An Improved Pseudorandom Sequence Generator and its Application to Image Encryption

Keshav Sinha[1*], Partha Paul[2], and Amritanjali[1]
[1]Birla Institute of Technology, Mesra, Ranchi
Jharkhand, 835215 India
[e-mail: keshav.sinha@yandex.com, amritanjali@bitmesra.ac.in]
[2]Sarala Birla University, Ranchi
Jharkhand, 835103 India
[e-mail: p_india@rediffmail.com]
[*]Corresponding author: Keshav Sinha

Received July 30, 2021; revised October 1, 2021; revised February 5, 2022; accepted February 16, 2022; published April 30, 2022

Abstract

This paper proposes an improved Pseudorandom Sequence Generator (PRSG) based on the concept of modular arithmetic systems with non-integral numbers. The generated random sequence use in various cryptographic applications due to its unpredictability. Here the mathematical model is designed to solve the problem of the non-uniform distribution of the sequences. In addition, PRSG has passed the standard statistical and empirical tests, which shows that the proposed generator has good statistical characteristics. Finally, image encryption has been performed based on the sort-index method and diffusion processing to obtain the encrypted image. After a thorough evaluation of encryption performance, there has been no direct association between the original and encrypted images. The results show that the proposed PRSG has good statistical characteristics and security performance in cryptographic applications.

Keywords: Pseudorandom sequence generator, Image security, Probability distribution, Non-integral numbers, Modular arithmetic system, Differential attack.

1. Introduction

An adversary performs various types of attacks on cryptography applications. However, various cryptographic algorithms require random numbers to generate secret keys, nonce, and session keys. The role of the random sequence is to withstand various attacks and provide security to the data. The cryptographic algorithms divide into (i) Symmetric and (ii) Asymmetric encryption. A random number is required for key generation in symmetric encryption, whereas asymmetric encryption requires random bits for padding with the original data. However, hash functions require random numbers for indexing and mapping to provide data integrity and authenticity. This paper concentrates on pseudorandom sequence generation, tests the generated sequence with NIST-STS, and then is used to sort-index based image encryption. The encryption using random numbers requires four factors such as (i) Re-generable, (ii) Statistically Independent, (iii) Uniformly Distributed, and (iv) Storage Efficient. The pseudorandom number generator (PRNG) provides an appropriate solution for generating a secret key. PRNGs are similar to cryptography, where the user feeds the secret seed (S) (low entropy input) into the pseudo system, and it will generate random output (fuzzy random stream).

The cryptographic algorithms have inbuilt blocks for key generation, but they require intense computation. PRNG is used for key generation to cope with the problem of computational time. However, these PRNGs fail to produce large space sequences due to the linearity produced by the generator's integer parameters. The proposed Pseudorandom Sequence Generator uses the property of irrationality to generate the random sequence. The randomness of PRSG determines by evaluating the random bits with the NIST- STS. Apart from the statistical properties, there are two main requirements for PRSG such as speed and robustness. The speed may not play an essential role while generating the single cryptographic key, but in the case of a stream cipher, simulation or masking of protocols require a large number of random sequences, and thus speed becomes the essential prerequisite during that situation. The robustness of PRSG against attacks plays an essential role in data security. The significant contributions of this work are as follows:

(1) The improved pseudorandom sequence generator using non-integral numbers is proposed to avoid the linearity in the generation and provide large sequence space.
(2) The main idea uses the properties of non-terminating, non-repeating decimals in a modulo arithmetic system to generate the sequence endlessly.
(3) The standard and empirical analysis is performed to validate the randomness of generated sequences.
(4) The secure image encryption has been performed using the constructed pseudo-random generator.
(5) The uniformity in generated keys for image encryption is examined using variance and key sensitivity analyses.
(6) The degree of uncertainty in the generated random variable is examined using Entropy analysis.
(7) The key-space and speed-space analyses is used to measure the strength of the proposed generator that resists brute force attacks.

The rest of this work has been organizing as follows. We introduce various traditional mathematical models for pseudorandom number generators in Section 2. Section 3 and Section 4 present the overall architecture and performance evaluation of generated sequence. Section 5 examines the generated sequence by using image encryption. At last, the entire works have been concluding in Section 6.

2. Related Work

2.1. Random Number Generator

In the earlier time, mathematicians and scientists were very fascinated about generating random numbers using deterministic systems. The feature of the generated sequence does not allow any pattern or cycle in the generation. The family of randomness divides into four groups [26] (i) Ontic, which always follows some fixed law or parameters to generate the sequence, (ii) Epistemic follows the environmental factors or unknown laws for a generation. (iii) Pseudo, uses the statistical properties to generate the random stream, and (iv) Telescopic, which is neither random nor looks random. It is tough to conceive that any modern cryptographic algorithm does not use random numbers. The random number generator classifies into three classes: (i) True Random Number Generator (TRNG), (ii) Pseudorandom Number Generator (PRNG), and (iii) Hybrid Random Number Generator (HRNG). A TRNG uses natural randomness rather than non-deterministic sources. On the other hand, PRNG uses the deterministic system to generate random numbers. The HRNG uses a seed value selected from the hardware interface and atmospheric noise to generate the sequence.

2.2. Pseudorandom Number Generator

The pseudorandom generator uses mathematical models to generate the sequence. The sequence generation requires seed value (S_x) and modulus function to generate numbers (S_x'). For the next round, the (S_x') has fed as seed ($S_{x+1}' = S_x \leftarrow S_x'$), the iteration continues until the system terminates. The PRNGs depend on the period that decides the length of the sequence before reaching the same vector. For cryptographic application, the PRNGs required two properties (i) they must pass the statistical test, and (ii) they resist various attacks [3].

2.3. PRNG Classification

The PRNG classifies into two types (i) 1/P Generator and (ii) Modulo Arithmetic System. In 1/*P* generator, the user input (P) as a prime number, which is divided by one to get the output as a quotient. The drawback of this method is that (i) entirely predictable and (ii) generates a small segment of a stream. On the other hand, Carl Friedrich Gauss [22] has first presented the modern approach of the Modular arithmetic system. It introduced the terminologies such as (i) Seed value, (ii) Deterministic sequence which passed randomness, and (iii) Cycle length, Tail, and Period. In 1949, John Von Neumann had first developed the Middle-Square Method (MSM), which has a concise range of sequence generation. The working principle of MSM is elementary; here, the user has to take the (n) digit; from that, the middle (r) digit is squared and used for the next round [18]. In [7], the authors generated uncountable finite numbers using the concept of 1/p generator and presented them by using Eq. 1.

$$\alpha_{b,c} = \sum_{m=1}^{\infty} \frac{1}{(b^m c^{b^m})} \tag{1}$$

Here c > 1 (integer), b = odd number and co-prime to c, and m = number of iteration. Stoneham first coined the background of this generator as 2-normal. There are various pseudorandom number generators based on the modular system. These systems had originated by using Fermat law and the Euclid Division algorithm [28]. The Euclid Division Algorithm stated that for any two integers ‘a’ and ‘b’, there exist unique numbers ‘q’ and ‘r’ such that, (a = bq + r), {i.e., when a divides by b, it leaves quotient q and remainder r}, where ($0 \leq r < b$ and $b \neq 0$) and ‘q’ and ‘r’ is unique. The proposition of Fermat law for any integer is as follows, for any integer ‘a’, there exists an integer ‘b’ such that, ($a \equiv r \bmod b$), where {r = remainder and b divides a, $\{(0 \leq r \leq b), r \in I\}$. Now, according to Euclid and Fermat theorem, it is written as

[(a ≡ r mod b) ≅ (a = bq + r)] for some (q ∈ I). The congruence of the Fermat theorem had extended the Euclid algorithm. Fermat stated [16] that for any integer 'a' and 'm', where (m ≠ 0) are said to be congruent modulo (m) is written as Eq. 2.

$$a \equiv b \bmod m \tag{2}$$

As seen from the Euclid algorithm, 'b' is unique, and the congruence mainly focuses on the remainder. The properties of Euclid and Fermat theorems specify as follows:

(1) Property 1: Both calculate the remainder when another non-zero number divides the number.
(2) Property 2: The number has uniquely written in the form of (a = bq + r) for every number 'a' and 'b', and it defines the stream (r_1, r_2, …, r_i). Here, r_i = remainder in each successive step.
(3) Property 3: For every large integer, we obtain stream <r> = [r_1, r_2, …, r_n], which is large or small depending on the choice of the numbers.

The limitations of Euclid and Fermat theorem are as follows: (i) Both the algorithm congruence do not define for non-integral numbers, (ii) It must need to have the track of every upcoming number (even in decimals), (iii) It takes a very long time, and procedure for the calculation of large numbers, and (iv) For every rational number the stream <r> is permanently terminating, and hence it repeats after some time. These are some properties of traditional generators for sequence generation.

