I. J. Mathematical Sciences and Computing, **2025, 3, 32-46**
Published Online on October 8, 2025 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijmsc.2025.03.03

Enhanced Hybrid Pseudo Random Number Generator – Chaotic HPRNG

Md Mahbubur Rahman*
Patuakhali Science and Technology University, Faculty of Computer Science and Engineering, Patuakhali, 8602, Bangladesh
E-mail: first.mahbub.cse@pstu.ac.bd
ORCID iD: https://orcid.org/0009-0006-9891-125X
*Corresponding Author

Hind Biswas
Patuakhali Science and Technology University, Faculty of Computer Science and Engineering, Patuakhali, 8602, Bangladesh
E-mail: ug2302016@cse.pstu.ac.bd
ORCID iD: https://orcid.org/0009-0009-9601-4821

Chinmay Bepery
Patuakhali Science and Technology University, Faculty of Computer Science and Engineering, Patuakhali, 8602, Bangladesh
E-mail: chinmay.cse@pstu.ac.bd
ORCID iD: https://orcid.org/0000-0002-5837-2099

Md Atikqur Rahaman
Patuakhali Science and Technology University, Faculty of Computer Science and Engineering, Patuakhali, 8602, Bangladesh
E-mail: atik.csit@pstu.ac.bd
ORCID iD: https://orcid.org/0009-0002-6653-5836

Md. Moshiur Rahman
School of Science and Technology Bangladesh Open University Gazipur-1705, Bangladesh
E-mail: moshiur@bou.ac.bd
ORCID iD: https://orcid.org/0000-0001-9190-7510

Received: 26 February, 2025; Revised: 21 April, 2025; Accepted: 13 July, 2025; Published: 08 October, 2025

Abstract: Pseudo Random Number Generators (PRNGs) are deterministic and periodic in nature. Hybrid Pseudo Random Number Generators (HPRNGs) address some limitations by using time-based seeding with a modified Linear Congruential Generator (LCG). While HPRNGs improve upon the deterministic nature by using dynamic time-based seeds, they still suffer from periodicity and potential seed-related issues. This study addresses the deterministic nature further as well as the periodicity of PRNGs by proposing an enhanced HPRNG, making it more suitable for high-security applications.

Index Terms: Pseudo Random Number Generator, Chaotic Maps, Tent Map, Logistic Map, Bitwise Rotation, Hybrid PRNG, Chaos-Based RNG

1. Introduction

Many applications, including games, simulations, and cryptography, require random numbers. To satisfy this, Random Number Generators (RNGs) are used. While True Random Number Generators (TRNGs) [1] offer real randomness, they employ external sources of entropy and other hardware and are thus slower and computationally heavier. In contrast, PRNGs e.g. LCG, Mersenne Twister (MT19937) [2], Xorshift Generators, Linear Feedback Shift Register (LFSR) [3] etc. are software-based solutions that offer significant speed advantages and lower resource

requirements; however, their deterministic nature (because they generate the number based on predefined mathematical formula) means that the same seed and algorithm will always produce an identical sequence of numbers. The same mathematical formula leads to periodicity, where the output eventually repeats, potentially compromising security if the seed is discovered or the sequence being rendered useless once it repeats.

Although periodicity and determinism may not be problems in most applications, they could be a significant problem for those applications that demand more security and randomness. To address these vulnerabilities, researchers have explored Hybrid Pseudo Random Number Generators (HPRNGs) [4] that incorporate dynamic seeding methods-most commonly using epoch timestamps-to introduce additional entropy into the generation process. While this method addresses determinism, periodicity still afflicts it, especially when the seed time is predictable or known. Furthermore, poor or predictable seed choice (from poor timing) can subvert randomness. For instance, Origines et al. [5] demonstrated that while epoch-based methods can diversify the output sequence, the underlying periodicity of traditional PRNG algorithms remains a significant challenge.

Based on these discoveries, chaos-based methods [6,7,8,9] have become a viable remedy. PRNGs can be made more unpredictable by taking advantage of chaotic systems' complicated, non-linear behavior and sensitive reliance on initial conditions. Integrating chaotic maps, including the Logistic Map [9], Tent Map, and Chebyshev Map [8], can produce sequences with high entropy and no association, according to research by Kocarev and Lian [6], among others. Furthermore, comparative studies have shown that, in comparison to traditional approaches, chaos-based PRNGs offer gains in execution time and resource efficiency in addition to achieving greater randomization features [10].

Despite these developments, residual predictability and the possibility of cycle formation might make hybrid models-even those that use time-based seeding-ineffective in high-security settings. The necessity for more innovation in this field is emphasized in recent research. Research on parameter switching techniques [8], for instance, has demonstrated how dynamically changing the parameters of a chaotic system can obfuscate any deterministic structure and disturb periodic patterns. Furthermore, research on counter-mode deterministic generators [10] offers a standard by which to measure randomness, demonstrating that incorporating several entropy sources can greatly improve security.

In this study, we propose an enhanced HPRNG algorithm that combines dual chaotic maps with advanced bitwise mixing and periodic reseeding in addition to dynamic, time-based seeding. In order to provide a statistically superior random number output, this multidimensional technique aims to eliminate both determinism and periodicity. Our approach provides a solid solution for applications where good randomness quality is crucial, building on well-established theoretical frameworks and current experimental findings in the field.

2. Methodology

The two major problems faced by any traditional PRNGs are their deterministic nature and periodicity. Just like HPRNG using Epoch Timestamp or System Time as Entropy Source helps mitigate the deterministic problem to an extent. But if the exact time is known somehow, the model fails. At the same time the problem with periodicity remains. To improve on this, we have used the Periodic Reseeding Model as well as some other strategies [11] like chaotic maps, bit mixing, bitwise rotation, chaos map mixing.

The proposed algorithm, hereafter referred to as Chaotic Hybrid PRNG (CHPRNG), consists of several key components:

2.1 Initialization and Seed Generation

The algorithm begins by extracting a high-resolution seed x from the current system time (in nanoseconds). A normalized fractional value t is derived from x by

$$t = \frac{x \bmod 10^6}{10^6} \tag{1}$$

Ensuring $t \in [0, 1)$ which serves as the starting point for one of the chaotic maps. A secondary variable l is also initialized with the same normalized value to drive the Logistic Map. This dual initialization provides the basis for two independent chaotic sequences, each contributing distinct dynamic characteristics. One thing to notice is that, here, we rely on time.time_ns() as the primary entropy source for initial seeding as well as to periodically reseed our chaotic PRNG due to its high resolution and low overhead. However, solely relying on system time introduces vulnerabilities like timestamp predictability, snapshot weakness, hypervisor time manipulation. Although the reseeding model makes timestamp prediction harder, the system can still be vulnerable. Introducing a separate source of entropy (e.g., CPU jitter, RDRAND, TRNG chip etc.) and XOR-ing it with the timestamp during seeding and reseeding process can mitigate these vulnerabilities.

