A Dissection of Pseudorandom number Generators

Ankur
Research Associate
AIM&ACT, Banasthali Vidyapith
Rajasthan, India
ankur.rathi9@gmail.com

Divyanjali
Assistant Professor
CEA Department, GLA University
Mathura, India
dvynjli@gmail.com

Trishansh Bhardwaj
M.Tech Researcher
CSE Department, NIT Warangal,
Telangana, India
itrishansh@gmail.com

***Abstract*—Security over the network has become the most challenging issue with the day by day increase in use of internet and other such services provided by network. The secure transmission, storage and access of data or information are the key issues for security. To understand and implement security one needs to first know in deep the concept of pseudo-random numbers because they play a major role in internet security. Pseudo-random number generators are used to generate these numbers which can be then used as keys or in any other form. In this paper we have presented various pseudorandom number generators that are used for security purpose to generate the encryption keys, SSL connection, database security etc. There is a need to understand how and for what purpose these pseudorandom number generators can be used. We also elaborate some of the limitations of the same.**

Keywords—Random numbers, Pseudorandom generators, Linear Congruential Generator, Lagged Fibonacci Generator, Inversive Congruential Generator.

I. INTRODUCTION

Today most of random numbers are generated by computers using algorithms which are always deterministic in nature. This leads to the concept of pseudo-random numbers, in which numbers are generated from some random seed values. It is very difficult for an observer to make a distinction between true random numbers and pseudorandom numbers. These generators are called pseudo random number generator (PRNG) due to lack of pattern and possibility to be used instead of true random number generators, which are expensive to use [1].

The security of many cryptographic systems depends on the generation of unpredictable entities. These entities must be large enough in size and random. Randomness is high means the probability of any particular value being selected must be sufficiently small to predict the coming sequence of a random number generator.

One of the most important tasks of random number generators is key generation but its uses are not limited to cryptography. They can be classified, depending upon nature of generator; if the random number generator is based on any one-way function it is said to be cryptographically secure pseudo random number generator (CSPRNG). A property of these RNGs is that no algorithm exist that can predict the next bit that will be generated in polynomial time. It is not possible to find out next bit in the sequence even when previous bits are known without the knowledge of seed. If the RNG is to be used in simulation, it need not be cryptographically secure but should have long cycle length and uniform distribution over the range of number domain.

Recently, it is found that National Security Agency (NSA) was researching on the methods for weakening the random number generation algorithms. The reason behind carrying out such research is the root involvement of random number generations in all the routine tasks like SSL, communication through internet browsers and military services, key generation, session key generation, and unique ID generation. So weakening RNG would enable one to spy and wiretap the data and communication secretly, because no matter how secure the algorithm is, according to **Kerckhoff's Principle**, guessing the key should be so difficult that there is no need to hide to encryption/decryption algorithm. This is the reason that algorithms are always public but keys are always secret. Secrecy can be achieved if the key of the encipherment algorithm is truly a random number generated by a good random number generator [2].

II. RANDOM NUMBER GENERATORS

Definition 1: Let k, l be positive integers such that $l \geq k+1$. A (k, l) - pseudorandom bit generator is a function $f: (Z_2)^k \rightarrow (Z_2)^l$ that can be computed in polynomial time. The input s_0 (Z_2) k is called the seed and the output $f(s_0) \in (Z_2)^l$ is generated bit stream.

The function f is deterministic, so the bit string $f(s_0)$ is dependent only on the seed. Our goal is that the generated bit string $f(s_0)$ should look like truly random bits, given that the seed is chosen at random [3].

The random number generators can be classified further as

- **True Random Number Generators**

A true random number generator (TRNG) uses a natural or non-deterministic source to produce randomness. The majority of TRNG's operate by capturing unpredictable natural processes, such as pulse detectors of ionizing radiation events, radio frequencies, radioactive decay etc to obtain the entropy having sufficient unpredictability. The quality of random numbers depends upon the phenomenon used and design of that TRNG.