2.3.1. Modulo Arithmetic System

In this section, we discuss some of the generators based on the modulo arithmetic system. In 1958, Linear Congruential Generator (LCG) had developed using the discontinuous linear equation function, represented by Eq. 3 [4].

$$X_{n+1} = (a \times X_n + c) \bmod m \tag{3}$$

Here seed value ($X_n \in I$), m = moduli, (m > 0), a = multiplier, and c = incrementor, and the range of stream belongs to ($0 \leq a < m$) & ($0 \leq c < m$) and [a, c ∈ I]. The LCG system generates long-range sequences, which use in various applications [20]. In 1986, Michael O. Rabin's one-way function generated the random sequence presented using Eq. 4 [4].

$$X_{n+1} = X_n^2 \bmod m \tag{4}$$

Here X_n = seed value, m = (p × p) the product of large co-prime numbers (safe prime). The BBS system generates an extended range of cryptographically secure sequences [23]. In [24], the author has introduced the Fibonacci series-based pseudorandom number generator is represented by using Eq. 5.

$$F_k = (A \times k) \bmod B \tag{5}$$

Here k = (1, 2, …, (B-2), (B-1)), and 'A' & 'B' are the Fibonacci numbers. The author had claimed that it is easy to compute, secure, and generate long-range random sequences. The generation of the pseudo number is a challenging task, and researchers are always trying to find different ways to overcome this challenge.

2.3.2. Non-Integral Number

In this section, we present non-integral number-based sequence generation. In [14], the author generated 134 million decimal random values of π on the NES SX-2 supercomputer. The 100,000 decimals random values had generated using the Euler constant (e) [9]. In [15] author had computed one million random decimal values square root of 2. In [30], the author had computed the different base values of square roots for 3, 5, 6, 7, and 10 for random number

generation. The digit generated for different base values is as follows: 55296, 36864, 32768, 30720, and 24576. In [5], the author had investigated the behavior of pseudorandom numbers based on irrational numbers. The tests contain 100,000 decimal values for each π, Napier constant (*e*), and the square root of 2, 3, 5, 7, 11, and 13 for the randomness test. In [19], the author has proposed pseudorandom number generators based on the irrational numbers presented by Eq. 6.

$$X_j = \left(\frac{(X_{j-1}+n) \times K_1 \times (M-j)}{K_2}\right) \bmod n \tag{6}$$

Here X_j = Seed value, $j \in \{1, 2, \ldots, N\}$, n = positive integer (natural number), K_1 & K_2 are set of irrational numbers, M = N + p (N is the maximum number of element 'X_j' and $p \in N$).

2.3.3. Chaotic System

The Chaos theory has an alternative way for the generation of random sequences for cryptographic applications. The chaotic map categorizes into two different types (i) discrete-time (logistic map) and (ii) continuous-time (Chen chaotic). Here we present five different chaotic systems used in GANs training set [40].

(1) QCNN [32]:

$$\begin{cases} g_1 = -2a_1\sqrt{1-g_1^2}\sin h_1 \\ h_1 = (-b_1(g_1 - g_2) + 2a_1\cos h_1)/(\sqrt{1-g_1^2}) \\ g_2 = -2a_2\sqrt{1-g_2^2}\sin h_2 \\ h_2 = (-b_2(g_2 - g_1) + 2a_2 g_2\cos h_2)/(\sqrt{1-g_2^2}) \end{cases} \tag{7}$$

Here g_1 and g_2 are polarizabilities, h_1 and h_2 are the quantum phase, a_1 and a_2 are proportional energy coefficients, and b1 and b_2 are weighted influence factors of adjacent cells. The chaotic system works when it is initiated with ($a_1 = a_2 = 0.28$, $b_1 = 0.7$, and $b_2 = 0.3$). The system will generate millions of random bits, which has used for image encryption.

(2) 4D hyperchaotic system [31]:

$$\begin{cases} x_1 = \delta_1(x_2 - x_1) \\ x_2 = \delta_2 x_1 + \delta_3 x_2 - x_1 x_3 + x_4 \\ x_3 = x_2^2 - \delta_4 x_3 \\ x_4 = -\delta_5 x_1 \end{cases} \tag{8}$$

Here x_1, x_2, x_3, and x_4 represent the system state vector, δ_1=27.5, δ_2=3, δ_3=19.3, δ_4=2.9, and δ_5=3 are initial hyperchaotic system value, the generation required Lyapunov exponents which is initialized as $\lambda_1 = 1.6170$, $\lambda_2 = 0.1123$, $\lambda_3 = 0$, and $\lambda_4 = -12.8245$. The positive value indicates the system has positive exponents, generating a higher security random sequence.

(3) Fractional Chen hyperchaotic system [17]:

$$\begin{cases} \frac{d^{\alpha}}{dt^{\alpha}} y_1 = \omega_1(y_2 - y_1) + y_4 \\ \frac{d^{\alpha}}{dt^{\alpha}} y_2 = \omega_2 y_1 - y_1 y_3 + \omega_3 y_2 \\ \frac{d^{\alpha}}{dt^{\alpha}} y_3 = y_1 y_2 - \omega_4 y_4 \\ \frac{d^{\alpha}}{dt^{\alpha}} y_4 = y_2 y_3 + \omega_5 y_4 \end{cases} \tag{9}$$

Here y_1, y_2, y_3, and y_4 represent the initial system parameter and initialized as 2, 2, 25, and 2. The system uses the positive values for ω_1, ω_2, ω_3, ω_4, and ω_5 and it is initialized as 35, 3, 12, 7, and 0.58 respectively. The Lyapunov exponents of the hyperchaotic systems initialized as $\lambda_1 = 0.2104$ and $\lambda_2 = 0.126$. The system will create higher security.

(4) Lorenz chaotic system [37]:

$$\begin{cases} \frac{dq}{dt} = \phi_1(w - q) \\ \frac{dw}{dt} = qr - \phi_2 q + w \\ \frac{dr}{dt} = qw - \phi_3 r \end{cases} \tag{10}$$

The system initializes as φ_1=10, φ_2=24.72, and φ_3>8/3 are the parameter of the Lorenz system, and it is proportional to the Prandtl number and Rayleigh number. Here 'r' represents the rate of change with respect to time, 'q' represents the rate of convection, 'w' represents the horizontal temperature variation, and 'z' represents the vertical temperature. It will generate a random sequence. The generated sequence is said to be the best chaotic system.

(5) Henon chaotic system [6]:

$$\begin{cases} x(n+1) = 1 - ax(n)^2 + y(n) \\ y(n+1) = bx(n) \end{cases} \tag{11}$$

Here a = 1.4, and b = 0.3 are the control parameters, and $x, y \in R^w$ to control the dimension of the chaotic system. The Henon map generates a large set of random sequences. The Chaotic sequence is mainly used for cryptographic applications.

2.3.4. Traditional Generators

As mentioned earlier, PRNGs use encryption in various ways, such as XORing of random bits with original data, padding with data, nonce, and key generation. In [29], image encryption uses BBS systems, where the substitution cipher performs using random sequence on image intensity. In [39], recurrent neural networks and long short-term memory (LSTM) design to generate the random stream. M. Bellare et al. [21] proposed a finite pseudorandom function for message block encryption using the XOR scheme. The [2] hybrid visual crypto-steganography approach presents the color image, where the permutation and combination technique shuffle the pixels. The strength of the PRNGs identifies using key-space, length of the generated sequence, and speed of the generator, which are crucial for the cryptographic key. In [8], the block cipher technique examines and observes that the weak cryptographic key has generated using the tent map efficiently exploits the security of the S-box. The permutation and diffusion structure has been created for image pixels to prevent the chosen-plaintext attack [36]. In [12], the author has applied the brute force attack on Chen's chaotic generated

sequence. The attack reveals 66% of the generated random sequence. Ultimately we have concluded that the small key-space of PRNG quickly exposes the randomness.

3. Proposed Model

In this paper, a modular arithmetic system uses to generate the PRNG. As discussed earlier, the generators state will easily get predicted when the low entropy of seed generates the random sequence. The proposed generator has used the non-integral numbers to achieve sufficient entropy.

3.1. Mathematical Foundation

This section presents the mathematical foundation for the proposed pseudorandom sequence generator. The traditional pseudorandom generator is based on Fermat's law and the Euclid Division algorithm [42]. The limitations of Euclid and Fermat's Theorem (1) It needs to track every upcoming number in each step of the division algorithm, which is practically not possible with non-integral numbers, (2) It takes a very long time, and procedure for the calculation of extremely large numbers, and (3) Every rational number, the generated stream is always terminating and hence it repeats after some time. The mathematical comparison for traditional pseudorandom number generators, indicate some limitation of uniform random sequence. It is observed that the streams are uniformly distributed but not cryptographically secure.