2.2 Periodic Reseeding

Since every PRNG has a period, if the algorithm is reseeded after it's assumed worst case period, the problem with periodicity is mitigated to a higher extent. To inject fresh entropy and avoid short cycles, the generator periodically updates x every T_{period} iterations, where

$$T_{period} = round\ (w \times m) \tag{2}$$

And w is a small weight (e.g., 0.01 or 1%) that is the assumed worst-case period in percentile while m defines the modulus range from a typical LCG algorithm. At the conclusion of each T_{period} (the reseeding interval), the seed x is refreshed using the current system time. This systematic reseeding injects fresh entropy, effectively breaking any emerging cycles and ensuring the randomness remains robust over prolonged sequences.

2.3 Using Dual Chaotic Maps

We integrated 2 one-dimensional chaotic maps-the Tent Map and the Logistic Map-to generate complementary chaotic sequences. The Tent Map [12], selected for its rapid convergence and computational simplicity, produces one sequence while the Logistic Map [13], renowned for its strong nonlinear dynamics, produces a second. By iterating these maps in parallel, the algorithm harnesses their independent yet complementary behaviors. On the other hand, in a comparative study of 1-D chaotic maps, Tent Map turned out to be the fastest (in terms of processing) with the Logistics Map right behind it [14]. Even though Gauss Map meet the requirement of quantum resistance [15], it is too slow and resource intensive [14] for our Algorithm. The Two chaotic maps are iterated in parallel:

- Tent Map:

$$t_{n+1} \leftarrow \begin{cases} \mu.t_n, t_n < 0.5 \\ \mu.(1-t_n), t_n \geq 0.5 \end{cases} \tag{3}$$

- Logistics Map:

where, $0 < t_0 < 1$ and $0.7 \leq \mu \leq 1$

$$l_{n+1} = r.l_n.(1-l_n) \tag{4}$$

where, $0 < l_0 < 1$ and $3.7 \leq \mu \leq 4$

Their outputs are scaled to a common integer range and combined using a bitwise XOR operation, ensuring that any residual patterns from a single chaotic source are obscured through this mixing process.

2.4 Bitwise Mixing

After scaling, the algorithm enhances randomness through a sophisticated bitwise mixing stage. First, the scaled outputs (the updated t and l are scaled to integers by multiplying by 10^6) from the Tent and Logistic maps are combined using XOR to generate an intermediate value:

$$mix_1 = \lfloor t \times 10^6 \rfloor \oplus \lfloor l \times 10^6 \rfloor \tag{5}$$

Next, both the intermediate result and the current seed are subjected to bitwise left rotations-where the rotation amounts are determined by the chaotic outputs themselves:

$$mix_2 = rotl\ (mix_1, \lfloor t \times 64 \rfloor) \oplus rotl\ (x, \lfloor l \times 64 \rfloor) \tag{6}$$

This bit-level manipulation significantly increases the sensitivity to any small variations in the chaotic sequences, thereby amplifying the overall unpredictability of the generator.

2.5 Final Update and Output Generation

Finally, using the mix, the new random number is calculated as:

$$x \leftarrow (mix_2 \oplus (x \times a))\ mod\ (m) \tag{7}$$

Where a is a constant multiplier and m is the modulus. The greater the value of m, the better the result would be. It would be best to take $m \leftarrow 2^e - 1$ where 'e' is the computer's word length and subtracting 1 from it would result in the highest prime number. (Lehmer's algorithm [16]), the higher the period. The value of x is then appended to the output sequence. This process is repeated n times to generate a sequence of pseudo random numbers.

3. Flow Chart and Implementation

The algorithm has 3 main parts i.e., Entropy Reseeding Model, Chaos Map Mixing Model and finally the generator formula. We have chosen these different unrelated models because:

"An acceptable random generator must combine at least two (ideally, unrelated) methods. The methods combined should evolve independently and share no state. The combination should be by simple operations that do not produce

results less random than their operands." [17]

3.1 Entropy Reseeding Model

The first proposed model is the Entropy Reseeding Model (Fig.1.). It works by taking a worst-case period, T_{period} and reseeding with new entropy seed after every T_{period} times generations

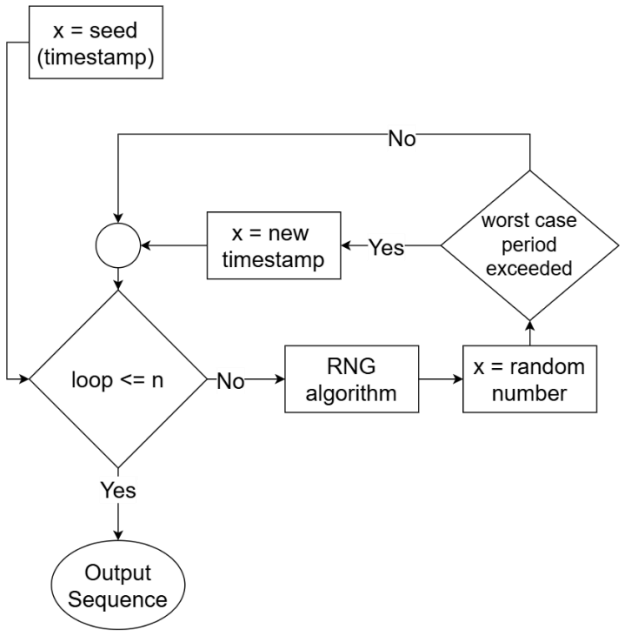

Fig. 1. Entropy Reseeding Model

3.2 Chaos Map Mixing Model

The second model is Chaos Map Mixing (Fig.2.). This step is used to introduce chaos to the generated random numbers. Here the first mixing formula is used after taking the chaotic values from the two selected chaos maps. After generating the chaotic values, the internal states t and l are respectively updated.

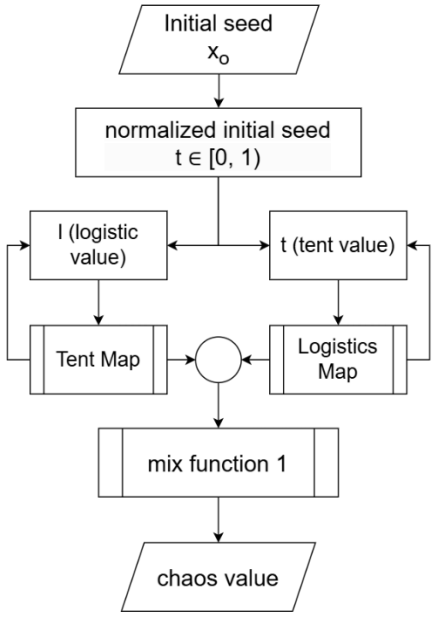

Fig. 2. Chaos Map Mixing Model

3.3 Main Generator

Finally, the two models (i.e., Entropy Reseeding Model and Chaos Map Mixing Model) are used in a modified LCG after bitwise rotation and bit mixing as shown in Fig. 3.

