PRaCto: Pseudo Random bit generator for Cryptographic application

Saiyma Fatima Raza[1], **Vishal R Satpute**[1]
[1]Visvesvaraya National Institute of Technology,
Nagpur, India
[e-mail: saiyma.fatima.raza@gmail.com; vishal1210@yahoo.com]
Corresponding author : Saiyma Fatima Raza

Received March 15, 2018; revised June 12, 2018; accepted July 5, 2018; published December 31, 2018

Abstract

Pseudorandom numbers are useful in cryptographic operations for using as nonce, initial vector, secret key, etc. Security of the cryptosystem relies on the secret key parameters, so a good pseudorandom number is needed. In this paper, we have proposed a new approach for generation of pseudorandom number. This method uses the three dimensional combinational puzzle Rubik Cube for generation of random numbers. The number of possible combinations of the cube approximates to 43 quintillion. The large possible combination of the cube increases the complexity of brute force attack on the generator. The generator uses cryptographic hash function. Chaotic map is being employed for increasing random behavior. The pseudorandom sequence generated can be used for cryptographic applications. The generated sequences are tested for randomness using NIST Statistical Test Suite and other testing methods. The result of the tests and analysis proves that the generated sequences are random.

Keywords: Rubik cube, statistical tests, pseudorandom number generator, cryptographic hash.

1. Introduction

With the advent of Internet and technology, security is an essential requirement in communication. Security protocols make use of cryptographic techniques for achieving security. Many cryptographic algorithms require random binary sequences to be used as initial seed, nonce, secret key, session keys etc. Encryption requires secret key and depending on the type of encryption, the number of keys required vary. Block ciphers and stream ciphers require random seed for encryption. Along with encryption, keyed hash functions are used for providing data integrity and authenticity also require secret key. Simulations, gaming, cryptography, etc. require random numbers but in this paper we are focusing on random numbers for cryptographic applications. Generation of true random numbers requires a true source of randomness like thermal noise in circuits, atmospheric noise, electromagnetic phenomena, etc. An inaccurate randomness source can compromise a strong cryptographic primitive. Hence cryptographic primitive require unbiased and uncorrelated random number for providing effective security. As many cryptographic algorithms do not have a true random source so these algorithm use pseudorandom number generators (PRNG). PRNG uses low entropy source of input to generate streams of bits that are indistinguishable from random numbers. Lots of work has been done on the generation of pseudorandom number generation. Most of the methods present in use now are based on shift registers i.e. linear or non-linear shift registers, linear congruential methods. These methods are not secure enough due to the linearity in the generation method. Most of the PRNG uses existing cryptographic primitives like block ciphers, stream ciphers, and hash functions, as the security of these primitive will contribute to the security of PRNG. Among other methods proposed for PRNG based on chaos theory using stationary and non-stationary chaotic maps.

Some of the proposed algorithms for security are not secure even though the authors claim them to be secure. Earlier methods for generating pseudorandom numbers include use of block ciphers, stream ciphers, hash functions, chaos theory etc. cryptographic primitive have advantage of having already build blocks available, but they require heavy computations.

In the work presented, we propose a pseudorandom bit generator using Rubik's Cube, a three dimensional puzzle to produce random bits using the mixing property of the puzzle. Some application of Rubik's cube includes image encryption, secret key management in wireless network and other encryption systems.

The major contribution of this work can be summed as below:

1. The work proposed introduces a new approach for generation of pseudorandom bits.
2. Rubik's Cube, a 3D puzzle has been used for the generation of bits.
3. The method integrates chaos theory, standard hashing algorithm and a new method Rubik's Cube scrambler.
4. A dynamic Look-up table for seed value mapping has been incorporated.

Rest of this paper is arranged as follows. Section 2 discusses some of the requirement and properties of random number generator. Different methods for generation of pseudorandom number are discussed in Section 2 along with introduction to Rubik's Cube, Chaos theory and hashing algorithm. The proposed pseudorandom bit generator is explained in Section 3.

NIST statistical test suite used for testing of random and pseudorandom numbers is discussed in Section 4. The result of statistical tests is given in Section 5. Section 6 concludes the paper.

2. Related Work

2.1. Random Number Generator

The application of random number generator has been discussed previously. Classification of random number can be done in three classes: True Random Number Generator (TRNG), Pseudo Random Number Generator (PRNG), and Hybrid Random Number Generator (HRNG).

A TRNG works on the principle of extracting randomness from natural rather non-deterministic sources it is generally a physical process. It utilizes the unpredictability of these processes and the entropy of the generator output depends on the entropy of the source. Some of the proposed generator uses analog circuits for noise like behavior [1], noise from semiconductor [2], oscillator sampling technique [3], chaotic systems [4-8], [11-19] and time elapsed between the emission of particle during radioactive decay [20] and variations in disk drive speed [21]. Most of the random number generator utilize hardware [22] and therefore are many times referred to as hardware-based generator. The output is noise like with uncorrelated output.

On the other hand, PRNG uses deterministic algorithms to generate random numbers. The output of PRNGs is not truly random but the properties of sequence generated approximate the property of a random like sequence. PRNGs require an initial seed as input, which can be used to determine the output of the PRNG as it being a deterministic process.

Due to the requirement of random seed for PRNG, HRNG uses seed generator. The seed generator is expanded from a TRNG. A seed generator uses hardware interface as mouse movements, key press measure, disk read timings, etc.

As true random numbers are difficult to generate, pseudorandom numbers are considered. They have properties similar to random number and cannot be easily distinguished from a true random number. Some of the properties a PRNG require are, first is pseudo randomness i.e. the output should appear random like, forward security requires that at any state in the generator the adversary cannot predict the previous output given the current state. Whereas backward security implies that if the adversary knows the current state, he cannot guess the output of the next state.