- **Pseudorandom Number Generator**

Pseudorandom numbers are produced by an algorithm using computers; generated random numbers are not truly random they appear to be random. Various types are discussed further.

Random number generators are very famous and are extensively used for simulation and security purposes. Some

of these are Linear congruential [4], Lagged Fibonacci [5], Quadratic congruential [6], Linear feedback shift register [7], Inversive congruential [8].

III. PROPERTIES OF GOOD PSEUDORANDOM NUMBER GENERATORS

An ideal pseudorandom number generator should posses certain properties that defines the quality of the outputs produced some of these features are listed below.

A. Large period length and Good uniform distribution properties

The applications of simulations require more and more unique random numbers and computers now allow more iteration steps than they used to provide some time back. Hence the random number generator must have a very large period so that cycle should not be formed and should be uniformly distributed over [0, 1].

B. Little intrinsic structure and Efficiency, fast generation algorithm

Successive values produced by some generators have a lattice structure in any dimension. The smaller the distance between hyper planes is, the better is the generator. In addition to this the algorithm should allow a fast generation of random numbers and it should not require too much memory space.

C. Portability, Repeatability and Unpredictability

Portability means that the generator should produce exactly the same sequence on different computers or with different compilers [9]. The generator should be repeatable in the sense that it should be able to reproduce exactly the same sequence can be necessary and very useful for practical applications. Unpredictability ensures forward and backward unpredictability.

IV. STATISTICAL TESTING

There are some predefined criteria and statistical tests mentioned to decide the quality and randomness of any given random number generator (RNG).

Each pseudorandom number is generated from the previous value of sequence, after performing many transformations, which enhances and add additional randomness to the next generated sequence. Such series of transformation leads to eliminate statistical auto-correlation between input and output of previous and successive generated numbers. Thus the generated pseudorandom numbers have better statistical properties, and to guarantee these statistical properties various statistical tests can be applied to the sequences to analyze and weigh the true randomness of the sequences. There are many possible statistical tests, each of which tries to find out presence and absence of a "pattern" which, if identified, the sequence is found to be non-random. As there are so many tests present to quantify the randomness of generated sequences, no specific test is deemed fully complete; they only try to find out the very obvious patterns, correlation and non-randomness in the sequences. Some of these tests are NIST statistical test suite 800-22 [10], Diehard, Crypt-XS, and TestU01. These entire tests suites contain a series of tests like poker test, serial test, universal test, frequency test, runs test, overlapping template test to find out the defects if any exist and non-randomness in generated sequences.

To Test the statistical properties of mentioned generators, we apply NIST 800-22 (National Institute of Standards and Technology). A Statistical Test Suite for Random and Pseudorandom Number Generators is easily available on NIST website, mostly recommended and trustworthy. It consists of 15 tests, each test is performed to analyse different patterns of 0 and 1 to verify that no correlation, cycle, pattern exist in the generated random numbers, if it happens then the generator is considered to be biased and the output is no more random.

V. SOME PSEUDORANDOM NUMBER GENERATORS

Some popular pseudorandom number generators are explained in detail in this section.

A. Linear Congruential Generator

Linear Congruential Generators (LCG) are the most widely used generators to produce random numbers for non-cryptographic purpose i.e. useful for Monte Carlo Simulation; they have a linear structure. This method was introduced by D. H. Lehmer in 1949 [4]. In this approach we select four variables m, a, c, X_0 where

m = "modulus", $m>0$
a = "multiplier", $0 \le a < m$
c = "increment", $0 \le c < m$
X_0 = "starting value or seed", $0 \le X_0 < m$

The required sequence of random number can be obtained by the following equation:

$$X_{n+1} = (aX_n + c) \bmod m, \quad 0 \le X_n+1 < m,\ n \ge 0, \tag{1}$$

The output is the linear congruential sequences.