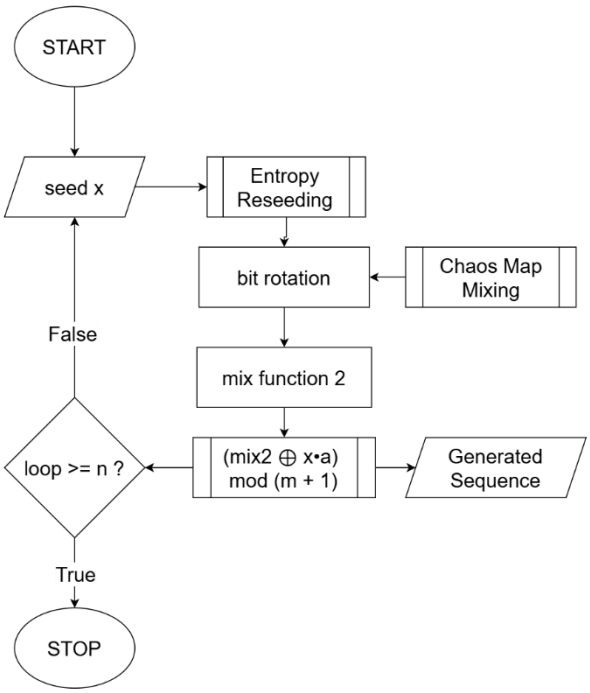

Fig. 3. Chaotic HPRNG

3.4 Code Implementation

The implementation of the algorithm in Python v3.13.2.

```
import time

def tent(x: float, mu: float = 1.7) -> float:
    return mu * x if x < 0.5 else mu * (1 - x)
def logistic(x: float, r: float = 3.99) -> float:
    return r * x * (1 - x)
def rotl(x: int, k: int, bits: int = 64) -> int:
    mask = (1 << bits) - 1
    return ((x << k) & mask) | (x >> (bits - k))

"""
Parameters:
    m (int): Modulus value for the generator.
    n (int): Number of random numbers to generate.
    a (int): Exponent for scaling the multiplier.
    w (float)  : Worst case period percentage for seed switching
Returns:
    A list of n random numbers between 0 and m.
"""
def chaos_hprng(m: int, n: int, a: int, w: float = 0.01):
    x = time.time_ns()
    worst_case_period = round(w * m)
    t = (x % 1_000_000) / 1_000_000
    l = t
    random_numbers = []
    for i in range(n):
        if i % worst_case_period == 0:
            x = time.time_ns()
        t = tent(t, 2)
        l = logistic(l, r=3.99)
        mix = int(t * 1_000_000) ^ int(l * 1_000_000)  // mix function 1
            mix = rotl(mix, int(t * 64)) ^ rotl(x, int(l * 64)) // mix function
2
        x = (mix ^ (x * a)) % m
```

```
        random_numbers.append(x)
    return random_numbers
```

4. Statistical Tests

An extensive statistical test was performed to ensure the quality and consistency of the pseudo-random numbers produced by our proposed CHPRNG. The purpose of these tests is to determine whether the output sequences follow the expected uniform distribution, which is a crucial prerequisite for simulations and applications requiring random numbers.

We implemented a rigorous testing methodology in our experimental setup that involved 1000 threads, each of which ran 10 tests with various parameter configurations (i.e., different values of m and a). As a result,$1000 \times 10 = 10000$ separate tests were produced. We created thorough statistical reports, rejection heatmaps, random number distribution profiles, and rejection rates for the tests.

4.1 Test Methodology

To rigorously evaluate the randomness of our proposed Chaotic Hybrid Pseudo-Random Number Generator (CHPRNG), we employed two primary statistical tests: the Chi-Square Test [18] and the Kolmogorov–Smirnov (KS) Test [19]. These tests assess different aspects of the generated sequences to ensure they meet the criteria for uniformity and randomness. In addition to classical uniformity testing, we evaluated the reseeding model using two structural diagnostics i.e. Autocorrelation Analysis and Cycle-Length Detection

4.1.1 Chi-square Test:

This test assesses how well the observed frequency distribution of numbers aligns with the expected uniform distribution. The Chi-square statistic is computed as:

$$X^2 = \sum_i^n \frac{(O_i - E_i)^2}{E_i} \tag{8}$$

Where O_i and E_i are the observed and expected frequencies for the iii-th bin, respectively. A p-value is derived from the statistic, and if the p-value is below the significance level $\alpha = 0.05$ the test rejects the hypothesis that the distribution is uniform.

[N.B. It's important to note that the Chi-Square Test requires a sufficiently large sample size to ensure that the expected frequency in each bin is adequate, typically at least 5, to validate the test's assumptions.]

4.1.2 Kolmogorov–Smirnov (KS) Test.:

This non-parametric test compares the empirical distribution of the generated numbers to a theoretical uniform distribution. It computes the maximum deviation (D) between the two distributions and evaluates the null hypothesis H_0 that the numbers are uniformly distributed. The KS Test is advantageous because it does not require binning of data and is effective for continuous distributions. However, it is more sensitive near the center of the distribution than at the tails.

4.1.3 Autocorrelation Analysis

To empirically show that the reseeding model mitigates the periodic nature, we conducted an autocorrelation test where we compute the sample autocorrelation function (ACF) up to a maximum lag L. For a sequence $x_i{}_{i=0}^{N-1}$ with sample mean $\bar{x}$ is,

$$\bar{x} = \frac{1}{N}\sum_{i=0}^{N-1} x_i \tag{9}$$

The lag-k autocorrelation is,

$$\rho(k) = \frac{\sum_{i=0}^{N-k-1}(x_i - \bar{x})(x_{i+k} - \bar{x})}{\sum_{i=0}^{N-1}(x_i - \bar{x})^2}, k = 0, 1, \dots, L \tag{10}$$

4.1.4 Cycle Length Distribution

To verify whether the reseeding model is effective in improving upon periodicity and disrupting long cycle, we have conducted a cycle-length distribution test. This test does not find the period of the generated sequence (i.e. does not find the first recurrence of the whole series), but finds the recurrence of a generated value over a block. So shorter cycle length indicates more disruption of long deterministic cycles (The original non-reseeded chaotic generator may enter long loops, repeating long, fixed patterns) as well as improved randomness across block. Here we partition the full output stream into non-overlapping blocks, then for each block apply a first-repetition search.

4.2 Implementation and Test Procedures

The testing framework was implemented in Python v 3.13.2 using the SciPy (v1.14.1) library for statistical analysis. The data has been stored in SQLite database. Key steps in the process include:

- **Normalization:** The raw output from the PRNG is normalized to the [0, 1] interval to facilitate comparison with the uniform distribution.
- **Test Execution:** For each algorithm, a sequence of N random numbers is generated and normalized. The Chi-square and KS tests are then applied to this sequence. The entire testing routine is executed repeatedly to compute average test statistics and execution times.
- **Data Aggregation:** Test results-including the Chi-square statistic, KS statistic, corresponding p-values, and execution times-are stored. Aggregation scripts then compile rejection counts and performance metrics across multiple threads.
- **Visualization:** Results are visualized using Python libraries such as Pandas and Seaborn, providing a clear comparative overview of the rejection rates and execution times for each algorithm.

