what strategies the adversary uses to choose the value a, the addition result will be uniformly distributed among all the possible values in $\mathbb{Z}/p\mathbb{Z}$.

Note that there is a tiny difference between a random number in $\mathbb{Z}/p\mathbb{Z}$ and a random bit string in $\{0,1\}^{\lceil\log(p)\rceil}$. Hardware-based TRNGs usually create random bit strings other than random elements in $\mathbb{Z}/p\mathbb{Z}$. To convert a binary string to an element of $\mathbb{Z}/p\mathbb{Z}$, we can simply treat the bit string as an integer and apply a modulo p operation. When the length of the bit string is longer than $\lceil\log(p)\rceil$, the converted result roughly follows uniform distribution in $\mathbb{Z}/p\mathbb{Z}$. Given a value p, a longer bit string can lead to a more uniform distribution in $\mathbb{Z}/p\mathbb{Z}$. If the dApp needs a random binary string other than an element in $\mathbb{Z}/p\mathbb{Z}$, we can treat the element as a binary string and reduce the leading bits to guarantee it roughly follows the uniform distribution.

On-Chain performance analysis of the improved HRNG on Ethereum. The improved HRNG relies on a smart contract to open/verify a set of Pedersen commitments and aggregate all the random numbers using modular additions over a prime finite field. Here we assume the Pedersen commitment scheme is built on an elliptic curve points group, which is common in blockchain. Table 3 summarizes the gas cost for performing arithmetic operations on the Ethereum Virtual Machine (EVM), where the gas cost for computing point addition and scalar multiplication is based on the precompiled contracts on the `alt_bn128` elliptic curve [19].

Table 4 provides a comparison of the gas cost between the non-optimized HRNG construction and the optimized one as described in Section 4 and 6, respectively. Thanks to the homomorphic property of the Pedersen commitment scheme, it is not difficult to find that the optimized implementation

Table 3: Gas cost for performing arithmetic operations in EVM of Ethereum [20]

Arithmetic Operation	**EVM Op-code**	**Gas Cost**
Modular Addition	`addmod`	8
Modular Multiplication	`mulmod`	8
EC Point Addition	`ecadd`	150
EC Scalar Multiplication	`ecmul`	6,000

Table 4: Performance evaluation and gas cost for opening and aggregating ℓ random numbers in EVM of Ethereum.

HRNG	**Operation Calls**			**Gas Cost**
	#`ecadd`	#`ecmul`	#`addmod`	
Non-optimized	ℓ	$2 \cdot \ell$	$\ell - 1$	$12,158 \cdot \ell - 8$
Optimized	ℓ	2	$2 \cdot (\ell - 1)$	$166 \cdot \ell + 11,984$

of HRNG can save the gas cost significantly with the increasing of aggregated random numbers. Note that the gas cost is directly associated with computation complexity, i.e., a method is more expensive in gas if it requires more arithmetic operations.

Figure 2 shows the gas costs of the two versions of HRNG. It is easy to see that the gas cost of the non-optimized HRNG grows linearly as more random numbers are used to produce the final result, while the gas cost of the improved HRNG is almost constant. This difference is important from security perspective, as it is harder for an adversary to manipulate the final result when more numbers are involved. When the system uses 12 random numbers to produce the final result, the improved HRNG only consumes about 10% of the gas of HRNG without optimization.

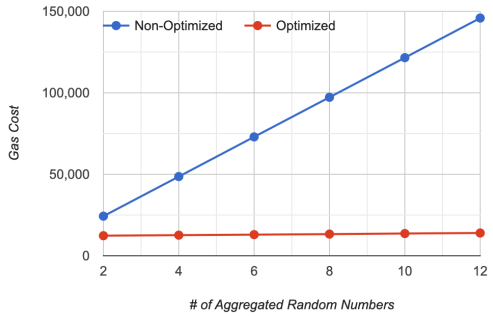

Figure 2: The gas costs for non-optimized and improved HRNG. Here both versions use the elliptic curve-based Pedersen commitment scheme but the unoptimized HRNG does not utilize the homomorphic feature. The x-axis is the number of values to be aggregated and the y-axis is the amount of gas required to finish the work (commitment openings and aggregation).