While choosing the parameters the following conditions should be satisfied:

- c and m should be relative prime, no common factors i.e. *gcd(c,m)*=1.
- a-1 is divisible by all prime factors of m.
- a-1 is a multiple of 4 if m is a multiple of prime and Lehmer also recommended to use the value of modulus as $m= 2^{31}$-1 a meresenne prime [11].

The sequence generated with the help of m, a, c, X_0 are not always random and may have small period if they are not chosen appropriately. One important thing is that we have to choose the value of m, which defines the period length of the generator. And because the period length is equal to m-1, it also affects the speed since the period does not contain the number of elements more than m. But the choice of m also depends upon the word length of machine on which it is to be used in the form of 2^k where k is the size of word. The increment variable i.e. c had the value $c = 0$ in Lehmers original proposed idea but he also left the possibility that the value of $c \neq 0$ can also be considered. When the value of c=0 is used, the period length get shortened although $c \neq 0$ can lead to better period length [6]. If we use the value of $c = 0$, the linear congruential generator will reduce to multiplicative

congruential generator and the mixed congruential generator when $c \neq 0$.

All congruential generators have a defect i.e. correlation among generated values that cannot be diminished with a correct choice of multiplier, modulus or starting value such as seed that makes them inappropriate for many Monte Carlo problems.

Greorge Marsaglia, the famous mathematician published a paper in June 1968 "Random Numbers Fall Mainly in The Planes" [12]. In that paper he elucidate the issues of multiplicative congruential generators that if we view the generated sequences as the points in a unit cube of k dimensions, all points lie in small numbers of parallel hyperplanes i.e. the sequences exhibit the correlation property as it is clearly available in figure 1.

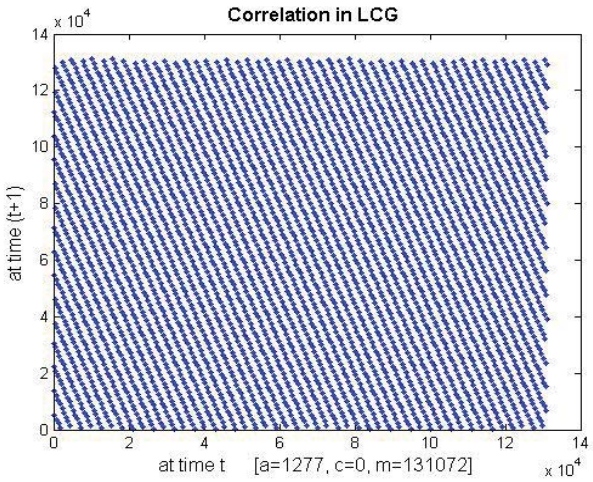

Fig. 1: Correlation Visible in LCG

B. *Lagged Fibonacci Generator*

Fibonacci sequence is the sequence in which X_{n+1} depends on two preceding values. Lagged Fibonacci Generator is a pseudorandom number generator and it is named Fibonacci because it is primarily based on Fibonacci series.

$$X_{n+1} = (X_n * X_{n-k}) \bmod M \qquad (2)$$

This recurrence relation was introduced by Green, Smith and Klen in their paper published in 1959 [5]. This g enerator was viewed as an improvement over Linear congruential generators in early 1950s, and it usually provides a period length greater than m. But test conducted have shown that the numbers produced by Fibonacci recurrence are definitely not satisfactorily random.

The operator * here do not represents the multiplication, it may be any binary operation likewise addition or subtraction, but not exclusive. Marsaglia explained that if we use exclusive-or operation the resultant bits depend only on the same bits of two operands on which the operation is performed but in the case of '+' or '-' the resultant bits depend on other neighbouring bits also because of carry or borrow bits, which in turn provides a mixing of bits.