We computed the autocorrelation function (ACF) up to lag 200 using a Fast Fourier Transform (FFT)-based estimator. The ACF was normalized by removing the sample mean and dividing by the zero-lag variance. Finally, to identify short-term repetition, each 'block_size'-sample block was scanned for the first repeat of any output value. Both the Autocorrelation and Cycle‑length Distribution tests were done with the parameters from Table 1. below:

Table 1. No-reseeding vs Reseeding model test parameters.

Parameter name	Values
Modulus (m)	2^{16}-1
Sequence Length (N)	10,000,000
Multiplier (a)	7
Worst case period factor (w)	0.01
Block size (block_size)	10,000

5. Results

We conducted the tests on our proposed CHPRNG and base model algorithm, HPRNG. We compiled the result based on 4 parameters, i.e., Rejection Rate, Rejection Heatmap, Random Number Distribution, and Test Statistics.

[**N.B.** in the result figures, HPRNG has been used as 'hybrid' and CHPRNG as 'tent 3' or 'chprng']

5.1 Rejection Rate

The rejection rate measures the proportion of generated sequences that fail the uniformity tests at a significance level of $a = 0.05$ The results of the Chi-squared and Kolmogorov–Smirnov (KS) tests for both the proposed CHPRNG and the baseline HPRNG, some other algorithms (Mersenne Twister, PCG, Well512a) are summarized in Fig. 4.

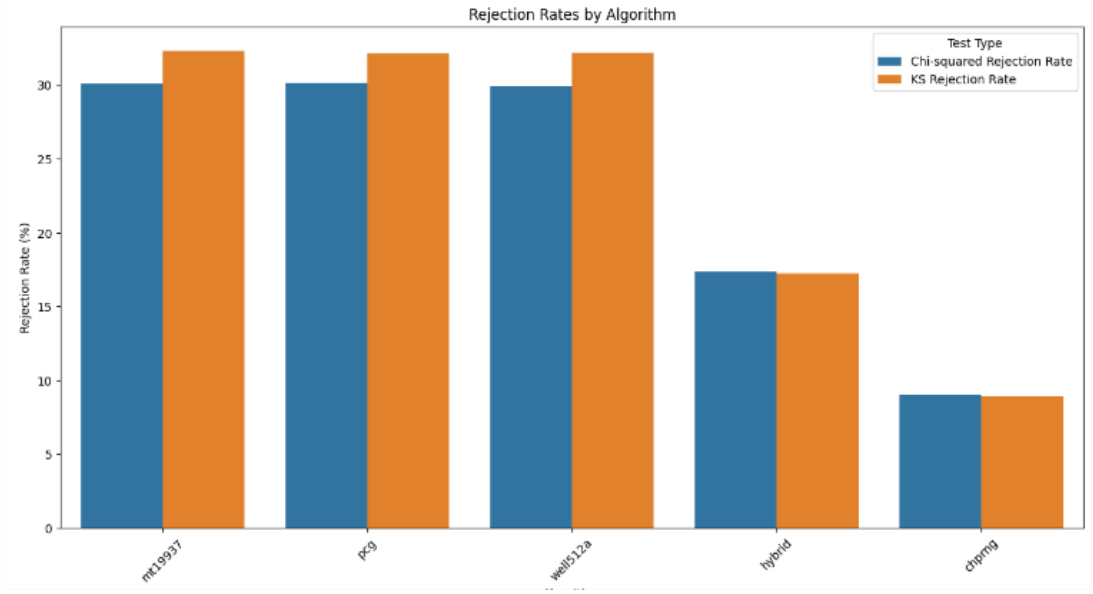

Fig. 4. Rejection Rates for different algorithm. Blue is the Chi2 rejection rate and Yellow is the KS Test rejection rate. The rate of rejection is on the y-axis and the algorithms on the x-axis. The algorithms from left to right respectively are: Mersenne Twister, PCG, Well512a, HPRNG and CHPRNG.

For the HPRNG algorithm, the rejection rates were 17.18% and 17.11% for the Chi-squared and KS tests, respectively. In contrast, the proposed CHPRNG achieved lower rejection rates of 8.67% (Chi-squared) and 9.00% (KS). This indicates an improvement in statistical uniformity and randomness quality in the proposed algorithm compared to the baseline:

Table 2. Rejection Rate

Algorithm	Chi^2	KS	Total Tests
MT19937	30.08%	32.14%	10000
PCG	30.13%	30.24%	10000
WELL512	29.92%	31.12%	10000
HPRNG (base)	17.33%	17.23%	10000
CHPRNG (proposed)	08.67%	08.94%	10000

The results from Table 2 suggest that CHPRNG exhibits enhanced randomness properties, as evidenced by lower rejection rates across both statistical tests i.e., approx. 49.97% and 48.12% improvement in Chi-square and KS Tests respectively compared to the base model of HPRNG.

5.2 Rejection Heatmap

To further illustrate how rejection rates vary with different sample sizes, we present the rejection heatmap in Fig.5. Each row corresponds to a specific m value (ranging from 10^2 to 10^6), while each column represents a particular combination of algorithm (HPRNG or CHPRNG) and statistical test (Chi-square and KS). A cell shaded in red (=1) indicates that the corresponding test rejects the null hypothesis, whereas a blue cell (=0) signifies no rejection. We can see that the base model fails at lower values of m, and in some other higher values as well. We noticed this trend in most of the test runs across different parameters. Whereas our PRNG performs pretty uniformly across all values of m.