To overcome the above mentioned issues, we propose a pseudorandom generator based on the non-integral number with modular arithmetic to obtain the real numbers. The non-integral number is categorized in two parts (i) Rational, and (ii) Irrational numbers. The rational number is written in the form of $\left(\frac{p}{q}\right)$, where ($q \neq 0$). The proposed generator uses the non-integral number which means 'q' does not divides the 'p', then there are two possibilities, the decimal representation which terminates after a while or the sequence repeats itself and goes on forever. The non-terminating, non-repeating behavior of non-integral number is understood by using the continued fraction (C.F). At first sight it seems difficult to counter the problem of recurrences. The use of continued fractions for any real number (whether rational or irrational) is to be written or approximated as:

$$a = a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{a_4 + \ldots}}} \tag{12}$$

Here every number (a_2, a_3, …) except 'a_1' are positive integer. The continued function having remarkable property based on Euclid's division algorithm. Before that, it is important to mention that every rational number has a finite sequence in its canonical form of the continued fraction. But every irrational number has an infinite sequence in its canonical form of the continued fraction. (*Note:* Every number is written uniquely in the form of C·F). Because in every number ($a_2 + k$), then ($k = a_3 + \alpha$), then ($\alpha = a_4 + \beta$, …) then (a_1, a_2, a_3, …) are determined using Euclid's division algorithm for every (a_2, a_3, …) generated is unique sequence. The advantages (C·F) are: (1) It use irrational number for the construction of random numbers (As Fermat's and Euclid's are only limited to a rational number. We use this fact to generate large sequence of random numbers without repetition), and (2) Irrational number is non-terminating, non-repeating and generate large range of random numbers without repetition. Here we present the proof for non-repeating, non-terminating sequence using the continued fraction.

Proof: Let as take a number 'a' divided by 'b' we get 'a_1' as quotient and 'b_1' as a reminder.

$$k = \frac{a}{b} = a_1 + \frac{b_1}{b} \tag{13}$$

Therefore, $a = a_1 b + b_1$ (by Euclid's). Now, let 'b' be written as $a_2 + \frac{b_2}{c}$ for some $a_2, b_2,$ and c; $b = a_2 c + b_2$, therefore $k = (a_1 + \frac{b_1}{a_2 + \frac{b_2}{c}})$. If this process continues, then we get a finite or infinite sequence of $a_i's$ and $b_i's$. The $a_i's$ and $b_i's$ could be real or imaginary values (we'll only consider real values). Therefore every 'k' can be written as:

$$k = a_1 + \frac{b_1}{a_2 + \frac{b_2}{a_3 + \frac{b_3}{\dots}}} \tag{14}$$

Let 'I' be an integral part of any number 'r'. Therefore, $I = \lfloor r \rfloor$ (Here, $\lfloor \cdot \rfloor$ is the greatest integer function). Then, ($f = r - I$) (Here, 'f' is the fractional part), and ($r = I + F$). If (I_1) be an integral part of $\left(\frac{1}{f}\right)$; $I_1 = \left\lfloor \frac{1}{f} \right\rfloor$, Then, $f_2 = \frac{1}{f} - I$; Then again, $= I + \frac{1}{\left(\frac{1}{f}\right)} = I + \frac{1}{I_1 + f_2}$, so we write in the form (C.F) as.

$$r = I + \frac{1}{I_1 + \frac{1}{I_2 + \frac{1}{\dots}}} \tag{15}$$

Now, it is evaluated that (C·F) of $\frac{1}{f_2}$ which is required to get the canonical form of the continued fraction. Now, based on that we have proposed the pseudorandom sequence generator which generates sequence with continuous linear equation. The key feature of the generator is based on the non-integral numbers. (*Note:* The number which cannot be represented in the form of uniform continued fraction, then it is called an irrational number. There are some of the universal systems are (i) Circle's circumference to its diameter (π), (ii) Euler's number (e), and (iii) Square root of natural numbers except perfect squares). Let us see the working of continued fraction for non-integral numbers. The mathematical represent for the non-integral number ($r = \sqrt{3}$) is presented as: If, r = (I + f), (I= integral part and f = fractional part). Then,

$$f = R - I = (\sqrt{3} - 1) = \frac{1}{f} = \left(\frac{\sqrt{3}+1}{2}\right)$$

Now, repeating the process for $\frac{1}{f}$; we get $\frac{1}{f} = I_1 + f_1$ (Here, $I_1 = 1$)

$$f_1 = \frac{1}{f} - I_1 = \left[\frac{\sqrt{3}+1}{2} - 1\right]$$

$$\frac{1}{f_1} = \sqrt{3} + 1.$$

Therefore, by continuing this manner we will get the C.F of $\sqrt{3}$. This process will explain the nature of irrational numbers and provide approximated values for said problems. Now, for the selection in-between rational and irrational numbers, it is clearly evidence that irrational number provide better solution then rational for pseudorandom number generation. The mathematical theory states that all the integer lies on the number line and the set of positive integers counted from (1 to ∞) (goes on forever) and it is also called a set of natural numbers (N = 1, 2, 3, …). In modulo system ($a \equiv b \bmod m$), where ($a \leq b < m$) & ($b \in N$) gives only 'm' possibilities of remainder which has finite set of resides. This creates random sequence of very low entropy, the introduction of decimal system in modulo system generates the long

range of sequence, and lead to the conclusion that between any two integers there are infinite real numbers which increase the efficiency of the generator.

3.2. Proposed Generator

According to the mathematical foundation the proposed pseudorandom sequence generator uses non-integral numbers with modular arithmetic to obtain the real numbers. The mathematical model for the proposed generator is presented in Eq. 16.

$$\text{Decimal output } (S_{dec}) \equiv X_{n+1} = (X_n \times M \times I) \bmod m \tag{16}$$

Here X_n = Seed value [positive Real number], I = Non-Integral Real number with a sufficiently long fractional part, M = Multiplier 'Maddy Constant' (any number $\in I^+$ to increase the value of seed), m = Moduli (any sizeable natural number). (Note. The value of multiplier (M) and non-integral part (I), where (M × I), not an integer). Afterward, the generated sequence divides by moduli (m) to get the integer value. The range of integers stands in between (0 to m), and the conversion of the decimal (S_{dec}) to integer (S_{dig}) number takes place by using Eq. 17.

$$\text{Integer output } (S_{dig}) = \left[\frac{X_{n+1}}{2}\right] \tag{17}$$

Here [•] is the greatest integer function that generates the 'S_{dig}' floor integer value. The initial condition for conversions should be $X_{n+1} > 2$. The conversion of an integer number (S_{dig}) to binary (S_{bin}) performs using Eq. 18.

$$\text{Binary output } (S_{bin}) = X_{n+1} \bmod 2 \tag{18}$$

The range of binary sequences lies in-between $[0, 1]^*$. The proposed PRSG has a moduli value, which means that the lesser the value of 'm', creates low the entropy of keyspace. In this paper, the proposed generator uses the Turing machine computation, a probabilistic procedure to generate random bits. (Probabilistic) poly (n), polynomial-time procedures halt in (worst-case) time, and 'n' is the input length. The generator used the moduli (m) to generate the maximum length of the sequence. **Fig. 1** shows the block diagram for the proposed pseudorandom number generator.

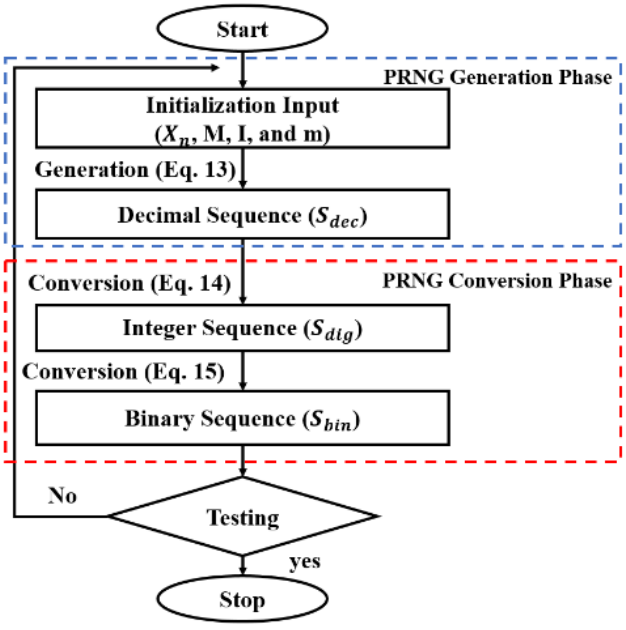


Fig. 1. Block diagram for proposed generator

Initialization: In this phase, the secret seed value 'X_n' is taken for the generator and rest of the parameter has been assigned such as non-integral (I), multiplier (M), and moduli (m).