In a very general form the equation can be represented as

$$X_n = (X_{n-p} + X_{n-q}) \bmod M \quad , 0 < p < q \qquad (3)$$

Then p and q will be considered as lags and the period of generator can be defined as $(2^q\text{-}1)$ $(2^\alpha\text{-}1)$, where 2^α (α is word size) is equal to M and $p < q$. The choice of p, q and initial q numbers is critical to have a good period length and the value of M is generally chosen as the 2^α to get more efficiency in calculation but choice of a prime number as modulus is the best one. The value of lags i.e. p and q is not random. Many special cases are mentioned by Knuth [6] like (273,607), (576, 3217), (1029, 2281), (4187, 9689) and for further details of lag pairs go through the following paper [13]. Such a generator if is using the addition operator then the initial q numbers must be a good mix of even and odd numbers since if all of them are even then adding two even numbers eventually produce an even number and hence all the generated numbers would be even.

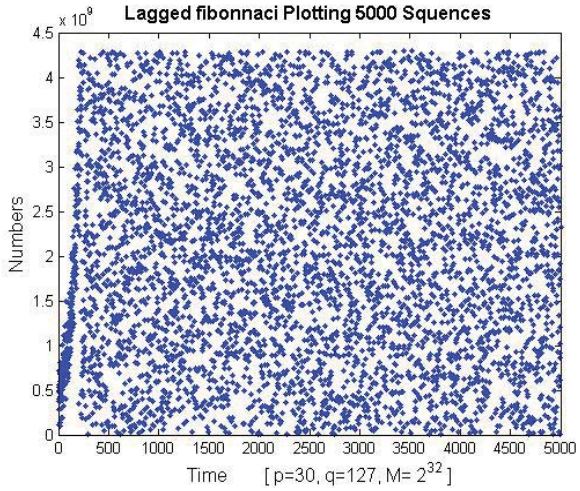

Fig. 2: Lagged Fibonacci Initial complexity from 0 to 500

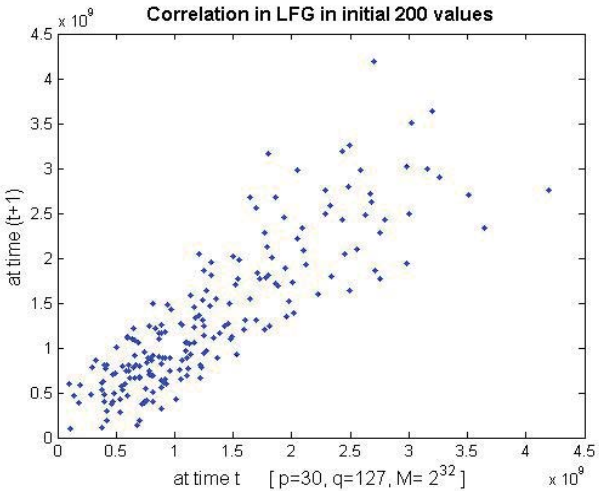

Fig. 3: Correalatiom in LFG in initial values at time t and t+1

The initialization of LFGs is a very complex problem. The output of LFGs is highly susceptible to initial conditions, because of using the previously generated numbers; correlation is visible in graph that is plotted using initial 200 values, we can easily see that sequences are not uniformly distributed.

The scatter graph below easily explains the initial complexity that arises in LFG but rest is highly distributed.

When we apply NIST test suit over the Lagged Fibonacci generator output, with the parameters i.e. lag values p=30, q=127 and modulo M is equal to 4294967296. It passes all the tests except for linear complexity.

C. Inversive congruential generator

Inversive congruential generator was proposed by Jurgen Eichenauer and Jurgen Lehn in 1986 [14] using the modular multiplicative inverse to generate the next number in the sequence. Using the relation

$$x_{n+1} = \begin{cases} a \cdot x_n^{-1} + b \ (mod\ p) & x_n \geq 1, 0 \leq x_{n+1} < p, n \geq 0 \\ b & x_n = 0 \end{cases} \quad (4)$$