Further in this section work on TRNG and PRNG has been discussed. As mentioned earlier a true random source is difficult to realize and require a physical source of true randomness. Some TRNG has been proposed in the past.

In [20] random numbers are generated by the inherent uncertainty in the quantum mechanical laws. Integrated ICs are also used to generate random numbers. These ICs can provide random number at a very high rate, as they are dedicated hardware generator. Some of the TRNG uses cryptographic primitives. Linux random number generator [23] uses various kernel events (key press statistics, mouse movements, interrupts) as randomness source and uses an LFSR pool along with hash function.

TRNG based on chaotic systems have been proposed [9], [10]. The output of a chaotic system can be determined if the initial condition is known, but in long term it can be considered unpredictable due to the sensitivity of chaotic systems to initial condition. In [10] a TRNG using mouse movement is proposed. To randomize the regular mouse movement a chaotic hash function is used. In [9] a novel bit generator based on double scroll attractor is proposed in a chaotic system. The sequence generated passes the FIPS 140-1 and Diehard test suite, which makes it suitable for use in cryptographic applications.

In [11] Andrecut proposed logistic map based random number generator and compared it with the congruential random generators that are periodic, with the logistic RNG. Szczepanski et al. [12] proposed PRNG based on discrete chaotic dynamical systems. Li et al. [13] proposed a generator method for stream cipher using couple chaotic systems.

Hauping et al. [14] proposed a PRNG using one way coupled chaotic map lattice. Similarly Li et al. [15] proposed a chaotic one way coupled map lattice using logistic map, which serves as spatiotemporal chaotic system. Patidar et al. [16] proposed a generator using logistic map. The generator uses two maps iterated with different parameters side by side and generates random number by comparing the output of the two chaotic maps. In [17] Guyeux et al. combined ISAAC and XOR shift generator by iterating chaotic map.

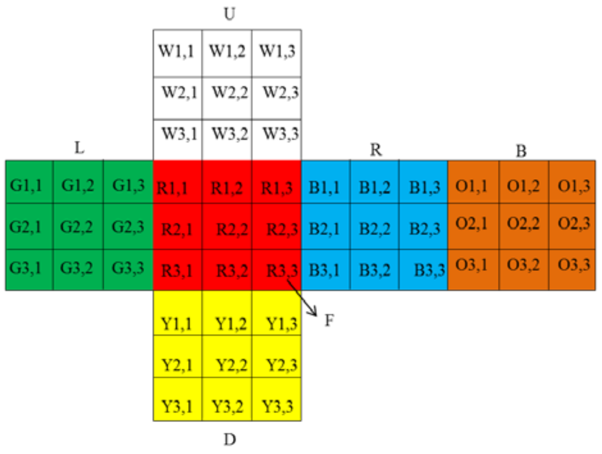
Cristina et al. [18] proposed a chaos based PRBG for developing a stream cipher using tent map. Wang et al. [19] uses two chaotic maps, parameter of one chaotic map is used to trigger trajectory of another chaotic map. This is advantageous in increasing the cycle length of another chaotic map. Most of the chaos based PRBG uses stationary maps. Sequence generated by using stationary chaotic map can be traced by tracing the system orbit by extracting this information from chaos theory methods for example phase space construction. Therefore Liu et al. in [5] proposed a non-stationary logistic map based PRBG. The algorithm dynamically changes the parameter sequence of chaotic map.

PRNG based on hash functions have also been worked upon. Hash algorithms are being used for various applications like in encryption for data encryption, authentication, multimedia applications in image encryption and hashing [28, 29], data hiding and retrieval. Bertoni et al. in [25] proposed a pseudorandom number generator using sponge function. They propose a reseedable PRNG by integration of various sources of seeding. In [24] Ruehle proposes a PRNG based on iterative hash, where the output of the numerical sequencer like linear feedback shift register or counter is hashed to increase the unpredictability of the output. In [26] NIST published a recommendation for generation of random number using deterministic generators. The recommendation includes specification for building PRNG using cryptographic primitives like hash or keyed hash function, block cipher or elliptical curve.

2.2. Rubik’s Cube (Magic Cube)

Rubik’s cube is a three dimensional puzzle, with six faces with each covered by six different color. Each face can be rotated in clockwise or anticlockwise direction by certain angle. The rotation of faces is termed as move. To perform different moves the cube face can be rotated by 0°, 90°, 180°or -90°. In Rubik’s cube terminology 90° and -90° moves is termed as quarter turn and 180°moves is termed as double turn. In all there are 18 different moves possible based on the rotation direction and the face being rotated. The possible number of permutations of the pieces of a Rubik’s cube is very large. There are 4.3252 x 1019 possible combinations, which approximates to more than 43 quintillion possible combinations.

This large possible number of combination of Rubik’s cube causes high level of complexity. **Fig. 1** shows an opened up view of Rubik’s Cube.


U
W1,1 W1,2 W1,3
W2,1 W2,2 W2,3
W3,1 W3,2 W3,3
L
R
B
G1,1 G1,2 G1,3 R1,1 R1,2 R1,3 B1,1 B1,2 B1,3 O1,1 O1,2 O1,3
G2,1 G2,2 G2,3 R2,1 R2,2 R2,3 B2,1 B2,2 B2,3 O2,1 O2,2 O2,3
G3,1 G3,2 G3,3 R3,1 R3,2 R3,3 B3,1 B3,2 B3,3 O3,1 O3,2 O3,3
F
Y1,1 Y1,2 Y1,3
Y2,1 Y2,2 Y2,3
Y3,1 Y3,2 Y3,3
D

Notations

F- Fronts face, U- Upper face, D- Down Face, R- Right face, L- Left face and B- Back face.