Generation: Here, the proposed generator has produced three different sequence patterns. The non-integral (I) is used to produce the decimal pattern (S_{dec}) using Eq. 16. Whereas cryptographic applications use PRNG bits, the next step is to transition from decimal to integer numbers. The generation of integer sequence is performed by dividing 'S_{dec}' with moduli 'm' that produce integer sequence. Afterward, the integer sequence is converted into a bit 'S_{bin}' by taking modulo operation.

Testing: It uses to test the randomness of the generated sequence. The NIST-STS, DIEHARD, and ENT statistical tests have been performed on the generated sequence to detect the uniformity of the random bits.

Encryption: The generated bit sequence uses for the cryptographic application. Here, the additive cipher performs image encryption, where the random frame of (256×256) has generated and XOR with the original image to produce the ciphered image.

$$\text{Secret key}_{rand\_bit} = \text{frame } (256\times256) \leftarrow S_{bin}$$

$$\text{Image}_{scramb} = \text{secret key}_{rand\_bit} \text{ XOR Image}_{orig}$$

Analysis: The analysis of the generator is performed based on the key sensitivity analysis. The slight change in the secret seed value generates different sets of random sequences. Then the generated sequence has been XORed with the image file. After that, the variance and correlation analysis perform to detect the deviation in the output.

Here we discussed several properties of the proposed generator which is required to generate the random sequence.

Property 1. If moduli (m = 1, 2), the generated sequence lies between the intervals (0, m). The integer output will be $\{[(X_{n+1}/2) \bmod m] < 2\}$ and the binary sequence generated will be $\{0\}_n$ or (sequence = 0, 0, 0, …) whatever the value of 'M' & 'I' will be chosen.

Property 2. If moduli (m = 3), then the generated sequence lies in between the range of $(0 < X_{n+1}/2 < 1.5)$. It observes that the generated output narrows down the generation and produces a low entropy sequence.

Property 3. If moduli ($m \geq 4$), then the generated sequence belongs to the interval of [0, *m*]; the integer conversion $(0 < X_n/2 < m/2)$. Moreover, this $(X_n \bmod 2)$ will give the sequence of 0's and 1's. (Note: It observes that the multiplier 'M' takes particularly large to generate a more random sequence of 0's and 1's). In generated space, 'S' elements have been repeated because the range of sequence correlates with modulo 'm'. Therefore, the estimated range of generated sequence is about $(m \times 10^6)$ for the proposed generator ($10^{16} \in$ size of the elements after decimal points in R-language). The elements in the generated space (S) have been determining the period of the generator.

Property 4. If moduli (m) be the prime number and multiplier M = 1, then the length of the generated period is up to $(m\times10^6)$. The secret seed $\in (0, m\times10^6)$ observes that the generator does not depend on 'm' whether it is prime or not.

Property 5. If moduli m = power of (2) and multiplier M = 1, then the period of the generated sequence is up to $(m\times10^6)$. It concluded that the generator is partially dependent on moduli value for the generation of random sequence.

Property 6. If modulo ($m \in I^+$), multiplier M = 1, and seed $X_n = 1$, based on the initial parameters if the non-integral ($I \in I^+$), the proposed generator acts similarly to the LCG, which means that the range lies in between ($S \leq m$).

Property 7. If moduli (m = 1) at this point, any traditional known generators do not perform any operation. The proposed generator using the concept decimal number, which

means that the residue below '1' has also existed; hence it is concluded that the proposed generator produces a vast range of numbers (about 10^{16}). The generated sequence has low entropy, and it also exhibits that the proposed pseudorandom sequence generator overcomes the weakness of previously known generators.

Algorithm 1 presents the sequence of steps required to generate the random sequence.

Algorithm 1: Proposed pseudorandom sequence generator (KM-Generator)

01: Start
02: Select seed value (X_n)
03: Initialized non-integral (I), multiplier (M), moduli (m), and n.
04: for i = 1 to n, do
05: Generate: float $S_{dec} \leftarrow X_{n+1} = (X_n \times M \times I) \bmod m$, {Sequence} ∈ {0 to m}
06: Convert: int $S_{dig} \leftarrow$ as.integer (S_{dec}), integer output ∈ {0 to m}
07: Convert: int $S_{bin} \leftarrow (S_{dig} \bmod 2)$, bits ∈ $[0,1]^*$
08: end for
09: Stop

Randomness Evaluation

NIST-STS: The NIST statistical suite has been applied to the generated sequence for randomness testing. The input parameters for testing are initialized based on the standard NIST SP 800 [4], listed in **Table 1**. NIST implementation is performed on Linux Mint 19.1 environment and using R-studio ver.1.1.463 for random sequence generation.

Table 1. Input Parameters

Test Name	Block Length
Block Frequency	128
Non-overlapping Template Matching	9
Overlapping Template Matching	9
Linear Complexity	500
Serial	16
Approximate Entropy	10

The NIST suite is a statistical package that consists of 16 tests and is used to examine the randomness of generated sequences. These tests focus on different types of non-randomness in the sequence [4]. The tests are as follows. (1) Frequency test is used to check the proportion of 0's and 1's in the entire sequence, (2) Block frequency detect the proportion of ones in an M-bit blocks, (3) Cumulative sum test focuses on maximal excursion of 0's in the random walk, (4) Runs test is used to determine the oscillation of 0's and 1's, which are too fast or too slow, (5) Longest run of ones in a block test checked the irregularity of sequence length, (6) Rank test check the linear dependencies among fixed length substring on the original sequence, (7) FFT test detect the repetitive patterns that are near to each other, (8) Non-overlapping template matching test detect too many occurrence of given non-periodic patterns, (9) Overlapping template matching test identify the number of occurrences that are pre-specified with target strings, (10) Universal statistical test detects the number of bits between matching patterns, (11) Approximate entropy test compares the frequency of overlapping blocks of two consecutive length, (12) Random excursion test describe the cycle of exactly k-visits in

random walk, (13) Random excursion variant test detects the deviations from expected number of visit to various state in random walk, (14) Linear complexity test determine that the sequence are complex enough to be consider random, (15) Serial 1 detect the frequency of possible overlapping of m-bit patterns, and (16) Serial 2 detect the 2^m bit overlapping patterns in random sequence. For each statistical test, a set of P-values is obtained corresponding to the produced sequence. Each test considers being successful if the P-value $\geq \alpha$, otherwise it is called failure. The fixed significance level ($\alpha > 0.01$) is required to accept the generated sequence as random.

The initial parameters for the proposed generator are as follows: seed (X_n) = 97645, I = 1.732, M = 18349, and m = 2^{32} - 1. We have generated 1000 samples of length (n = 10^6 bits) and 100 samples of n = 8×10^7 bits). **Table 2** presents the NIST results for two different sample sizes.

Table 2. NIST SP 800-22 statistical test on generated sequence

S. No	Statistical Test	n = 10^6 bits		n = 8×10^7 bits	
		p-values	Results	p-values	Results
01	**Frequency**	0.9043	Passed	0.2144	Passed
02	**Block Frequency**	0.9903	Passed	0.3874	Passed
03	**Cumulative sums**	0.7889	Passed	0.2484	Passed
04	**Runs**	0.9650	Passed	0.5092	Passed
05	**Longest runs**	0.8600	Passed	0.4512	Passed
06	**Rank Test**	0.8032	Passed	0.3631	Passed
07	**FFT**	0.9075	Passed	0.5075	Passed
08	**Non-overlapping Templates**	0.9299	Passed	0.00045	Failed
09	**Overlapping Templates**	0.8820	Passed	0.00005	Failed
10	**Universal**	0.4288	Passed	0.3059	Passed
11	**Approximate entropy**	0.9835	Passed	0.3553	Passed
12	**Random excursions (*x* = *-4*)**	0.3591	Passed	0.4517	Passed
13	**Random excursions variant (*X* = *-9*)**	0.0990	Passed	0.3272	Passed
14	**Linear Complexity**	0.9589	Passed	0.5753	Passed
15	**Serial 1**	0.9453	Passed	0.6986	Passed
16	**Serial 2**	0.9210	Passed	0.5822	Passed

NIST-STS Result Interpretation

As we see the results in **Table 2**, the proposed generator has passed all the statistical tests for 10^6 bits. Nevertheless, for 8×10^7 bits, the generator has failed in Non-overlapping and overlapping Template matching tests. It indicates that the test is not accurate for the high 'n' values [13].

Proportion Analysis: The test is used on certain portions of the sequence to determine the randomness. For example, if we take 1000 sequences as a sample (m=1000) and then the test has applied on it, where the level of significance (α) = 0.01, then it has suggested that 995 sequences have passed the test with (p-value $\geq$ 0.01), with the proportion of 995/1000=0.995. This range for proper proportions has been determined by confidence interval (CI) using Eq. 19.

$$CI = p \pm 3\sqrt{p\,(1-p)/m} \tag{19}$$

Here p = (1 - α), 'α' = level of significance, and 'm' = sample size. If the proportion falls outside of this interval, then we say that the sample set is non-random. The NIST has standardized the confidence p-value interval range between [0.9805, 0.9960].