The parameters here represents p is a prime number, a and b are positive integers $\in Zp$, $x_0 \in Zp\{0,1,.....,p\text{-}1\}$, $x^{-1}{}_n$ denotes the inverse of x_n modulo p for all $x \neq 0$ by

$$1 = x^{-1} \bullet x \mod p \quad (5)$$

The inverse of x can be easily calculated using the extended Euclidean algorithm which is a very fast method. Inversive congruential generators are purely periodic and highly distributed. They pass uniformity test if prime modulo p used, is of sufficient size. The value of parameters that are used is obtained by the equation

$$x^2 - bx - a \quad (6)$$

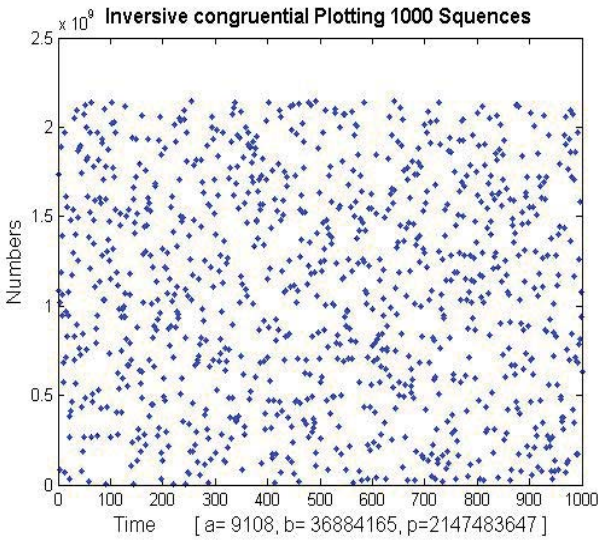

Fig. 4: Scatter plot of ICG

This is primitive polynomial over the finite field Zp , then inverse congruential have sequences $(x_n)n > 0$ has maximal period length p. The corresponding polynomials need not to be primitive polynomials. The inversive congruential generator passes the Marsaglia's lattice test "if we assume generated sequences as the points in a unit cube of k dimensions then all points lie in small numbers of parallel hyperplanes" with maximal period length p for all dimensions $k \geq (p+1)/2$ as it is proved by Niederreite [15] but linear congruential generators fails this Marsaglia,s lattice test for all dimensions $k \geq 2$. We used NIST statistical test suite to test the statistical properties of ICG.

D. Mersenne Twister

Lewis and Payne introduced a variation of Tausworthe's linear feedback shift register and named it generalized feedback shift register (GFSR) in 1973 [16]. Generalized feedback shift register generates l-bits using the recurrence relation

$$x_i = x_{i-j} \oplus x_{i-k} \quad (7)$$

j and k here are the mersenne primes which should satisfy the following property

$$i^2+j^2+1 = \text{prime}$$

Mersenne primes are calculated as 2^p-1, where p is prime number.

For example like i=98 and and j=27 can be used. But in generalized feedback shift register an initialization algorithm is also used for high equidistribution of sequences that makes the process time consuming and period length is not very good.

Matsumoto and kurita in 1992, 1994 introduced the variation of generalised feedback shift register as *Twisted GFSR generator* [17] and *Twisted GFSR generator II* [18]

$$x_i = x_{i-j} \oplus x_{i-k} \bullet A \quad (8)$$

where A is a $w*w$ matrix with 0,1 components, with suitable choice of j, k, and A the maximal period $2^{j.k\text{-}1}$ can be attained. In twisted GFSR II they described how to choose the value of A and change the output so as to achieve good uniform distribution over (0, 1).

Then after all these generators **Mersenne Twister** was introduced by Matsumoto and Nishimura [19] as advanced version of above mentioned GFSR generators, using the recurrence relation

$$x_{k+n} = x_{k+m} \oplus \left(x^u_k \mid x^l_{k+1} \right) \bullet A \quad \text{k= (0, 1,….)} \quad (9)$$

x^u represents higher order w-r bits of x where x is the integer with word length w and x^l represents the lower order r bits. The symbol | signifies concatenation. So, ($x_k{}^u \mid x_{k+1}{}^l$) means most significant w-r bits of x^k are concatenated with least significant r bits of x^{k+1} to form a new w bit word. Integer n is degree of recurrence, r is chosen from range $0 \leq r \leq w\text{-}1$, m is an integer between $1 \leq m \leq n$ and A is a $w \times w$ matrix having only 0's and 1's.