Fig. 1. Rubik's Cube

2.3. Chaos Theory

The study of chaos and cryptography has shown a strong relation between them. Chaotic systems have properties like ergodicity, sensitivity to initial condition, mixing property, and also have deterministic dynamic and complex structure. These properties are comparable to confusion and diffusion in ciphertext with changes in plaintext and key in traditional cryptosystems. Chaos based cryptosystems are classified as analog or digital. Analog chaotic cryptosystem is based on synchronization and control. Digital cryptosystem is based on discrete deterministic dynamic system. In the method proposed in this paper iterated deterministic logistic chaotic map is used to generate pseudorandom bits. The complexity of chaos based system is mainly based on the dynamics of the chaotic systems. Along with complexity chaotic system depicts random behavior without losing its deterministic property.

In **Fig. 2**, the sensitivity of chaotic map to initial condition is shown. The trajectories soon diverge as the initial condition change and have no correlation between them. The logistic map function behavior for different value of map parameter is shown in **Fig. 3**.

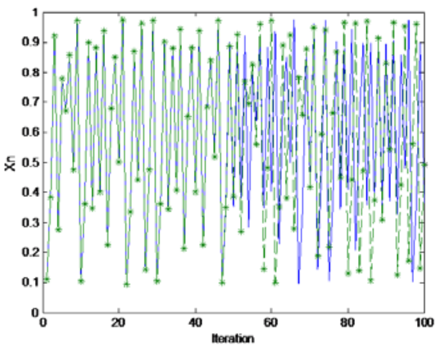


Fig. 2. Sensitivity to initial condition

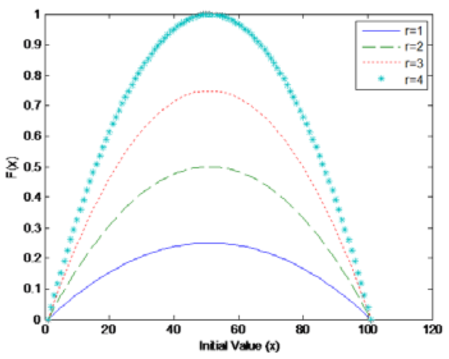


Fig. 3. Logistic Map function for different parameter value (r)

Chaotic system use Lyapunov constant to define the chaotic behavior of the system. A system is termed chaotic if the deviation of two trajectories (orbit) of the system that start very close to each other increases with time. Lyapunov exponent measures the rate of deviation of nearby trajectories from each other.

The plot of Lyapunov exponent for logistic map used in the proposed work is shown in **Fig. 4**. In the proposed method logistic map is used to create confusion by scrambling the LUT. It also used for the generation of bit pattern. The logistic map function is defined by

$$s_{n+1} = r * s_n(1 - s_n)$$

Where r ϵ (0, 4] is the chaotic map control parameter. For logistic map to be chaotic r should lie in the range [3.56695, 4] which is evident from **Fig. 4**, it shows the plot of Lyapunov exponent for logistic map. s_n is the map parameter and s_n ϵ [0, 1].

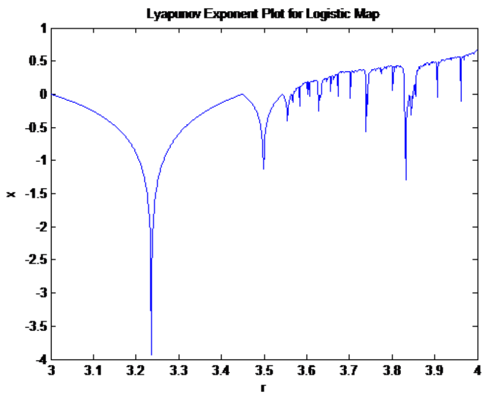


Fig. 4. Lyapunov Exponent plot for Logistic Map

2.4. Hash

Hash functions are used for various applications in cryptography. Some of these applications are Digital Signature Algorithm (DSA), Message Authentication Codes (MAC), Pseudorandom Number Generator (PRNG), etc. Hash algorithm is generally used for data integrity, authentication codes, etc. Dedicated hash algorithms are being developed for different application including image encryption and authentication [28, 29], data hiding, content retrieval, multimedia security. Hash functions take a message data as input and output a data referred to as hash codes, message digest or simply hash. A hash function takes a variable length message as input and maps it to a fixed length message digest. The output of the hash functions serves as a compact representation of message. The hash maps the entire message bits in the compressed output. This is useful for checking message integrity without revealing the message. The message cannot be predicted from the hash value thereby concealing the message.

In this paper, hash function is used to generate seed for the PRNG. For the PRNG to be secure the states of generator should have sufficient entropy so that an adversary cannot predict the state of the generator. For this reason, the seed to the generator should have sufficient entropy. Physical sources provide seed with low entropy. Thus, seed from variable sources are taken. In the work presented, the seed is generated using a hash function. Initial vector is input to the hash function the output hash is then taken as seed for the generator.

3. Proposed Work

As discussed earlier a PRNG is a deterministic algorithm. It generates a random like output bit stream by taking output from a RNG as seed as input. The proposed pseudo random bit generator uses Rubik's cube, chaotic map and cryptographic hash function for bit generation. **Fig. 5** shows the block diagram for the proposed pseudorandom number generator. **Algorithm 1** presents the proposed generator model. **Fig. 7** represents the flowchart for the PRNG in the proposed work.

The proposed algorithm is explained in the section below.