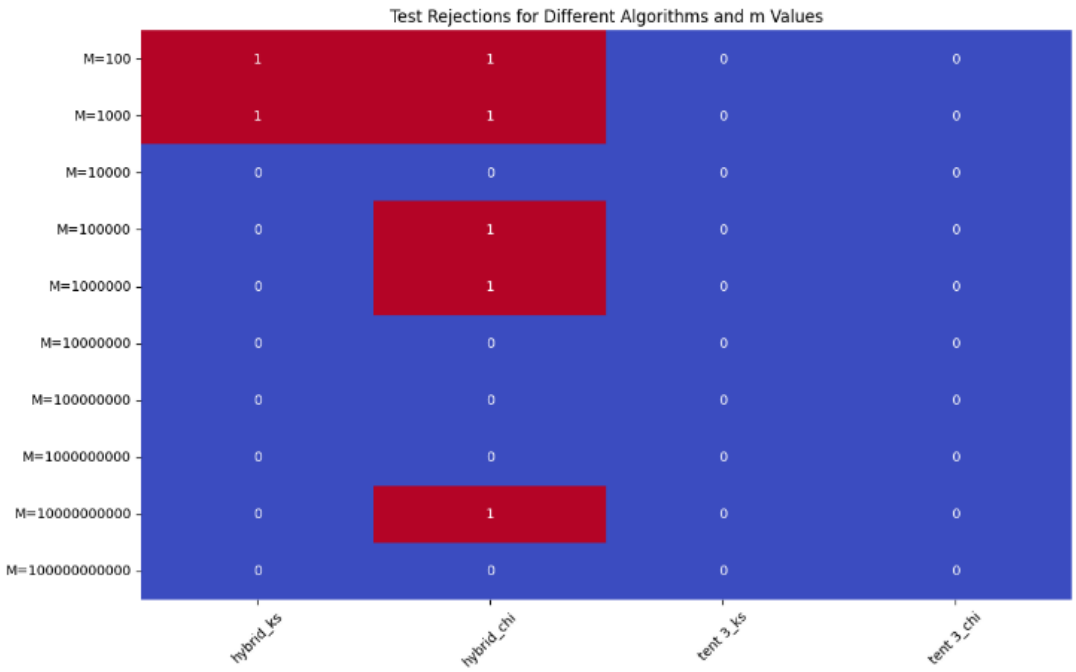

Fig. 5. Rejection Heatmap across different values of m

This visualization highlights that CHPRNG maintains more consistent performance across a range of sample sizes, whereas HPRNG experiences uniformity issues at several points.

5.3 Random Number Distribution

The Fig. 6. and Fig. 7 compares the distributions of generated random values for HPRNG (left) and the proposed CHPRNG algorithm (right) across a range of sample sizes. Each box represents the interquartile range (IQR) of the generated values, with the whiskers extending to the minimum and maximum observations.

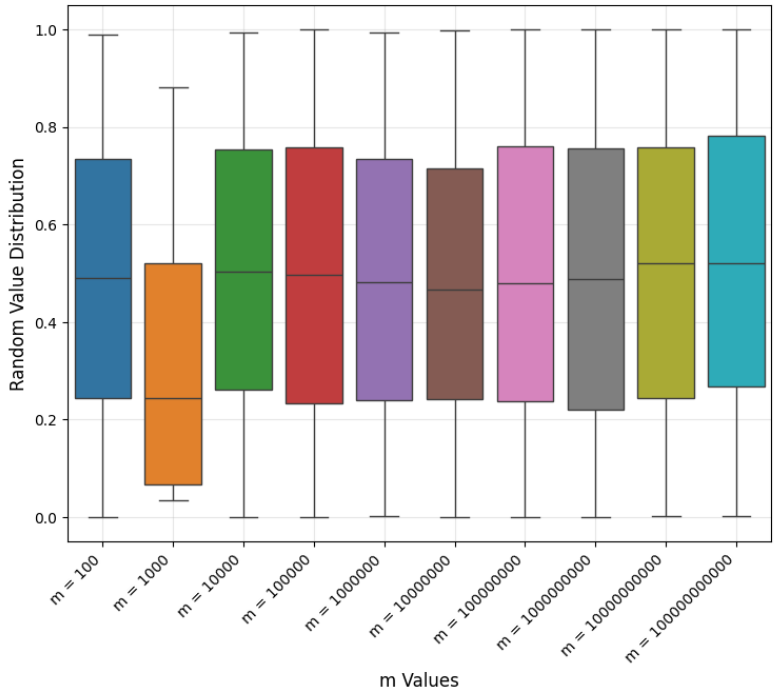

Fig. 6. Generated Number Distribution for base model HPRNG

For small sample sizes, the HPRNG shows slightly more variation and potential clustering, as indicated by wider boxes or outliers. Meanwhile, CHPRNG maintains a comparatively uniform spread even at lower sample sizes. As m increases, both algorithms' distributions begin to fill the interval more consistently, reflecting a tendency toward uniformity.

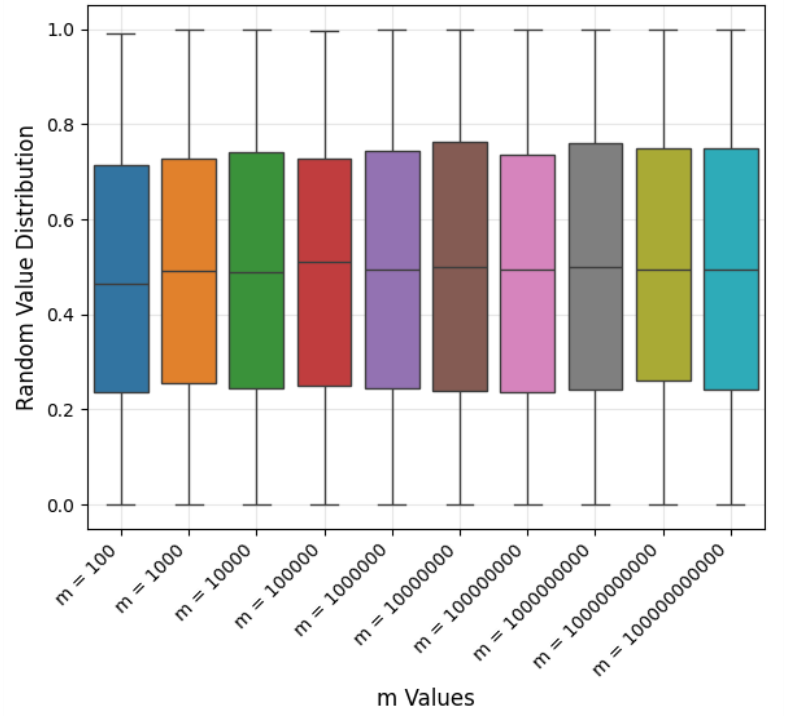

Fig. 7. Generated Number Distribution for proposed CHPRNG model

However, CPRNG maintains a more stable distribution across the entire range of m-values that supports the findings from the rejection rate and heatmap analyses.

5.4 Tests Statistics

The Fig. 8. and Fig. 9 respectively shows the Kolmogorov–Smirnov (KS) and Chi-squared test statistics for the baseline HPRNG (colored blue) and the proposed CHPRNG algorithm (colored red) across various sample sizes (m). On the left, the KS statistic is plotted against the value of m, while on the right, the Chi-squared statistic is depicted. A smaller statistic in both tests indicates a closer fit to the theoretical uniform distribution.

