sciendo International Journal of Mathematics and Computer in Engineering 1(1) (2023) 25–44

International Journal of Mathematics and Computer in Engineering

https://sciendo.com/journal/IJMCE

Review Study

A study on comparison of pseudorandom number generator

Viktória Padányi[1†], Tamás Herendi[1‡]

[1]University of Debrecen, Faculty of Informatics, Debrecen, Hungary

Communicated by Haci Mehmet Baskonus; Received: 22.03.2023; Accepted: 15.05.2023; Online: 09.06.2023

Abstract

In this paper, we have collected some theoretical and practical pseudorandom number generators (PRNGs) with uniform distribution. We have also analyzed some of the most prevalent statistical tests and we present some observations and statistical tests of the sequences generated by the described generators.

Keywords: Pseudorandom number generators, statistical tests, uniformly distributed sequences, comparison of pseudorandom number generators, discrepancy.
AMS 2020 codes: 11K45; 11B50; 65C10.

1 Introduction

A.Kolmogorov introduced a new notion of complexity and he first published on the subject in 1963. Independently and nearly at the same time, G. Chaitin and R. Solomono worked out a similar complexity theory. Their basic idea was to express the entropy of a sequence of symbols by the length of the shortest algorithm which can produce it. The verification of the goodness of pseudorandom number generators (PRNGs), the algorithmic characterization of random phenomena and the universal computer learning algorithm constituted the most crucial parts of their research. According to Kolmogorov *et al.* a sequence of symbols are considered as random if the shortest algorithm describing it contains almost the same amount of information as the sequence itself. It is called the information content of the sequence and is incompressible [1–4].
Modeling of randomness can be theoretical and practical, while the latter can be approached from two directions: physical and arithmetical. However, in practice, arithmetic modeling is more convenient to use.

1. Physical model

The physical model is based on random processes occurring in nature. Using and transforming the stochastic features of these processes enables the generation of random number sequences. These are, therefore true random numbers (TRNs). However, the construction and use of equipment to measure and transform the values of physical and chemical processes such as radioactive decay, noise source (often a resistor or a semiconductor diode), diusion, etc., aggravate the original purpose to such an extent that this type of solution has limited use.

†Corresponding author.
Email address: padanyi.viktoria@inf.unideb.hu

sciendo
ISSN 2956-7068
doi:10.2478/ijmce-2023-0003

2. Arithmetic model

One can use mathematical algorithms to generate sequences of numbers, but obviously, they are not true random numbers due to the nature of the approach. As a distinction, these are called pseudorandom numbers (PRNs). In the following, we distinguish two main types: theoretical and practical PRNs.

We consider three key factors of the quality of PRNs and TRNs, and in this paper, we are going to present them in detail:

1. Statistical properties: They are for testing how chaotic a number sequence is and how close to the expected distribution is.
2. Period length: It is used for analyzing when a previous sample number sequence returns.
3. Discrepancy: It measures the quality of uniformity of a finite random point set.

One may define PRN sequences in several ways. Some of these definitions can be used to construct practical PRNGs and some are good only for theoretical considerations. In the theoretical case, the definitions may use operations on arbitrary precision numbers (or on an infinite set). In contrast, in the practical case, one can use operations only on fixed precision numbers (or on a finite set, usually with some algebraic structure). In both cases, the natural requirement is uniform distribution. In the practical case, the definition of a PRNG can be interpreted as a recursive algorithm to compute the members of the sequence from some finitely many previous values.

PRNGs were needed even in the early ages of computing. In the course of our work, we collected some famous and less popular PRNGs, studied what important properties they have, and the beneficial attributes associated with dierent applications. We attempted to dive into several sequences, from classical (early) generators to modern ones with more detailed or better properties. The first practical random number generator was John von Neumanns Middle Square Method (MSM) which was introduced in 1946. Although its basic properties were not proven, it was an interesting concept to construct a uniformly distributed PRNG based on simple arithmetic and fast enough to execute with ENIAC. Later N. Metropolis adopted the idea of binary number systems and R.R. Coveyou modified it for his simplified MSM.

One of the most significant species of PRNGs is generators based on arithmetic in residue class systems. Because of their periodicity, all PRNs can be expressed in an inefficient way by linear congruential generators. Shuffing methods can be used very efficiently to generate random values by scrambling their seed members. This usually requires only a few simple operations. Moreover, there is a group of generators based on particular ideas. For example, the Xorshift uses a bitwise xor operation between the seed elements. Finally, one may construct new algorithms by embedding generators into each other.