Step 1

Look up table (LUT) initialization: First step of the generator is LUT initialization. This is used to generate initial vector for generating the seed for use as input to the generator. The size of the LUT is 32x8, mapping all the letters, numbers for defining cubie position and special characters. The mapping is then scrambled using chaotic map, iterated using a specific initial condition and map parameter.

Step 2

Initial Vector generation: Initial vector is generated based on the standard colors present in the standard Rubik's Cube and the position of the cubie. The characters forming the color and position are mapped by the values in the LUT forming the initial vector.

Step 3

Seed Value generation: The initial vector generated in Step 2 is hashed using Standard Hash Algorithm (SHA-512). This hashed initial vector is input to the Rubik's Cube generator. The seed length is 3456 bits.

Step 4

Scramble Parameters: The seed value generated is arranged on Rubik's cube faces. The number of moves and the movements are decided from the LUT. The value mapped in the LUT is used to determine the number of moves and the movement from 18 different moves.

Step 5

Scrambling Rubik's Cube: After data arrangement the Rubik's cube is scrambled based on the number of moves and the rotation pattern generated form the LUT in the previous step.

Step 6

Chaotic bit pattern generation: In this step bits are generated using chaotic map. Logistic map is used for generation of bit pattern. Two logistic maps are used, the initial parameters are set. The outputs of the two maps are compared and the bits are generated accordingly. Let $S_{0,n}$ and $S_{1,n}$ be the output of the two maps and R_n are the bits generated, the chaotic map is iterated n times. The bits are compared as follows:

$$R_{x+1} = \begin{cases} 0, & If\ S_{0,x+1} \neq S_{1,x+1} \\ 1, & If\ S_{0,x+1} = S_{1,x+1} \end{cases} \tag{1}$$

The bits generated R_n are then XORed with the scrambled bits obtained after Rubik's cube scrambling. The bits obtained after scrambling is the final bit sequence of the generator. The length of the generated sequence 3456 bits. (Detailed explanation can be found in [16]). **Fig. 6** shows the block diagram for bit generator using chaotic map.

Step 7

LUT update: After each sequence generated the LUT is updated. For updating LUT 256 bits are selected in a sliding window way. The position of selecting 256 bits is taken from the iteration of the generator.

Algorithm 1: Proposed algorithm

```
1: Look-up table initialization: 32x8 sized LUT.
2: Initial vector generation $V_{init}$.
3: Seed value ($V_{seed}$) =Hash ($V_{init}$)          Hash=SHA-512.
4: LUT$\rightarrow$ (Moves, Movement).
5: $V_{scramb}$=$Scramble_{RC}$ ($V_{seed}$, Moves, Movement)
6: Chaotic Bit generator ($V_b$)
7: if $S_{0,j+1} = S_{1,j+1}$ then
8: $V_{b,j+1}$=1
9: else
10: $V_{b,j+1}$=0
11: end if
12: $V_{out}$=$V_{scramb}$ xor $V_b$
13: LUT update ($V_{out}$)
```