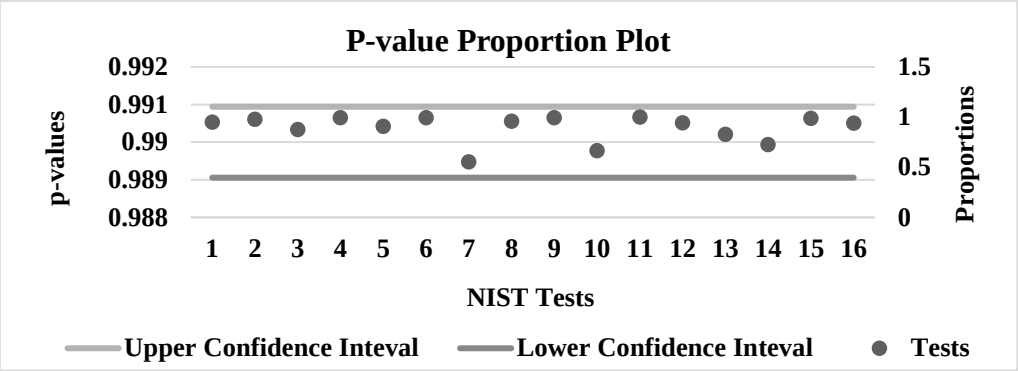


Fig. 2. The Proportion of P-values

Fig. 2 presents the p-value proportion analysis results where the p-value proportion plot shows that out of (16 tests), there are (4 tests) which having the (p-value < 0.9805) and the remaining 12 tests p-values proportion falls inside the interval, this shows that the sequence is sufficiently large and passes the proportion test of uniformity.

Distribution of p-value (Uniformity Analysis): The test uses for a visual illustration of p-value proportion. The uniformity of p-values lies in-between [0, 1]. The distribution divides into ten bins, where p-values lie within bins that have been counted and computed. The Chi-square test (χ^2) calculates the goodness-of-fit for each bin and checks the uniformity of the distribution by using Eq. 20.

$$\chi^2 = \sum_{j=1}^{10} \frac{\left(F_i - \frac{s}{10}\right)^2}{\left(\frac{s}{10}\right)} \tag{20}$$

Here F_i = number of p-values in each 'j' bin, and s = size of the sample. The uniformity of p-values has determined by threshold value which calculates using Chi-square. The test results must be greater than the threshold value to determine uniformity. The threshold p-value was calculated using p-values$_T$ = igamc (9/2, χ^2), where (p-values$_T \geq 10^{-4}$) for uniform distribution.

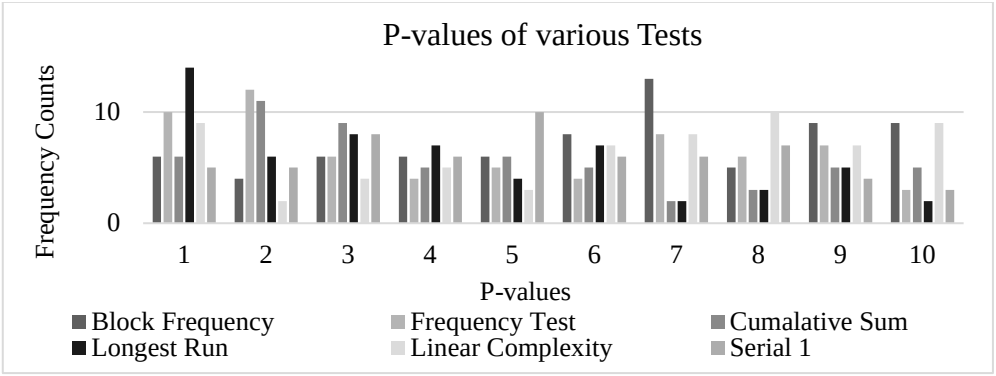


Fig. 3. Histogram of P-values

Fig. 3 represents the uniformity analysis of the P-values for six different statistical tests. The distribution of the p-value divides into ten bin intervals. Here we obtain P-values$_T$ = 0.071620, which is greater than 10^{-4}, indicating the uniformity of p-values in each distribution bin.

DIEHARD Test: The package contains 19 statistical tests for randomness. The tests applied for the same generated sequences. The acceptable range of entire suite passes with a 95% confidence interval for p-values between 0.0001 and 0.9999, and this method was used for our testing. The test description is summarized on the site (https://sites.google.com/site/astudyofentropy/background-information/the-tests/dieharder-test-descriptions).The Diehard test results are summarized in **Table 3**.

Table 3. DIEHARD statistical test

S. No	Statistical Test	$n = 10^6$ bits	
		p-values	Results
1	**Birthday spacing**	0.8016	Passed
2	**Overlapping 5-permutation**	0.0529	Passed
3	**Binary rank (32 × 32)**	0.7342	Passed
4	**Binary rank (6 × 8)**	0.7749	Passed
5	**Bitstream**	0.5905	Passed
6	**OPSO**	0.2735	Passed
7	**OQSO**	0.4123	Passed
8	**DNA**	0.0566	Passed
9	**Stream count-the-ones**	0.8822	Passed
10	**Byte count-the-ones**	0.9691	Passed
11	**Parking lot**	0.4213	Passed
12	**Minimum distance**	0.8743	Passed
13	**D spheres**	0.5438	Passed
14	**Squeeze**	0.5621	Passed
15	**Overlapping sums**	0.0391	Passed
16	**Runs up**	0.9965	Passed
17	**Runs down**	0.3397	Weak
18	**Craps**	0.7114	Passed
19	**K-S Test**	0.9441	Passed

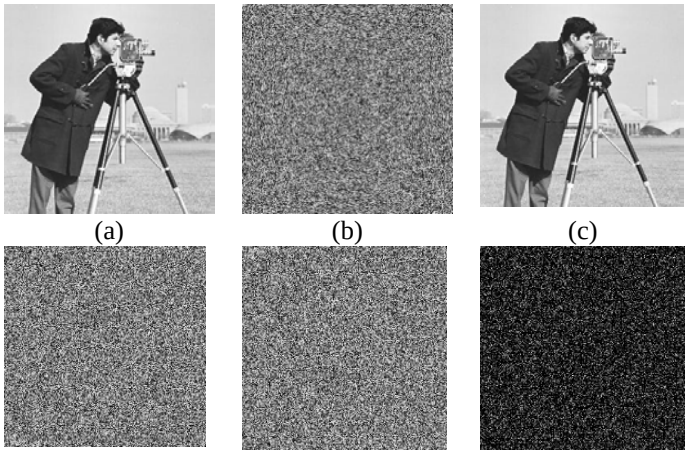
ENT Test: It is another statistical test for the randomness evaluation as follows- (1) Entropy Test measures the information density of generated sequence, (2) Optimum compression measures the sequence based on information density. If the sequence is highly dense, it is unlikely to be reduced in size, (3) Chi-square distribution is used to measure the randomness of data. The absolute number and percentage indicate the frequency of truly random sequence, (4) Arithmetic mean value test simply summing the results all the bytes (127.5 = random), (5) Monte Carlo value for Pi estimation is uses the six bytes sequence within a square. If the distance of the randomly-generated point is less than the radius of a circle inscribed within the square, it is considered a “hit”. The percentage of hits can be used to calculate the value of Pi, and (6) Serial correlation coefficient Test is used to measure the quantity each byte depends upon the previous byte, and the values will be close to zero. **Table 4** presents the results for all the tests from ENT.

Table 4. ENT statistical test

S. No	ENT test	Results
1.	**Entropy**	7.99994 bits per byte
2.	**Optimum compression (OC)**	OC would reduce the size of this 2019999-byte file by 0 percent.
3.	**Chi-Square Distribution**	The Chi-square distribution for 2019999 samples is 311.19 and randomly would exceed this value 97.97 percent of the time.
4.	**Arithmetic mean value**	126.489 (127.5 = random)
5.	**Monte Carlo 'π' estimation**	3.145120754 (error 0.08 percent)
6.	**serial correlation coefficient**	-0.00092 (totally uncorrelated = 0.0)

Performance Analysis with Image Encryption

In this section, the proposed generator uses for image encryption. Here matrix '*A*' has been created using the proposed generator, similar to the original image dimension (256 × 256) based on the floating point standard of IEEE for double variables (IEEE Computer Society, 2008). Then the XOR operation is used in between the original image and matrix (A). The encryption process used the sort-index method to shuffle the pixel positions. The generation of matrix 'A' with original key (K1) is given as X_n = 85289, I = 1.732, M = 95124, and m = 2^{32} - 1, and generate wrong key (K2) with small change in M = 95124 + 10^{-3}.