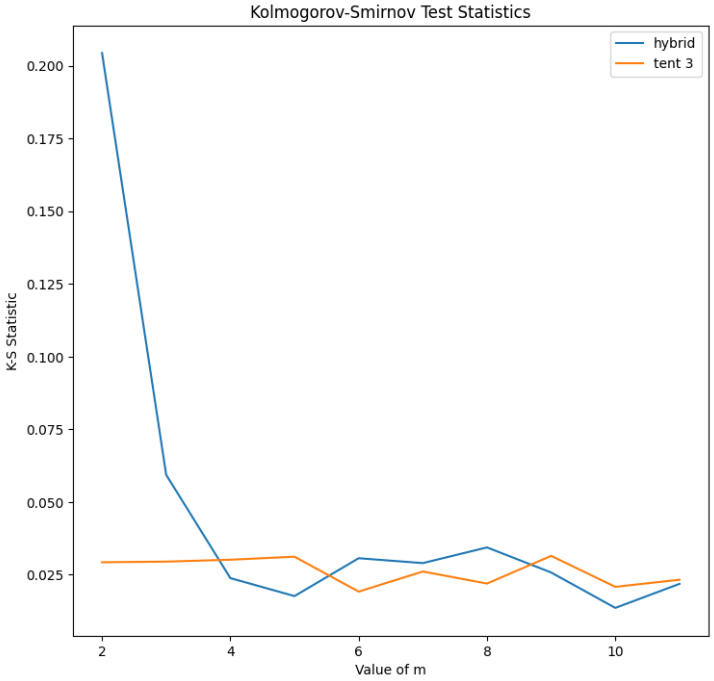

Fig. 8. KS Tests Statistics

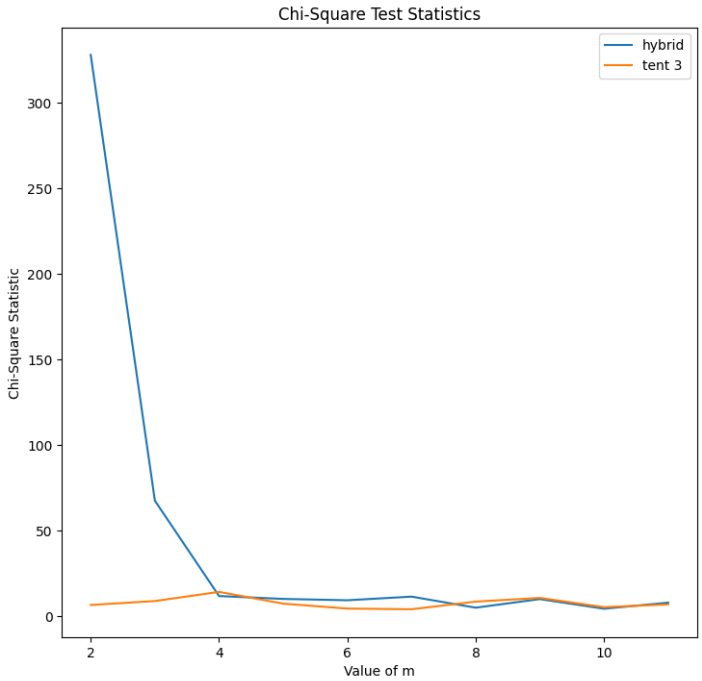

Fig. 9. Chi-squared Tests Statistics

For lower values of m, the base algorithm exhibits larger KS and Chi-squared statistics, suggesting less uniformity. As m increases, both algorithms' test statistics decrease, reflecting an overall improvement in distribution uniformity. Nevertheless, proposed algorithm consistently yields lower or comparable test statistics compared to HPRNG, indicating more robust performance across the examined sample sizes.

5.5 Autocorrelation and Cycle-length Distribution

Table 3. ACF for "no–reseed" versus "with–reseed".

	No Reseeding	With Reseeding
Mean	0.000082	0.001139
Std. Deviation	0.000056	0.001134

The Table 3 summarizes the first-lag through 200-lag statistics. Here, we can see that, although both series shows ACF values tightly clustered around zero, periodic reseeding reduces the mean by approximately 32%. This suggests that reseeding mitigates (but not entirely removes) the short-range serial dependence.

Table 4. Cycle - length statistics (block size = 2 000)

	No Reseeding	With Reseeding	Drop
Count	100	100	
Mean	199.0	145.9	26.68%
Median	168.0	125.5	25.29%
Max	690	532	22.89%

Next, we scanned each 2 000 - output block for the first repetition of any state value, and recorded the cycle-length. The statistics are summarized in Table 4, and the cycle lengths are provided below along with a histogram in Fig. 10. .

```
   === Cycle Lengths (No Reseed) ===
[4, 238, 288, 4, 54, 256, 61, 173, 217, 388, 478, 109, 394, 409, 120, 399, 1, 176,
245, 349, 78, 93, 55, 7, 84, 375, 32, 326, 317, 308, 432, 209, 20, 28, 385, 120, 177,
335, 690, 351, 155, 56, 82, 470, 145, 36, 312, 41, 68, 19, 294, 283, 27, 142, 213,
65, 311, 415, 311, 149, 267, 163, 198, 74, 119, 186, 274, 175, 253, 21, 99, 9, 462,
179, 153, 676, 159, 112, 10, 176, 161, 16, 143, 158, 344, 212, 139, 33, 408, 77, 89,
53, 43, 353, 523, 98, 340, 240, 117, 213]

   === Cycle Lengths (With Reseed) ===
[92, 10, 303, 102, 82, 78, 56, 464, 83, 92, 3, 65, 157, 195, 70, 152, 106, 124, 411,
532, 59, 211, 74, 158, 5, 222, 170, 69, 259, 184, 14, 340, 58, 49, 303, 344, 17, 229,
127, 221, 286, 124, 20, 199, 192, 193, 260, 307, 109, 78, 343, 213, 49, 66, 45, 251,
13, 5, 334, 300, 450, 160, 67, 68, 214, 159, 63, 236, 148, 189, 161, 98, 222, 59,
61, 28, 94, 177, 4, 181, 33, 18, 180, 217, 57, 202, 186, 161, 60, 177, 34, 139, 271,
75, 17, 73, 2, 6, 13, 154]
```

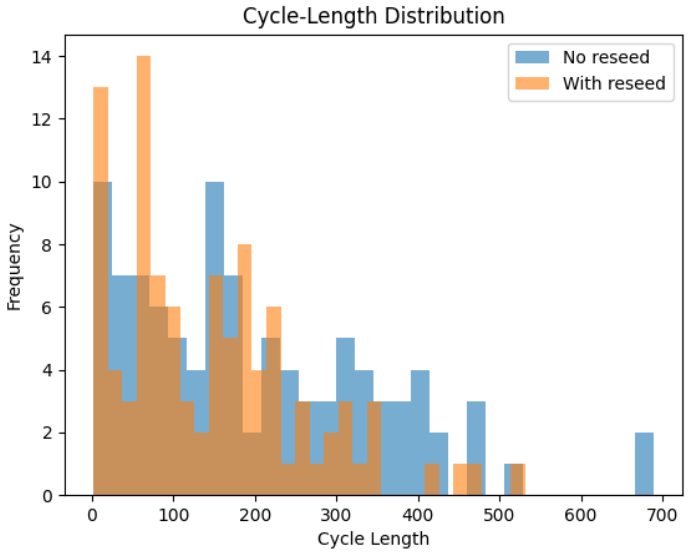

Fig. 10. Cycle - length histograms (block size = 2 000).

Here, we can see that the periodic reseeding reduces both the mean and max cycle lengths with a drop of 26.68% and 22.89% in mean and max values respectively. Thus, the periodic reseeding helps escape the attractors or repeating patterns the Chaotic PRNGs can lock into by periodically disrupting the cycle and introducing fresh entropy.