The seeds for this recurrence relation are integers from x_0 to x_{n-1} of word length w. Further values are determined by putting the value of k from 0.

If the value of r is 0 then ($x_k{}^u \mid x_{k+1}{}^l$) is equal to x_k and hence the generator would reduce to *Twisted GFSR* and if $A=I$ it would reduce to *GFSR*.

Merits of Merssene Twister

- Mersenne twister is a very popular generator till now because of its fast functioning since it only uses bit wise operator. It is faster than modulus generators.

- It has a very long period length equals to 2^{nw-r} -1, which is much greater then various present generators and satisfies the theoretical upper bounds.
- Value of *n* depends upon memory restrictions of system; *n* must be chosen as much as the system allows so that we have greater period length.
- Merssene twister is very popular as a random number generator that is used by most of the acknowledged programming languages like Java, Python, C++, Ada, PHP, Scala, Perl, Lisp, Fortran etc.

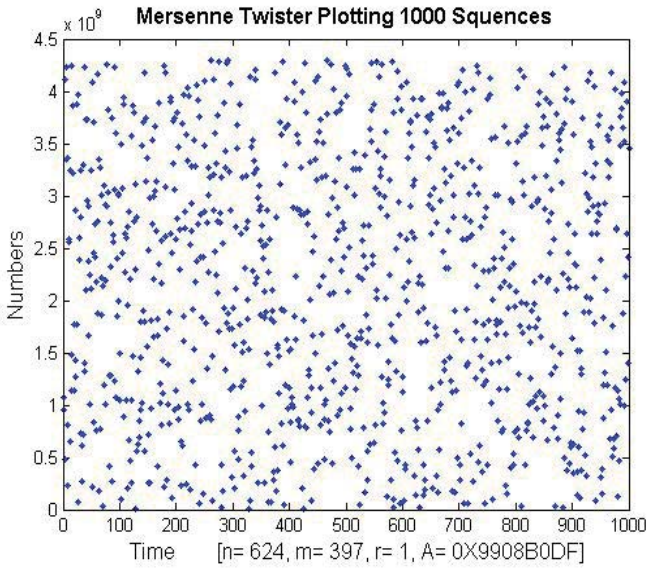

Fig. 4: Scatter plot of Mersenne Twister

VI. CONCLUSION

There are several pseudorandom number generators, but none of them is very good. Some are slow, some are not sufficiently random and others are not cryptographically secure. As we have mentioned many generators above, some of them can be used for Simulation and widely used for different applications. But it is found in our study that LCG are not good as they do not passes the lattice test and also posses patterns and generates 10^6 sequences in 16,423,979 ns quite good generation time comparison to other generators but when c=0, whereas the LFG generators are considered very high-quality generators and also used by Oracle database. But deciding the value of lags is very tedious and the initial values are quite not random that is easily visible in the figure 3, it takes 17,530,706 ns to generate 10^6 sequences. ICG, this is good enough but calculating inverse is a costly operation and fast generation is a need of random number generation. It generates 10^6 numbers in 1,242,363,486 ns that is very slow in comparison to others. But the last mentioned generator Mersenne twister is the fastest among all of them as it generates 10^6 sequences in 16,636,780, as well as gives a good period length and uniform distribution. The time variation that occurs during generation is visible in the time graph which shows the time consumes by a generator to generate 10^6 sequences.