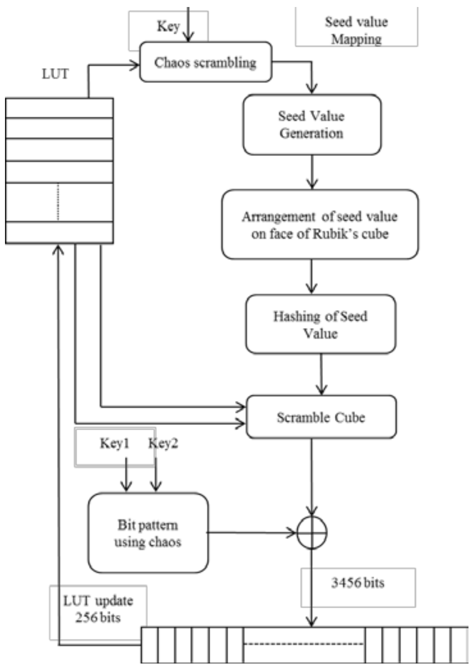


Fig. 5. Block diagram for proposed PRNG

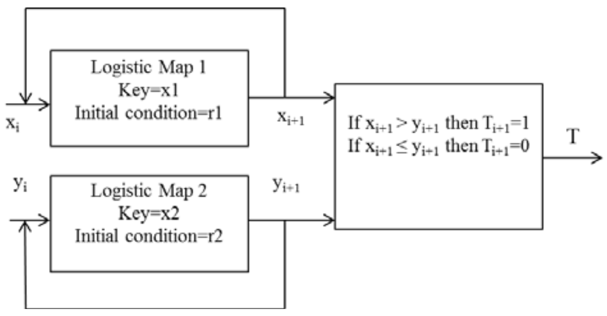


Fig. 6. Bit generator

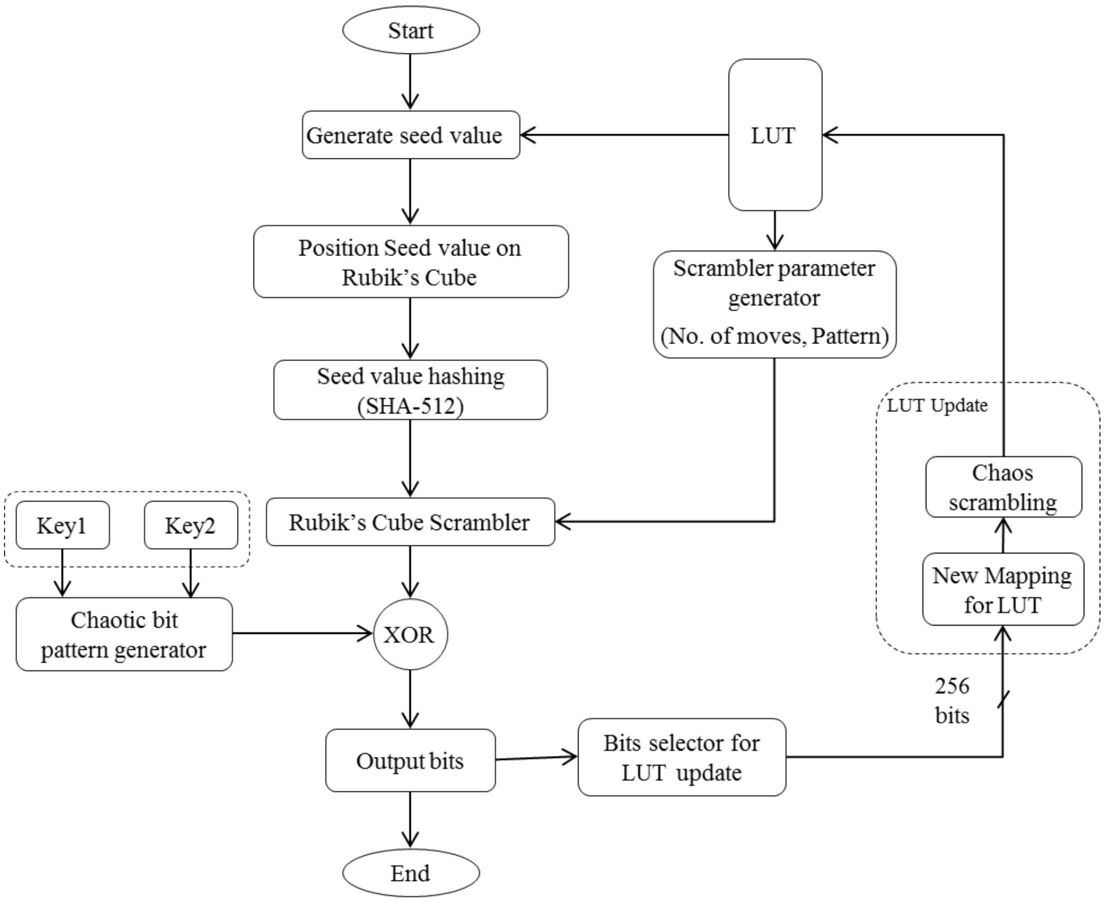

Start
Generate seed value
LUT
Position Seed value on Rubik's Cube
Scrambler parameter generator (No. of moves, Pattern)
Seed value hashing (SHA-512)
LUT Update
Chaos scrambling
Key1
Key2
Rubik's Cube Scrambler
New Mapping for LUT
Chaotic bit pattern generator
XOR
256 bits
Output bits
Bits selector for LUT update
End

Fig. 7. Flowchart for proposed PRNG

4. Statistical Analysis

Any random number generator should be tested for randomness. There are various test suites for testing of pseudorandom number. Though these tests cannot differentiate a true random number from a pseudorandom number, but can detect any statistical characteristics in the pseudorandom number that can be exploited to guess the generators output.

Some of the statistical tests suites frequently used for testing of pseudorandom number generators are DieHard Test, Crypt-XS statistical test suite, NIST statistical tests suite [26]. Out of these test suites NIST statistical test is mostly used for random number testing and is currently the industrial norm for testing of random numbers or randomness.

4.1. NIST Statistical Test Suite (NIST-STS)

The NIST statistical test suite [26] consists of 15 tests, to test the randomness of binary sequences, that are either generated by software or hardware based pseudorandom number generator. The test suite tests various types of non-randomness that could exist in a binary sequence. These 15 tests are further divided in sub groups of tests. The 15 tests are:

1. Frequency Test
2. Cumulative Sum Test
3. Frequency Test in a Block
4. Runs Test
5. Rank Test
6. DFT Test
7. Longest Run of Ones in a Block Test
8. Non-overlapping Template Matching Test
9. Overlapping Template Matching Test
10. Universal Statistical Test
11. Linear Complexity Test
12. Approximate Entropy Test
13. Random Excursion Test
14. Random Excursion Variant Test
15. Serial Test

These tests are further classified as parametric and non-parametric tests. The results of statistical tests for testing of randomness are given in the form of p-value. The p-value is the probability that given a perfect random number generator would produce less random sequence than the sequence being tested.

Standard normal and chi-square distribution are taken as reference for the tests. NIST considers the binary sequence under test to be random if all the tests are passed, but there is high probability that a true random sequence would also fail at least one of the tests. Statistical tests are formulated by assuming a null and an alternate hypothesis. For the given test considering the sequence under test to be random is taken as the null hypothesis and alternate hypothesis is that the sequence is not random. A significance level (α) is set for each test in the test suite that gives the percentage of sequence passed the test from the different test sequence provided. It is defined as the probability that the sequence is non-random as indicated by the test when it is random. An α value of 0.001 indicates that the sequence is taken to be random with the confidence of 0.999. For cryptographic purpose, the value of α commonly used is 0.01. Null hypothesis is accepted if the p-value is greater than α i.e. is the significance level. However, if the p-value is less than α then the alternate hypothesis is selected thereby proving the sequence to be non-random.

Selection of significance level is important in determining proper random number generator. A very high significance level can reject sequences that are truly random this is referred to as type I error. Whereas selecting a very low significance level will accept sequences as random that are not even random, this is referred to as Type II error.

For each statistical test in the test suite, a set of p-value is produced. As mentioned previously if the sequence has p-value produced from a test greater than α then the sequence pass that particular test.