7 Conclusion

Random numbers play a critical role in blockchain and dApps, and various random number generation mechanisms have been designed to meet this demand. Compared with previous methods, HRNG works in a hybrid manner, i.e., it utilizes both hardware-based TRNG and on-chain/off-chain algorithms for random number generation. This hybrid approach allows HRNG to enjoy the benefits of different random number generation strategies and can tolerate various malicious participants. As on-chain computation is usually expensive, we also design an efficient on-chain aggregation mechanism that does not sacrifice the quality of generated random numbers and decentralization level. The analysis and evaluation show that this improvement can save significant cost (both computation burden and gas fee if deployed for Ethereum) compared with the non-optimized construction.

References

[1] M. C. Ballandies, H. Wang, A. C. C. Law, J. C. Yang, C. Gösken, and M. Andrew, "A taxonomy for blockchain-based decentralized physical infrastructure networks (depin)," in *Proceedings of the 1st International Workshop on Decentralized Physical Infrastructure Networks (DePIN 2023)*. IEEE, 2023.

[2] L. Xu, X. Fan, L. Hall, and Q. Chai, "New gold mine: Harvesting iot data through defi in a secure manner," in *International Conference on Blockchain*. Springer, 2021, pp. 43–58.

[3] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2019.

[4] randao.org, "Randao: Verifiable random number generation." [Online]. Available: https://www.randao.org/whitepaper/Randao_v0.85_en.pdf

[5] B. Edgington, "Upgrading ethereum - a technical handbook on ethereum's move to proof of stake and beyond." [Online]. Available: https://eth2book.info/capella/part2/building_blocks/randomness/

[6] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 444–460.

[7] A. Skidanov, "Near protocol randomness beacon." [Online]. Available: https://pages.near.org/downloads/NearRandomnessBeacon.pdf

[8] Chainlink, "Chainlink vrf." [Online]. Available: https://docs.chain.link/vrf

[9] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Probabilistic smart contracts: Secure randomness on the blockchain," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 403–412.

[10] G. Blaut, X. Ma, and K. Wolter, "Exploring randomness in blockchains," in *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2023, pp. 1–5.

[11] ERC721R, "Bringing greater accountability to nft creators." [Online]. Available: https://erc721r.super.site/

[12] T. Lehman, "Erc721r: A new erc721 contract for random minting so people don't snipe all the rares!" [Online]. Available: https://medium.com/@dumbnamenumbers/erc721r-a-new-erc721-contract-for-random-minting-so-people-dont-snipe-all-the-rares-68dd06611e5

[13] M. Naor, "Bit commitment using pseudo-randomness," in *Conference on the Theory and Application of Cryptology*. Springer, 1989, pp. 128–136.

[14] E. Daniel and F. Tschorsch, "Ipfs and friends: A qualitative comparison of next generation peer-to-peer data networks," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 31–52, 2022.

[15] J. Long and R. Wei, "Scalable bft consensus mechanism through aggregated signature gossip," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 360–367.

[16] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.

[17] P. Hall, "The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable," *Biometrika*, pp. 240–245, 1927.

[18] J. Irwin, "On the frequency distribution of the means of samples from populations of certain of pearson's types," *Metron*, vol. 8, no. 4, pp. 51–105, 1930.

[19] A. Salazar Cardozom and Z. Williamson, "Eip-1108: Reduce alt_bn128 precompile gas costs." [Online]. Available: https://eips.ethereum.org/EIPS/eip-1108

[20] evm.codes, "Evm codes - an ethereum virtual machine opcodes interactive reference." [Online]. Available: https://www.evm.codes/