6. Observation and Discussion

The proposed algorithm implements a chaos ‑ based PRNG by dynamically integrating two well ‑ known chaotic maps [14], the tent map [12] and the logistic map [13], within a periodic reseeding framework. Initially, a high-resolution system clock (using nanoseconds) is employed to provide a dynamic seed, ensuring that the starting value is constantly refreshed. This approach, as highlighted in related works on entropy ‑ based seeding techniques, enhances unpredictability and minimizes the risk of repetitive sequences

Within the generation loop, the algorithm applies the tent map and the logistic map to two separate state variables. Their outputs are then scaled to a common resolution and are combined using a bitwise XOR operation. This mixing step not only works on the inherent periodicity present in digital implementations of individual chaotic maps but also amplifies the system's sensitivity to initial conditions-a key attribute that underpins chaotic behavior

Furthermore, the periodic update of the seed variable (every worst-case period determined by a fraction of the modulus m) ensures that the generator is periodically reinitialized. This measure disrupts potential cycle formation or periodicity of traditional PRNGs. Although reseeding reduces, but cannot mathematically guarantee, infinite non-periodicity; it does inject fresh entropy at controlled intervals. But by FFT-based autocorrelation analysis and cycle-length detection, we can see that reseeding can improve upon the random number sequence generated by the RNG by disrupting the natural cycle as well as make it more unpredictable.

6.1 Choice of Chaotic Map Combination

- Complementary Strengths: At their fully chaotic parameter values ($\mu = 2$ for Tent, $r = 4$ for Logistic), these maps are topologically conjugate, producing equivalent chaotic dynamics under a smooth transformation. In practice, each has distinct practical strengths: the Tent map generates high-speed, near-uniform distributions, whereas the Logistic map provides strong stretching and folding dynamics and deep sensitivity to initial conditions
- Enhanced Uniformity and Distribution Quality: Studies [20] show that the Tent map can outperform even established PRNGs such as Mersenne Twister in distribution uniformity and speed. When paired together, the resulting hybrid PRNG tends to have a more uniform statistical structure and flattened distribution histogram than either map individually, minimizing statistical bias [21].
- Greater Effective Lyapunov Exponent: Hybrid systems tend to exhibit positive Lyapunov exponents greater than those of the component parts. That translates into greater exponential divergence of nearby trajectories, enhancing unpredictability [22].
- Removal of Individual Weaknesses: The logistic map alone can be subject to periodic islands or shorter cycles under finite precision. The tent map alone can collapse to fixed or short-period states (e.g., zero) if not perturbed carefully. Combined, these weaknesses are effectively masked errors or drifts in one map are scrambled by the other via the mixing XOR step, making cycles less likely to align or reinforce.
- Empirical Success with Composition-Based Approaches: The deep‑zoom [23] composition of logistic + tent (which removes leading digits after the fraction) significantly improved randomness and passed statistical test suites where single‑map PRNGs often failed. Other studies [22] confirm that coupling the two maps or using composite maps combining both generates sequences with higher entropy, stronger diffusion, and better cryptographic properties than either alone.

6.2 Caveats and limitations

- Digital degradation due to finite precision: In real digital systems, the implementation of both maps is based on finite-precision arithmetic, which inevitably leads to a finite number of representable states. Trajectories therefore necessarily converge into periodic cycles, regardless of the original chaotic behavior.
- Known cycle structures: Studies show that logistic maps in finite precision display densely- distributed, yet ultimately limited, periodic orbits, whose lengths depend in a predictable way on the modulus or word length employed.
- Tent map periodic collapse: Digital Tent maps suffer from cycle onset and ultimate convergence (usually to zero), unless specially perturbed-for instance, with hardware XOR perturbation of the least significant bits to destroy short cycles.
- Parameter-range fragility: The chaotic dynamics in both maps are restricted to certain, well-defined parameter ranges ($r \approx 3.57$–4 for the logistic map, $\mu \approx 2$ for the tent map). Already small parameter deviations or implementation errors can cause the system to go over into periodic orbits or even converge to fixed points.
- Predictability issues: Since such maps (particularly the logistic map) are simple and deterministic, subtle structural properties in their numeric or symbolic dynamics can be used in attacks on chaos‑based generators, unless appropriate mixing and perturbation steps are incorporated.

7. Conclusion

This study works on an improved chaos-based pseudo random number generator that uses a synergistic composition of tent and logistic maps and a dynamic reseeding method based on system clock data. Due to periodic seed changes and bit-level mixing, this method is immune to common setbacks of digital chaotic processes (such cycle creation and periodicity). Additionally, experimental comparisons using Kolmogorov-Smirnov, Chi-square, and other statistical tests confirm that this algorithm offers a lower rejection rate and better uniformity than conventional hybrid models.

From the tests, we found improved randomness and uniformity as the rejection rates were roughly half. Although the extra processing (two map iterations, bit mixing, and periodic reseeding) does add computational overhead compared to very simple PRNGs, our choice of simple and complementary 1-D chaotic maps for efficiency, e.g., the tent map which is known to be extremely fast compared to other maps, and basic bitwise operations, the generator remains relatively lightweight for a chaos-based PRNG. But like any other chaos-based designs, the security of our PRNG is supported only by empirical testing, not by formal proofs.

The design demonstrates that leveraging chaotic maps, coupled with periodic reseeding, can yield a lightweight yet robust PRNG suitable for applications in resource-constrained environments. In conclusion, our model substantially improves statistical performance (supported by empirical testings) at the cost of extra computation. It fills a niche for high-randomness demands in limiting environment, but the runtime trade-offs should be considered in deployment.

References

[1] M. Stipčević and Ç. K. Koç, "True Random Number Generators," Open Problems in Mathematics and Computational Science, p. 75–315, 2014.

[2] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator.," ACM Transactions on Modeling and Computer Simulation (TOMACS)., vol. 8, no. 1, pp. 3-30, 1998.

[3] L. Oumouss, A. Younes, A. Ahmed and A. Rguibi, "Cryptographically robust pseudo-random binary sequence generator based on the integration of LFSRs and CAs," in 2024 International Conference on Circuit, Systems and Communication (ICCSC), Fes, 2024.

[4] M. M. Rahman and T. Ahmed, "The Hybrid Pseudo Random Number Generator," International Journal of Hybrid Information Technology, vol. 9, no. 7, pp. 299-312, 2016.

[5] D. V. Origines, A. M. Sison and R. P. Medina, "A Novel Pseudo-Random Number Generator Algorithm based on Entropy Source Epoch Timestamp," in 2019 International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, 2019.

[6] J. Amigó, "Chaos-Based Cryptography.," in Intelligent Computing Based on Chaos, L. Kocarev, Z. Galias and S. Lian, Eds., Berlin, Springer Berlin Heidelberg, 2009, pp. 291-313.

[7] S. Araki, J.-H. Wu and J.-J. Yan, "A Novel Design of Random Number Generators Using Chaos-Based Extremum Coding," IEEE Access, vol. 12, pp. 24039-24047, 2024.