4.2. Result Interpretation

Distribution of p-value (Uniformity Analysis): For each test mentioned earlier p-value of each test sequence is calculated. The interval [0 1] of p-value is divided into 10 bins. After completion of test for each sequence, all the p-value lying in one bin are counted and placed in that bin. For example first bin contains all p-value ranging between [0 0.1), second bin contains p-value between [0.1 0.2) and so on. After calculation of p-value, the uniformity analysis is conducted. This analysis checks the distribution of the p-value. For a sample of sequence to be random the distribution of p-value should be uniform. Uniformity analysis is done using Chi-Square test and p-value is determined corresponding to the Goodness of fit of distribution test.

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - s/10)^2}{s/10}$$

Here, P_j is the number of p-value in the interval *j*, and *t* is size of the sample. For the p-values to be uniform, a threshold p-value is calculated using χ^2. If the p-value generated for each test is greater than the threshold p-value then the distribution of p-value is considered uniform and the sequence to be random. The threshold p-value is calculated using p-value$_T$ = igamc (9/2, χ^2). P-value $\geq$ 0.0001 indicates uniform distribution. For statistically meaningful result, at least 55 sequences should be tested.

Proportions (Proportion Analysis): Testing of certain number of sequences is done for randomness testing. A certain proportion of this test should pass for the samples to be termed random. For example if 1000 (m=1000) sequences are tested and the significance level(α)=0.01 and test result are such that 995 sequence have p-values $\geq$ 0.01 then the proportion is 995/1000=0.995. The range of the acceptable proportion is determined by the formula,

$$t \pm 3\,\frac{\overline{t(1-t)}}{m} \qquad (t=1-\alpha)$$

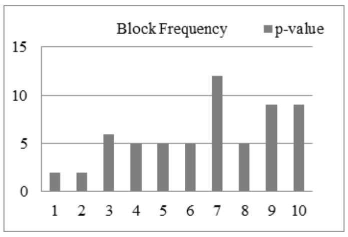
5. Result and Observation

In this section, results of the NIST statistical test suite for PRBGs are shown and discussed. The generated sequences are tested for each of the 15 test in the test suite. For testing, we have tested 100 samples of length 10^4 and 10^6 bit sequences. Input parameters used for test are given in **Table 1**.

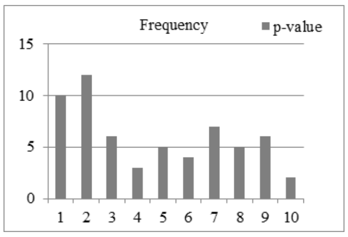
Table 1. Input Parameters

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

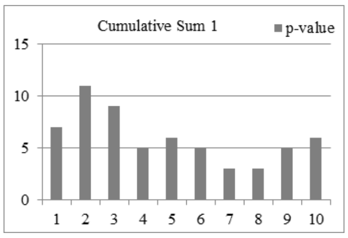
For proportion analysis the histogram for distribution of p-value is plotted for few tests. The intervals of the histogram are divided into 10 bins as for the p-value intervals. Histogram for distribution of p-value is shown in **Fig. 8**.



(a)



(b)



(c)

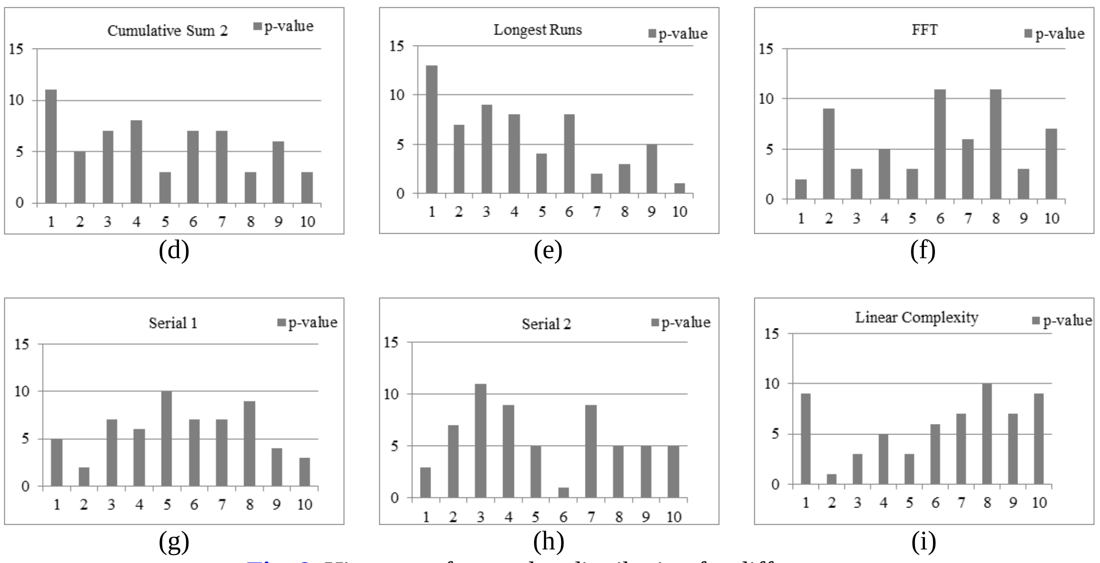


Fig. 8. Histogram for p-value distribution for different tests

As can be seen from **Table 2** the tests are successful. The histogram plot also indicates uniform p-value distribution. The results thus prove that the test sequences generated are random.

Along with NIST test the sequences is also checked for sensitivity for chaotic parameters i.e. the key to the logistic map. The key is changed and the difference in the bits between the two sequences is calculated. The difference is calculated in percentage. **Table 3** shows the percent change in the subsequent sample with the change in initial condition of the logistic map. The percent change in the sequence is found to be around 50%. These results prove that the generator is sensitive to initial condition.