(a) (b) (c)

(d) (e) (f)

Fig. 4. The results of image encryption using transposition of pixels

Fig. 4 shows the complete cycle of the cameraman image with the original and wrong key. **Fig. 4(a)** presents the original cameraman image. **Fig. 4(b)** represents the encrypted image using a secret key, 'K1'. **Fig. 4(c)** shows the decrypted image. **Fig. 4(d)** presents the decryption using the wrong key 'K2', **Fig. 4(e)** presents the encryption using the wrong key 'K2', and **Fig. 4(f)** represents the difference between 'K1' and 'K2'. It observes that the proposed generator is extremely sensitive towards its input values, and it does not reveal any information, which shows good coherence with image encryption.

Speed Analysis and Comparison: The comparison of encryption time is presented for cameraman images. The decryption process is just the inverse of encryption, and it requires equivalent time. Here, the proposed generator is compared with the traditional cryptographic algorithm and presented in **Table 5**.

Table 5. Comparison of encryption time

Image	Technique	Time (Unit: Second)
Cameraman (256 × 256)	AES	2.472704
	Ref [43]	0.005362
	Ref [44]	0.025781
	Proposed Generator	0.006542

Variance Analysis: It uses to check the uniformity of pixels in the image. According to [38], the quality of the key determines using the variance analysis on encrypted images. The encryption performs by creating a slight change in the input parameter of the original key (K) that creates five different keys (K_i, K_j, K_x, K_y, and K_z). If the variance of two encrypted images is closer to each other, it indicates the high uniformity in the ciphered image. Eq. 21 represents the mathematical formulation for variance analysis.

$$\begin{cases} \text{var}(P) = \frac{1}{256}\sum_{i=0}^{255}(P_i - \overline{P})^2 \\ \overline{P} = \frac{1}{256}\sum_{i=0}^{255} P_i \end{cases} \tag{21}$$

Here the pixel vector is denoted by P, 'i' is the pixel value, $\overline{P}$ = mean, and P_i is the ith pixel value of the pixel vector. The original image encrypts with different key sets that slightly change the parameters, and then the histogram variance has been computed. The standard image is obtained from the site (https://sipi.usc.edu/database/). **Table 6** presents the variance values of the different encrypted images.

Table 6. The variance analysis of small change among the secret key

Image	Plane Image	K	K_i	K_j	K_x	K_y	K_z
5.1.11	1094	5446	5481	5467	5472	5438	5446
5.1.12	3274	5459	5479	5471	5438	5462	5465
5.1.13	5735	5439	5463	5476	5471	5443	5439
5.1.14	1795	5469	5478	5461	5475	5435	5441
Lena	2289	5453	5463	5471	5479	5436	5469
Cameraman	4720	5466	5479	5449	5481	5463	5472

It observes that the variance value of encrypted images using different key sets is very close to the image encrypted with the original key. It implies that the proposed generator is highly sensitive to its initial parameters and uniformly distributed after encryption.

Correlation Coefficient Analysis: It uses to detect the deviation between two adjacent pixels. Here 3000 random pixels pairs are selected from the original and ciphered image to conduct the test—the correlation concept using the mean and variance of adjacent pixels calculated using Eq. 22.

$$\text{Cov}_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{22}$$

Here $\bar{x} = \left(\frac{1}{n}\right)\sum_{i=1}^{n} x_i$, and $\bar{y} = \left(\frac{1}{n}\right)\sum_{i=1}^{n} y_i$. Whereas (x) and (y) are two adjacent pixels of horizontal, vertical, and diagonal direction, 'n' is the total number of pixels. The correlation

coefficient of adjacent pixels is maximum (nearer to 1) for the original image, whereas minimum (nearer to 0) for the encrypted image. Furthermore, the results are prepared based on three different directions (horizontal, vertical, and diagonal), listed in **Table 7**.

Table 7. The Comparison of the Correlation Coefficient

Direction	Proposed Generator		Ref. [1]	Ref. [10]	Ref. [11]	Ref. [33]	Ref. [35]	Ref. [41]
	Plane Image	Cipher image						
Horizontal	0.9417	-0.0021	- 0.0127	0.0005	0.0004	-0.0045	0.0096	0.0101
Vertical	0.9616	-0.0028	- 0.0242	-0.0013	0.0008	0.0004	0.0342	0.0044
Diagonal	0.9243	-0.0026	-	-0.0011	-0.0007	-0.0194	0.0205	0.0006

Here the results of the proposed generator have been compared with various chaotic-based systems. The experimental results show that the plane image values are nearer to '1', whereas the ciphered images values are nearer to '0'. The results of the proposed generator are better than the methods in [1, 10, 11, 35] and comparable with those methods in [33, 41].

Differential Attack Analysis: It uses to detect the single-pixel change in the encrypted image. Two different mathematical models measure the detection (i) Number of Pixels Change Rate (NPCR) is used to check the percentage change in pixels, and (ii) Unified Average Change Intensity (UACI) is used to detect the average change of intensity of the pixel, and calculated by using Eq. 23 and Eq. 24.

$$\text{Percentage pixel (NPCR)} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{k} P(i,j)}{n \times k} \times 100 \tag{23}$$

$$\text{Percentage intensity (UACI)} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{k} |C_1(i,j) - C_2(i,j)|}{255 \times n \times k} \times 100 \tag{24}$$

The analysis required two encrypted images and 'C1' and 'C2' generated from two original images with only a one-pixel difference for analysis. Here we define 2D array D(i, j) having the same size as the original image. Now to detect the one-bit change, if (C1(i, j) = C2(i, j)), then D(i, j) = '1' or else it is '0', this analysis illustrates that the slight change in plaintext sense in the encrypted image. **Table 8** presents the comparison of NPCR and UACI for various traditional and proposed algorithms. Here, we have chosen the random pixel position (150, 36) on the Lena image and converted the pixel from 126 to 125. The standard statistical value for NPCR > 0.995 and UACI > 0.333 for the acceptance of results.

Table 8. The comparison of NPCR and UACI

Test	Ref. [1]	Ref. [10]	Ref. [11]	Ref. [33]	Ref. [35]	Ref. [41]	Proposed Generator
NPCR	99.65	99.63	99.57	83.45	99.82	99.13	99.87
UACI	33.60	33.40	33.36	34.68	33.46	28.72	33.47

Here, the proposed generator values are acceptable and match those methods [1, 10, 11, 33, 35, 41]. Both NPCR and UACI value confirms that the proposed generator resist the differential attack.

Entropy Analysis: It uses to measure the degree of uncertainty in the random variable. Theoretically, the entropy of a grayscale image is equal to ($2^8 \approx 256$) when all the pixels distributes uniformly. If the encrypted value is close to the 8Sh (Shannon), it is highly robust against attacks. The mathematical formulation for entropy 'E' is represented by Eq. 25.

$$E = -\sum_{i=0}^{n} p_i \log_2(p_i) \tag{25}$$

Here 'p_i' is the probability of occurrence of gray pixel value 'i'. The probability of each pixel is 1/256 and lies in the range of [0, 255], which uniformly distributed throughout the region. To create maximum obscuring in the image, maximize the entropy value. **Table 9** presents the comparison of various traditional and proposed generator-based encryption on Lena images. The standard value of entropy ($E \leq 8$) for the 256×256 grayscale image determines the randomness in the image.

Table 9. The comparisons of entropy value

Technique	Entropy value
Ref. [1]	7.9901
Ref. [10]	7.9971
Ref. [33]	7.9972
Ref. [35]	7.9984
Ref. [41]	7.9970
Proposed Generator	**7.9974**

It observes that the proposed generator-based encryption has produced acceptable and better results than those in [1, 10, 33, 41]. Also, the result obtained is slightly lower than the method in [35]. It concludes that the grayscale pixels are distributed uniformly in encrypted images.

2.4. Comparison

In this section, the proposed generator compares with various traditional algorithms. The results prepare using the p-value of the NIST test. **Table 10** presents the comparisons of traditional and proposed generators for 10^6 bits.