[8] İ. Öztürk and R. Kılıç, "A new pseudo random number generator based on Chebyshev maps and parameter switching," in 2018 6th International Conference on Control Engineering & Information Technology (CEIT), Istanbul, 2018.

[9] T. H. Teo, M. Xiang, M. Elsharkawy, H. R. Leao, M. J. Andrew Calderon, J. L. Lee, S. A. Bin Rosli, H. Y. See and E. Y. Lim, "Design and Implementation of a Logistic Map-Based Pseudo-Random Number Generator on FPGA," in 2024 IEEE 17th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Kuala Lumpur, 2024.

[10] R. I. Caran, "Comparative Analysis Between Counter Mode Deterministic Random Bit Generators and Chaos-Based Pseudo-Random Number Generators," in 2024 International Conference on Development and Application Systems (DAS), Suceava, 2024.

[11] V. Chugunkov, V. A. Gulyaev, E. A. Baranova and V. I. Chugunkov, "Method for Improving the Statistical Properties of Pseudo-random Number Generators," in 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg and Moscow, 2019.

[12] W. F. H. Al-shameri and M. A. Mahiub, "Some Dynamical Properties of the Family of Tent Maps," International Journal of Mathematical Analysis, vol. 7, no. 29, pp. 1433-1449, 2013.

[13] R. M. May, "Simple Mathematical Models with Very Complicated Dynamics," Nature, vol. 261, no. 5560, pp. 459-467, 1976.

[14] M. S. Kaya and K. İnce, "Benchmarking Various 1D Chaotic Maps For Lightweight Pseudo-Random Number Generation," in 2024 8th International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, 2024.

[15] Kumar and A. Mishra, "Evaluation of Cryptographically Secure Pseudo Random Number Generators for Post Quantum Era," in 2022 IEEE 7th International conference for Convergence in Technology (I2CT), 2022.

[16] D. H. Lehmer, "Mathematical Methods in Large-Scale Computing Units," in Proceedings of the Second Symposium on Large Scale Digital Calculating Machinery, 1951.

[17] W. T. Vetterling, Numerical Recipes - The Art of Scientific Computing - 3rd Edition, Cambridge University Press, 1986.

[18] "Chi-Square Test," in The Concise Encyclopedia of Statistics, New York, Springer New York, 2008, pp. 77-79.

[19] "Kolmogorov–Smirnov Test," in The Concise Encyclopedia of Statistics, New York, Springer New York, 2008, pp. 283-287.

[20] L. A. Demidova and A. V. Gorchakov, "A Study of Chaotic Maps Producing Symmetric Distributions in the Fish School Search Optimization Algorithm with Exponential Step Decay," Symmetry, vol. 12, no. 5, p. 784, 2020.

[21] R. Parvaz and M. Zarebnia, "A combination chaotic system and application in color image encryption," Optics & Laser Technology, vol. 101, pp. 30-41, May 2018.

[22] M. Alnajim, E. Abou-Bakr, S. S. Alruwisan, S. Khan and R. A. Elmanfaloty, "Hybrid Chaotic-Based PRNG for Secure Cryptography Applications," Applied Sciences, vol. 13, no. 13, p. 7768, 2023.
[23] J. P. d. V. Alvarenga, J. Machicao and O. M. Bruno, Chaotical PRNG based on composition of logistic and tent maps using deep-zoom, 2021.

Authors' Profiles

Md Mahbubur Rahman received the B.Sc. Engg. in computer science and engineering from Patuakhali Science and Technology University, Patuakhali, Bangladesh, in 2011. He went on to earn the M.Sc. Engg. in computer science and engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2018. He is currently Assistant Professor in the Department of Computer Science and Information Technology (CSIT) at Patuakhali Science and Technology University, Patuakhali, Bangladesh. He has served as author on numerous peer-reviewed papers published by IEEE, Springer, and other national and international journals. His research interests include wireless sensor networks, cybersecurity, machine learning, data science, and the Internet of Things. Mr. Rahman is a member of the Bangladesh Computer Society and serves on its program-planning committees. He has contributed to the technical program committees of several national and international conferences.

Hind Biswas was born in Khulna, Bangladesh, in 2004. He is currently pursuing the B.Sc. Engg. in computer science and engineering at Patuakhali Science and Technology University, Patuakhali, Bangladesh. He has worked as a Senior Web Developer on a freelance basis, providing web development services to several companies. He has served as author on peer-reviewed articles. His research and development interests include machine learning, data science, cybersecurity, and quantum mechanics.

Chinmay Bepery is Professor in the Department of Computer Science and Information Technology (CSIT) at Patuakhali Science and Technology University, Patuakhali, Bangladesh. He has authored numerous research articles published by IEEE, Springer, and other peer-reviewed venues. His research interests include machine learning, algorithms, image processing, bioinformatics, and the Internet of Things. Professor Bepery is a life-time member of the Bangladesh Computer Society and a member of the IEEE. He has served on editorial and program committees for several national and international conferences and journals.

Md Atikur Rahman received the B.Sc. Engg. in computer science and engineering from Patuakhali Science and Technology University, Patuakhali, Bangladesh, in 2008, and the M.Sc. Engg. in computer science and engineering from the University of Dhaka, Dhaka, Bangladesh, in 2014. He is currently pursuing the Ph.D. in information and communication technologies at the International University of Malaya-Wales, Penang, Malaysia. He is Professor in the Department of Computer Science and Information Technology (CSIT) at Patuakhali Science and Technology University, Patuakhali, Bangladesh. He has published extensively in IEEE, Springer, and other national and international journals. His research interests include cloud computing, big data analytics, human–computer interaction, blockchain, and augmented reality. Mr. Rahaman is a member of the Bangladesh Computer Society and participates in its curriculum-development and accreditation committees. He has also served on the program committees of several international conferences.

Md. Moshiur Rahman received his B.Sc. Engg. in Computer Science and Engineering from Patuakhali Science and Technology University, Patuakhali, Bangladesh in 2011, and his M.Sc. in Computer Science and Engineering from the Islamic University of Technology (IUT), Bangladesh. He is currently pursuing his Ph.D. in Computer Science and Engineering at Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh.He is currently an Assistant Professor in Computer Science and Engineering (CSE) at Bangladesh Open University, Dhaka, Bangladesh. Mr. Rahman has authored numerous peer-reviewed publications in national and international journals. His research interests include deep learning, data science, cybersecurity, and health informatics. He is an active member of the Bangladesh Computer Society and has served on technical program committees of national and international conferences. Mr. Rahman's work focuses on advancing both practical and theoretical aspects of modern computing, emphasizing secure, intelligent, and data-driven technologies.

How to cite this paper: Md Mahbubur Rahman, Hind Biswas, Chinmay Bepery, Md Atikqur Rahaman, Md. Moshiur Rahman, "Enhanced Hybrid Pseudo Random Number Generator – Chaotic HPRNG", International Journal of Mathematical Sciences and Computing(IJMSC), Vol.11, No.3, pp. 32-46, 2025. DOI: 10.5815/ijmsc.2025.03.03