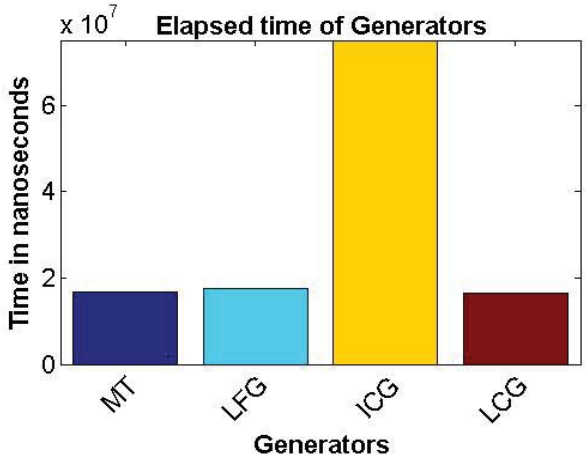

Fig. 5: Time evaluation graph of MT, LFG, LCG, & ICG

ACKNOWLEDGMENT

We acknowledge our regards and thanks to Banasthali Vidyapith for providing us an opportunity to perform such a research work in their premises and letting us use their resources to work on this unique area random number generation.

REFERENCES

[1] A. Menezes, P. Oorschot and S. Vanstone., " *Handbook of Applied Cryptography ",* CRC Press, Boca Raton, 1997.
[2] Behrouz A. Forouzan , Debdeep Mukopadhyay, *" Cryptography and Network Security"*, 2nd ed., McGraw Hill Education 2010, India.
[3] D. R. Stinson, "*Cryptography Theory and Practice "*, Taylor & Francis Group, 3rd. edition, pp.324-325, 2006.
[4] D. H. Lehmer, "*Mathematical Method in large-scale computing unit "*, Proceeding of the second symposium on Large-scale Digital Computing Machinery, Harvard University Press, Cambridge, Massachusetts, pp. 141-146, 1951.
[5] Green, Smith, Klem, *"Empirical Tests of an Additive Random Number Generator"*, Journal of the ACM Vol.-6, issue 4:pp. 527-537, Oct 1959.
[6] D. E. Kunth, "*The Art of Computer Programming, Vol 2: Semi-Numerical Algorithms"* , 2nd ed., Addison-Wesley 1981.
[7] R. C. Tausworthe,, " *Random numbers generated by linear recurrences modulo 2 "*, Mathematics of computation **19**, pp. 201-209.
[8] J. Eichenauer and J.Lehn, "*A non-linear congruential pseudorandom number generator"*, Springer journal Statistische Hefte **27**, pp. 315-326.
[9] James, Frederick, *"A review of pseudorandom number generators." Computer Physics Communications"*, pp. 329-344, 1990.
[10] Runkin et al., *"Statistical test suite for random and pseudo random number generators for cryptographic applications"*, NIST special publication 800-22, 2001.
[11] Stephen K. Park, Keith W. Miller, "*Random Number Generators: Good ones are hard to find"*, Communications of the ACM **31**, 1192-1201.
[12] G. Marsaglia, *"Random numbers fall mainly in the planes"*, Proceedings of the National Academy of Sciences 61, pp. 25-28, 1968.
[13] N. Zierler and J. Brillhart, *"Information and Control"* 13, pp.541-554, 1968, 14 pp. 566-569, 1969, 15 pp. 67-69, 1969.
[14] J. Eichenauer and J.Lehn, *"A non-linear congruential pseudorandom number generator*", Springer journal Statist. Papers 27 pp. 315-326.
[15] J. Eichenauer, A. Topuzoglu, *"On the period length of congruential pseudorandom number sequences generated by inversions"*, Journal of Computation of Applied Mathematics **31**, pp. 87-96, 1990.
[16] T. G. Lewis, and W. H. Payne, "Generalized feedback shift register pseudorandom number algorithms", Journal of ACM 20, pp. 456–468, 3 July 1973.

[17] M. Matsumoto amd Y. Kurita, "Twisted GFSR generators" ACM Transactions on Modeling and Computer Simulation 2, pp. 179-194, 1992.

[18] M. Matsumoto and K. Yoshiharu, "Twisted GFSR generators II", ACM Transactions on Modeling and Computer Simulation 4, pp. 245-266, 1994.

[19] M. Matsumoto, T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", ACM Transactions on Modeling and Computer Simulation 8 (1), pp. 3-30, 1998.