Table 10. The Comparison of NIST-STS P-values of the various generator for 10^6 bits

Statistical Test	P-values								
	Ref. [4]	Ref. [4]	Ref. [6]	Ref. [17]	Ref. [19]	Ref. [31]	Ref. [32]	Ref. [37]	Proposed Generator
Frequency	0.739	0.534	0.326	0.203	0.934	0.434	0.539	0.709	0.904
Block Frequency	0.122	0.350	0.763	0.888	0.886	0.308	0.647	0.886	0.990
Cumulative sums	0.384	0.534	0.375	0.352	0.928	0.185	0.596	0.975	0.788
Runs	0.213	0.534	0.902	0.248	0.709	0.677	0.949	0.325	0.965
Longest runs	0.122	0.534	0.311	0.160	0.951	0.696	0.041	0.508	0.860
Rank Test	0.213	0.350	0.629	0.534	0.663	0.677	0.885	0.330	0.803
FFT	0.739	0.350	0.184	0.017	0.907	0.397	0.188	0.116	0.907
Non-overlapping Templates	0.479	0.666	0.449	0.353	0.743	0.814	0.105	0.706	0.929
Overlapping Templates	0.350	0.350	0.107	0.885	0.997	0.352	0.086	0.219	0.882
Universal	0.213	0.017	0.188	0.116	0.605	0.680	0.902	0.016	0.428
Approximate entropy	0.991	0.739	0.170	0.929	0.955	0.480	0.436	0.998	0.983
Random excursions (x = -4)	0.852	0.823	0.609	0.473	-	0.010	0.128	0.655	0.359
Random excursions variant (X = -9)	0.419	0.654	0.133	0.512	-	0.844	0.711	0.806	0.099

Linear Complexity	0.213	0.534	0.405	0.386	0.975	0.100	0.996	0.638	0.958
Serial 1	0.365	0.739	0.998	0.950	0.998	0.161	0.875	0.845	0.945
Serial 2	0.413	0.634	0.997	0.672	0.982	0.032	0.946	0.794	0.921

It observes that p-values of the proposed generator were almost nearer to '1' for the maximum tests and passed all the NIST tests of randomness. It also found that the obtained values are better than the generators in [4, 6, 17, 19, 31, 32, 37]. It also shows that the generator in [19] has failed in two different tests (Random Excursions and Random Excursions Variant test).

Speed-Space Analysis: The analysis uses to evaluate the performance of the proposed generator. **Table 11** presents a comparison of traditional and proposed generators.

Table 11. The comparison of Traditional and Proposed PRSGs

Generator	Operation	Length	Parameters	Space (Mbits)	Speed (Mbits/ sec)
Ref. [4]	Moduli m = 2^{32}	10^6	X_n, a, c, and m	13.4	0.0148
Ref. [4]	Moduli m = 2^{32}	10^6	X_n, m = p × p	14.2	0.0011
Ref. [19]	Moduli m = 2^{32}	10^6	X_j, K_1, K_2, M = (N + p), n	17.1	0.0063
Ref. [33]	XOR	10^6	x_1y_1, x_1z_1, x_2y_2, x_2z_2, x_3y_3, x_3z_3, x_4y_4, x_4z_4, y_5z_5	-	0.3256
Ref. [41]	XOR and Moduli m = 2^{32}	10^6	S, m = p × p	-	0.3900
Proposed Generator	Moduli m = 2^{32}	10^6	X_n, I, M, and m	15.1	0.0065

We observe that the proposed generator has better speed and space than the traditional generators, which suggests [4, 19]. Also, the result is comparable with the methods of [33, 41].

Keyspace Analysis: The essential part of PRNGs is to prevent brute-force attacks. In addition to the secret key, the proposed generator initial parameters are integer values (X_n) and 'M', 'I' a non-integral number. Considering the floating point standard of IEEE for double variables (IEEE Computer Society, 2008), every double variable has precision of about 10^{-15}. The precision of generated sequence with initial parameters is $(10^{20})^4$, while the precision of parameter 'm' is $2^{64} \approx 10^{19}$, and the generator uses the floating-point standards, so the precision sequence length after the decimal is (10^{15}), the keyspace size will be $(10^{20})^4 \times 10^{19} \times 10^{15} = 10^{114} \approx 2^{380}$. **Table 12** presents the keyspace comparison of several random number generators.

Table 12. Keyspace comparison

Techniques	Keyspace
Ref. [10]	2^{199}
Ref. [11]	$> 2^{312}$
Ref. [27]	2^{339}
Ref. [33]	2^{760}
Ref. [34]	2^{149}
Ref. [35]	$> 2^{252}$
Ref. [41]	2^{16}
Proposed Generator	$\mathbf{2^{380}}$

The keyspace of the proposed generator is better than other well-known generators. It observes that the keyspace depends on the key size and possible values in each key. Hence, it observes that the sequence generated by the proposed generator is large enough to resist brute-force attacks.

Conclusion

In the paper, an image encryption scheme is presented based on the improved pseudorandom sequence generator using modular arithmetic systems with non-integral numbers, which also increases the efficiency of the proposed generator. The proposed generator solves the problem of the non-uniform distribution of sequence. The NIST-STS, DIEHARD, and ENT statistical test have been performed on the sequence to determine the randomness of generated sequence. The experimental results and theoretical analysis show that the proposed generator has many advantages, such as sensitivity to initial values, robustness, and resistance against common attacks. The performance of the proposed generator is measured using speed-space and keyspace analysis. Our intent in the future has to use the proposed generator with various cryptographic algorithms for key generation and padding. Furthermore, the proposed generator is applied to encrypt images for secure transmission over the Internet.

References

[1] Benlashram, M. Al-Ghamdi, R. AlTalhi, and P. Kaouther Laabidi, "A novel approach of image encryption using pixel shuffling and 3D chaotic map," *J. Phys. Conf. Ser.*, vol. 1447, p. 012009, Jan. 2020. Article (CrossRef Link)

[2] A. Hasnat, D. Barman, and S. Sarkar, "Color image share cryptography: a novel approach," *J. Intell. Fuzzy Syst.*, vol. 36, no. 5, pp. 4491–4506, May 2019. Article (CrossRef Link)

[3] A. Lavasani, and T. Eghlidos, "Practical next bit test for evaluating pseudorandom sequences," *Sci. Iran.*, vol. 16, pp. 19-33, June 2009. Article (CrossRef Link)

[4] A. Rukhin, J. Sota, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," *NIST*, 2000. Article (CrossRef Link)

[5] B. R. Johnson, and D. J. Leeming, "A study of the digits of π, e and certain other irrational numbers," *Sankhya: Indian J. Stat.*, vol. 52, no. 2, 183-189, 1990. Article (CrossRef Link)

[6] H. Luo and B. Ge, "Image encryption based on Henon chaotic system with nonlinear term," *Multimed. Tools. Appl.*, vol. 78, no. 24, pp. 34323–34352, Aug. 2019. Article (CrossRef Link)

[7] D. H. Bailey, "A Pseudorandom Number Generator Based on Normal Numbers," Office of Scientific and Technical Information (OSTI), Berkeley, CA, USA, Tech. Rep. LBNL-57489, Dec. 2004. Article (CrossRef Link)

[8] D. Lambić, "Security analysis and improvement of a block cipher with dynamic S-boxes based on tent map," *Nonlinear Dyn.*, vol. 79, no. 4, pp. 2531–2539, 2015. Article (CrossRef Link)

[9] D. Shanks and J. W. Wrench, "Calculation of e to 100,000 Decimals," *Math. Comput.*, vol. 23, no. 107, pp. 679-680, Jul. 1969. Article (CrossRef Link)

[10] H. Huang, S. Yang, and R. Ye, "Efficient symmetric image encryption by using a novel 2D chaotic system," *IET Image Process.*, vol. 14, no. 6, pp. 1157–1163, Apr. 2020. Article (CrossRef Link)

[11] H. Huang and S. Yang, "Image Encryption Technique Combining Compressive Sensing with Double Random-Phase Encoding," *Math. Probl. Eng.*, vol. 2018, pp. 1–10, 2018. Article (CrossRef Link)

[12] F. Özkaynak and S. Yavuz, "Security problems for a pseudorandom sequence generator based on the Chen chaotic system," *Comput. Phys. Commun.*, vol. 184, no. 9, pp. 2178–2181, Sep. 2013. Article (CrossRef Link)