Table 2. NIST Statistical Test Results

Test	**p-value**	**Result**
Frequency Test	0.181557	Success
Frequency Test within a Block	0.002559	Success
Longest-Run-of-Ones in a Block Test	0.023545	Success
Binary Matrix Rank Test	0.048716	Success
FFT Test	0.035174	Success
Non-overlapping Template Matching Test	0.816537	Success
Overlapping Template Matching Test	0.007694	Success
Linear Complexity Test	0.090936	Success
Serial Test	0.514124	Success
Approximate Entropy Test	0.078277	Success
Cumulative Sums Test	0.262249	Success
Random Excursions Test	0.637119	Success
Random Excursions Variant Test	0.911413	Success

The Approximate Entropy of the generated sequence is also calculated. For testing the randomness of an observed sequence [27] suggested use of approximate entropy. Approximate entropy sees for the repetition of patterns in a sequence and the likelihood of them to be present in the next incremental comparison. It measures the logarithmic frequency with a certain defined block size that is close together for blocks differing by one position. Small value of approximate entropy implies regular pattern in the sequence. A large value of entropy implies substantial fluctuation and irregularity in the sequence. Entropy has been plotted in **Fig. 9.** From the figure it is evident that the entropy of the sequence increases with increase in the length of the sequence, thereby implying the sequence under consideration to be random.

The execution time and speed for the generator have been calculated. The algorithm has been implemented in MATLAB 8.3. The algorithm is iterated on an Intel Core i7-4770 3.40 GHz CPU with 32 GB RAM running on Windows 8 OS for over 3000 iterations for calculation of average execution time and speed. The bit generation speed is calculated to be 1.85 Mbps with an average execution time of around 1.78 ms.

Table 3. Key Sensitivity Analysis

Input Sequence Pair	Percent Change (%)
(Sample1, Sample2)	50.49
(Sample2, Sample3)	48.32
(Sample3, Sample4)	48.90
(Sample4, Sample5)	50.84
(Sample5, Sample6)	51.01
(Sample6, Sample7)	49.31
(Sample7, Sample8)	49.83
(Sample8, Sample9)	49.36
(Sample9, Sample10)	51.04
(Sample10, Sample11)	49.22

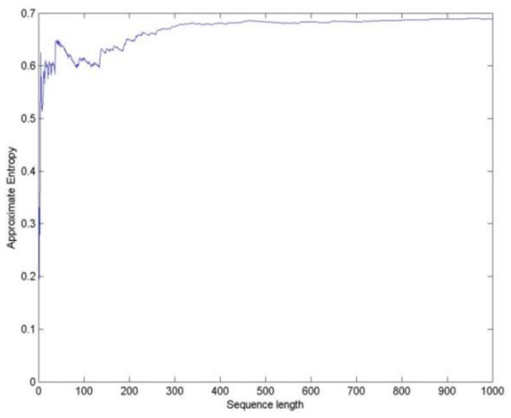


Fig. 9. Approximate entropy of generated sequence

6. Conclusion

In this paper we have proposed a PRNG using Rubik's cube scrambling. The generator uses hash algorithm to generate the seed. Chaotic map is used due to its chaotic property and sensitivity to variation in initial condition which helps in achieving randomness. The generated sequences are rigorously tested for randomness using NIST-STS. It consists of 15 different tests for checking properties proving an input or a generator to be random. The sample sequences tested qualify all the tests along with uniformity and proportion analysis.

References

[1] Petrie, Craig S., and J. Alvin Connelly. "A noise-based random bit generator IC for applications in cryptography," in *Proc. of Circuits and Systems, 1998. ISCAS'98. Proceedings of the 1998 IEEE International Symposium on.* Vol. 2. IEEE, 1998.Article (CrossRef Link)

[2] Brederlow, Ralf, Ramesh Prakash, Christian Paulus, and Roland Thewes. "A low-power true random number generator using random telegraph noise of single oxide-traps," in *Proc. of Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, pp. 1666-1675. IEEE, 2006.Article (CrossRef Link)

[3] Bucci, Marco, Lucia Germani, Raimondo Luzzi, Alessandro Trifiletti, and Mario Varanonuovo. "A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC." *IEEE transactions on computers* 52, no. 4, p. 403-409, 2003. Article (CrossRef Link)

[4] Zeng, Kencheng, C-H. Yang, D-Y. Wei, and T. R. N. Rao. "Pseudorandom bit generators in stream-cipher cryptography," *Computer* 24, no. 2, p. 8-17, 1991. Article (CrossRef Link)

[5] Liu, Lingfeng, Suoxia Miao, Hanping Hu, and Yashuang Deng. "Pseudorandom bit generator based on non-stationary logistic maps," *IET Information Security* 10, no. 2, p. 87-94, 2016. Article (CrossRef Link)

[6] Kocarev, Ljupco, and Goce Jakimoski. "Pseudorandom bits generated by chaotic maps," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 50, no. 1, p. 123-126, 2003. Article (CrossRef Link)

[7] Stojanovski, Toni, and Ljupco Kocarev. "Chaos-based random number generators-part I: analysis [cryptography]," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 48, no. 3, p. 281-288, 2001. Article (CrossRef Link)

[8] Pellicer-Lostao, Carmen, and Ricardo López-Ruiz. "Pseudo-random bit generation based on 2D chaotic maps of logistic type and its applications in chaotic cryptography," in *Proc. of International Conference on Computational Science and Its Applications*, pp. 784-796. Springer, Berlin, Heidelbe rg, 2008. Article (CrossRef Link)