[13] İ. Öztürk and R. Kılıç, "A novel method for producing pseudo random numbers from differential equation-based chaotic systems," *Nonlinear Dyn.*, vol. 80, no. 3, pp. 1147–1157, Feb. 2015. Article (CrossRef Link)
[14] I. Peterson, "Pi Wars: Dueling Supercomputers," *Science News, Wiley*, vol. 131, no. 8, p. 118, Feb. 1987. Article (CrossRef Link)
[15] J. Dutka, "The Square Root of 2 to 1,000,000 Decimals," *Math. Comput.*, vol. 25, no. 116, pp. 927-930, Oct. 1971. Article (CrossRef Link)
[16] J. Ferreirós, "Gauss and the Mathematical Background to Standardisation," *HoST - Journal of History of Science and Technology*, vol. 14, no. 1, pp. 32–51, Jun. 2020. Article (CrossRef Link)
[17] J. Peng, W. Yang, S. Jin, S. Pang, D. Tang, J. Bai, D. Zhang, "Image Encryption Based on Fractional-order Chen Hyperchaotic System," in *Proc. of 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, Kristiansand, Norway, pp. 213-217, Nov 2020. Article (CrossRef Link)
[18] L. C. Meiser, J. Koch, P. L. Antkowiak, W. J. Stark, R. Heckel, and R. N. Grass, "DNA synthesis for true random number generation," *Nat. Commun.*, vol. 11, no. 1, pp. 1-9, Nov. 2020. Article (CrossRef Link)
[19] L. Milinkovic, M. Antic, and Z. Cica, "Pseudorandom number generator based on irrational numbers," in *Proc. of 10th International Conference on Telecommunication in Modern Satellite Cable and Broadcasting Services (TELSIKS)*, Nis, Serbia, pp. 719-722, Oct. 2011. Article (CrossRef Link)
[20] M. A. Ivanov, I. G. Konnova, E. A. Salikov, and M. A. Stepanova, "Obfuscation of logic schemes of pseudorandom number generators based on linear and non-linear feedback shift registers," *Bezopasnost informacionnyh tehnology*, vol. 28, no. 1, pp. 74–83, Jan. 2021. Article (CrossRef Link)
[21] M. Bellare, R. Guérin, and P. Rogaway, "XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions," in *Proc. of CRYPTO 1995: Advances in Cryptology - CRYPT0' 95*, pp. 15–28, 1995. Article (CrossRef Link)
[22] M. Bullynck, "Modular arithmetic before C.F. Gauss: Systematizations and discussions on remainder problems in 18th-century Germany," *Hist. Math.*, vol. 36, no. 1, pp. 48–72, Feb. 2009. Article (CrossRef Link)
[23] O. Laia, E. M. Zamzami, and Sutarman, "Analysis of Combination Algorithm Data Encryption Standard (DES) and Blum-Blum-Shub (BBS)," *J. Phys. Conf. Ser.*, vol. 1898, no. 1, p. 012017, Jun. 2021. Article (CrossRef Link)
[24] P. G. Anderson, "A Fibonacci-Based Pseudorandom Number Generator," *Applications of Fibonacci Numbers*, vol. 4, pp. 1-8, July 30 - Aug 3, 1991. Article (CrossRef Link)
[25] R. P. Agarwal and H. Agarwal, "Origin of Irrational Numbers and Their Approximations," *Comput.*, vol. 9, no. 3, p. 29, Mar. 2021. Article (CrossRef Link)
[26] S. Chakraborty, "On Why and What of Randomness," *arXiv:0902.1232 [cs]*, February 2009. Article (CrossRef Link)
[27] S. Zhu, C. Zhu, and W. Wang, "A New Image Encryption Algorithm Based on Chaos and Secure Hash SHA-256," *Entropy*, vol. 20, no. 9, p. 716, Sep. 2018. Article (CrossRef Link)
[28] T. W. Judson, *Abstract Algebra: Theory And Applications*, USA: Orthogonal Publishing, 2020.
[29] V. Kapur, S. Teja Paladi, and N. Dubbakula, "Two Level Image Encryption using Pseudo Random Number Generators," *Int. J. Comput. Appl.*, vol. 115, no. 12, pp. 1-4, Apr. 2015. Article (CrossRef Link)
[30] W. A. Beyer, N. Metropolis, and J. R. Neergaard, "Square Roots of Integers 2 to 15 in Various Bases 2 to 10: 88062 Binary Digits or Equivalent," *Math. Comput.*, vol. 23, no. 107, p. 679, Jul. 1969. Article (CrossRef Link)
[31] W. Hao-Xiang, C. Guo-Liang, M. Sheng, T. Li-Xin, "Nonlinear feedback control of a novel hyperchaotic system and its circuit implementation," *Chin. Phys. B*, vol. 19, no. 3, p. 030509, Mar 2010. Article (CrossRef Link)

[32] W. Sen, C. Li, L. Qin, W. Gang, "Chaotic phenomena in Josephson circuits coupled quantum cellular neural networks," *Chin. Phys.*, vol. 16, no. 9, pp. 2631-4, Sep 2007. Article (CrossRef Link)
[33] X. Chen, S. Qian, F. Yu, Z. Zhang, H. Shen, Y. Huang, S. Cai, Z. Deng, Y. Li, and S. Du, "Pseudorandom Number Generator Based on Three Kinds of Four-Wing Memristive Hyperchaotic System and Its Application in Image Encryption," *Complexity*, vol. 2020, pp. 1–17, Dec. 2020. Article (CrossRef Link)
[34] X. Wang, X. Zhu, X. Wu, and Y. Zhang, "Image encryption algorithm based on multiple mixed hash functions and cyclic shift," *Opt. Lasers Eng.*, vol. 107, pp. 370–379, Aug. 2018. Article (CrossRef Link)
[35] X.-Y. Wang, S.-X. Gu, and Y.-Q. Zhang, "Novel image encryption algorithm based on cycle shift and chaotic system," *Opt. Lasers Eng.*, vol. 68, pp. 126–134, May. 2015. Article (CrossRef Link)
[36] Y. Liu, L. Y. Zhang, J. Wang, Y. Zhang, and K. Wong, "Chosen-plaintext attack of an image encryption scheme based on modified permutation–diffusion structure," *Nonlinear Dyn.*, vol. 84, no. 4, pp. 2241–2250, Feb. 2016. Article (CrossRef Link)
[37] S. Tariq, M. Khan, A. Alghafis, and M. Amin, "A novel hybrid encryption scheme based on chaotic Lorenz system and logarithmic key generation," *Multimed. Tools. Appl.*, vol. 79, no. 31–32, pp. 23507–23529, Jun. 2020. Article (CrossRef Link)
[38] Y.-Q. Zhang and X.-Y. Wang, "A symmetric image encryption algorithm based on mixed linear–nonlinear coupled map lattice," *Inf. Sci.*, vol. 273, pp. 329–351, Jul. 2014. Article (CrossRef Link)
[39] Y.-S. Jeong, K. Oh, C.-K. Cho, and H.-J. Choi, "Pseudo Random Number Generation Using LSTMs and Irrational Numbers," in *Proc. of IEEE International Conference on Big Data and Smart Computing (BigComp)*, Shanghai, China, pp. 541 – 544, Jan. 2018. Article (CrossRef Link)
[40] Z. Man, J. Li, X. Di, X. Liu, J. Zhou, J. Wang, and X. Zhang, "A novel image encryption algorithm based on least squares generative adversarial network random number generator," *Multimed. Tools. Appl.*, vol. 80, no. 18, pp. 27445–27469, May 2021. Article (CrossRef Link)
[41] T. Sivakumar, R. Venkatesan, "A Novel Image Encryption Using Calligraphy Based Scan Method and Random Number," *KSII Transactions on Internet and Information Systems*, vol. 9, no. 6, pp. 2317-2337, Jun. 2015. Article (CrossRef Link)
[42] J. B. Fraleigh, *A First Course in Abstract Algebra*, USA, United States: Pearson, 2014.
[43] F. Sun and Z. Lv, "A secure image encryption based on spatial surface chaotic system and AES algorithm," *Multimed Tools Appl.*, vol. 82, pp. 3959-3979, 2022. Article (CrossRef Link)
[44] A. Arab, M. J. Rostami, and B. Ghavami, "An image encryption method based on chaos system and AES algorithm," *J of Supercom.*, vol. 75, no. 10, pp. 6663–6682, May 2019. Article (CrossRef Link)
[45] K. Kordov and S. Zhelezov, "Steganography in color images with random order of pixel selection and encrypted text message embedding," *PeerJ Comput. Sci.*, vol. 7, p. e380, Jan. 2021. Article (CrossRef Link)

Keshav Sinha received a B.E. degree in Computer Science and Engineering and an M.E. degree in Software Engineering. As a research scholar, he is doing a Ph.D. from BIT, Mesra, in Cryptography and Network security. His current research interest is cryptography and network security which provide flexibility in data security.

Partha Paul received the B.E (CS) and M.E (CS) degrees from Moscow State University, Russia, in 1998 & 1999, respectively. He did his Ph.D. degree from Birla Institute of Technology, Mesra, Ranchi, India, in 2014. He is currently an Associate Professor in the Department of Computer Science & Engineering, Sarala Birla University, Ranchi, India. He has authored or co-authored more than 40 Papers published in various International Journals and Conference Proceedings. His research interests include Cryptography and Network Security, Artificial Intelligence, Cloud Computing, and Traffic Grooming in Optical WDM.

Amritanjali received B.E. degree in Computer Science in 2000, M.E. degree in Software Engineering in 2005 and Ph.D. in Engineering in 2014 from the Birla Institute of Technology, Mesra, Ranchi. After completing her B.E., she worked as Software Engineer at Computer Associates-TCG S/W Pvt. Ltd. where she was responsible for software development and maintenance in various industrial projects. In 2006, she joined the Birla Institute of Technology, Mesra, as Assistant Professor in Computer Science and Engineering. Her research interest is in Wireless Networks, Parallel Computing, Computational Biology, and Machine Learning.