[9] Hu, Yue, Xiaofeng Liao, Kwok-wo Wong, and Qing Zhou. "A true random number generator based on mouse movement and chaotic cryptography," *Chaos, Solitons & Fractals* 40, no. 5,p. 2286-2293, 2009.Article (CrossRef Link)

[10] Yalcin, Mustak E., Johan AK Suykens, and Joos Vandewalle. "True random bit generation from a double-scroll attractor," *IEEE Transactions on Circuits and Systems I: Regular Papers*51, no. 7, p. 1395-1404, 2004.Article (CrossRef Link)

[11] Andrecut, M. "Logistic map as a random number generator," *International Journal of Modern Physics B* 12, no. 09, p. 921-930, 1998. Article (CrossRef Link)

[12] Szczepański, Janusz, Zbigniew Kotulski, Karol Górski, Andrzej Paszkiewicz, and Anna Zugaj. "On some models of pseudorandom number generators based on chaotic dynamical systems," in *Proc. of RCMCIS* 99, p. 213-220, 1999. Article (CrossRef Link)

[13] Li, S. J., X. Q. Mou, and Yuan-Long Cai, "Pseudo-random bit generator based on couple chaotic systems and its application in stream-ciphers cryptography," in *Proc. of Progress in Cryptology–INDOCRYPT 2001: Second International Conference on Cryptology in India Chennai, India, December 16 C20, 2001 Proceedings*, pp. 316-329. 2001. Article (CrossRef Link)

[14] Lü, Huaping, Shihong Wang, and Gang Hu, "Pseudo-random number generator based on coupled map lattices," *International Journal of Modern Physics B* 18, no. 17n19, p. 2409-2414, 2004. Article (CrossRef Link)

[15] Li, Ping, Zhong Li, Wolfgang A. Halang, and G. R. Chen. "A novel multiple pseudorandom bits generator based on spatiotemporal chaos," in *Proc. of World Congress*, vol. 16, no. 2, p. 836. 2005. Article (CrossRef Link)

[16] Patidar, Vinod, Krishan K. Sud, and Narendra K. Pareek. "A pseudo random bit generator based on chaotic logistic map and its statistical testing," *Informatica* 33, no. 4, 2009. Article (CrossRef Link)

[17] Guyeux, Christophe, Qianxue Wang, and Jacques M. Bahi. "A pseudo random numbers generator based on chaotic iterations: application to watermarking," in *Proc. of International Conference on Web Information Systems and Mining*, pp. 202-211. Springer, Berlin, Heidelberg, 2010. Article (CrossRef Link)

[18] Cristina, Dăscălescu Ana, Boriga Radu, and Răcuciu Ciprian. "A new pseudorandom bit generator using compounded chaotic tent maps," in *Proc. of Communications (COMM), 2012 9th International Conference on*, pp. 339-342. IEEE, 2012. Article (CrossRef Link)

[19] Wang, Xing-Yuan, and Lei Yang. "Design of pseudo-random bit generator based on chaotic maps," *International Journal of Modern Physics B* 26, no. 32, 1250208, 2012. Article (CrossRef Link)

[20] Walker, John. "HotBits: Genuine random numbers, generated by radioactive decay," *Online: http://www. fourmilab. ch/hotbits*, 2001. Article (CrossRef Link)

[21] Jakobsson, Markus, Elizabeth Shriver, Bruce K. Hillyer, and Ari Juels. "A practical secure physical random bit generator," in *Proc. of Proceedings of the 5th ACM Conference on Computer and Communications Security*, pp. 103-111. ACM, 1998. Article (CrossRef Link)

[22] Jun, Benjamin, and Paul Kocher. "The Intel random number generator," *Cryptography Research Inc. white paper*, 1999. Article (CrossRef Link)

[23] Gutterman, Zvi, Benny Pinkas, and Tzachy Reinman. "Analysis of the linux random number generator," in *Proc. of Security and Privacy, 2006 IEEE Symposium on*, pp. 15-pp. IEEE, 2006. Article (CrossRef Link)

[24] Ruehle, Michael. "Hash-based pseudo-random number generator," *U.S. Patent Application 09/963,857*, filed March 27, 2003. Article (CrossRef Link)

[25] Bertoni, Guido, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Sponge-based pseudo-random number generators," in *Proc. of International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 33-47. Springer, Berlin, Heidelberg, 2010. Article (CrossRef Link)

[26] Barker, Elaine, and John Kelsey. "NIST special publication 800-90A: Recommendation for random number generation using deterministic random bit generators." 2012. Article (CrossRef Link)

[27] Pincus, Steve, and Burton H. Singer. "Randomness and degrees of irregularity," *Proceedings of the National Academy of Sciences* 93, no. 5, p. 2083-2088, 1996. Article (CrossRef Link)

Saiyma Fatima Raza is a Research Scholar at Department of Electronics and Communication Engineering at Visvesvaraya National Institute of Technology, Nagpur. She has completed her Master's degree in VLSI Design from Department of Electronics Engineering at Ramdeobaba College of Engineering and Management, Nagpur in the year 2015 and Bachelor's degree in Electronics Engineering in the year 2013. Her research work is in the field of Cryptography and Image Security.

V. R. Satpute is working as Assistant Professor in Department of Electronics and Communication Engineering at Visvesvaraya National Institute of Technology, Nagpur. He has completed his Bachelor's degree in Engineering from Nagpur University in 2001 in the field of Electronics Engineering and Master's degree M. Tech. (Communication Systems) from IIT Madras, Chennai in 2003. He completed his Ph. D. from Department of Electronics and Communication Engineering at Visvesvaraya National Institute of Technology, Nagpur in the year 2015. His Total teaching experience is 15 years. His research work is in the field of Image Processing, Video Compression, IoT, Computer vision and Cryptography.