Before a PRNG is used, one should know whether it is suitable for the intended purposes. Knowing the underlying algorithm is usually not informative enough. In many cases, the users do not even care about the technical details at all since the general performance observations are more relevant. There are thousands of statistical tests which can be utilized to express properties of randomness e.g., autocorrelation and uniformity of the different patterns. Analyzing the histogram and similar properties of sequences may show a uniform distribution. There are test batteries for comprehensive testing of sequences. The most well-known, among others, are D. E. Knuths tests, Diehard, Dieharder, NIST, and TestU01. In section 5, we collected the results of the NIST statistical test suite, which is particularly designed for testing cryptographic properties.

2 Theoretical model

This approach of PRNs is mainly used for theoretical observations of numerical computations. The most well-known areas of application are the Monte Carlo methods. Theoretical sequences are usually defined by some function over an infinite mathematical structure (most frequently over real intervals). The quality of such sequences is measured by some numerical properties, which are determined analytically. Based on their proven attributes, in practice, an approximate sequence can be used instead, where an error term can be determined.

2.1 Properties of number sequences

There are a couple of ways to analyze PRN sequences, both from theoretical and practical points of view. In this section, we define some tools for expressing the quality of the sequences.

Definition 1. Let $I = [0,1]$, $s \in \mathbb{Z}^+$, $\mathscr{B}$ is a family of Lebesgue measurable subsets of I^s and P be a point set consisting of $x_1, \cdots, x_N \in I^s$. Based on Niederreiter's notation (e.g., in [5]), we define the discrepancy of P according to the Lebesgue measure by the following

$$D_N(\mathscr{B}, P) = \sup_{B \in \mathscr{B}} \left| \frac{A(B,P)}{N} - \lambda_s(B) \right| ,$$

where λ_s is the s-dimensional Lebesgue measure and $A(B;P)$ is the cardinality of $P \cap B$.

We denote the star discrepancy of P by $D_N^*(P) = D_N(\mathscr{J}^*, P)$, and the (extreme) discrepancy of P by $D_N(P) = D_N(\mathscr{J}, P)$, where $\mathscr{J}^*$ and $\mathscr{J}$ are the families of all subintervals of I^s of the form

$$\prod_{i=1}^{s} [0, a_i) \quad \text{and} \quad \prod_{i=1}^{s} [a_i, b_i) \quad (\text{where } a_i, b_i \in [0,1]) ,$$

respectively.

The star and extreme discrepancies are related by

$$D_N^*(P) \leq D_N(P) \leq 2^s D_N^*(P) . \; [5, \; Proposition \, 2.4]$$

Notations 1 *With the notations of Definition 1, let u be a sequence of points in I^s. We denote the discrepancy of u by*

$$D_N(\mathscr{B}, u) = D_N(\mathscr{B}, \{u_0, \cdots, u_{N-1}\}) .$$

Definition 2. Let u be a sequence in $[0,1]$. Via [9], we say that u is uniformly distributed in the interval $[0,1]$ if

$$\lim_{N \to \infty} \frac{1}{N} |\{n \leq N | u_n \in [a,b)\}| = b - a , \quad \text{for all } 0 \leq a \leq b \leq 1.$$

Remark 1. The sequence u is uniformly distributed in $[0,1)$ if and only if $\lim_{N \to \infty} D_N(\mathscr{J}, u) = 0$.

Based on the notion of the discrepancy, one can define the uniform distribution property of sequences over finite structures.

Definition 3. Let A be a finite nonempty set and u be a sequence in A. One can say that u is uniformly distributed in A if

$$\lim_{N \to \infty} \frac{1}{N} |\{n \leq N \mid u_n = a\}| = \frac{1}{|A|} , \quad \text{for all } a \in A .$$

Definition 4. Let A be a finite set, and u be a sequence in A. One can say that u is periodic in A with a period length $\rho \in \mathbb{N}$, if there exists $\rho_0 \in \mathbb{N}$ preperiod, such that [9]

$$u_{n+\rho} = u_n \quad \text{for all} \quad n \geq \rho_0.$$

Remark 2. Let A be a finite set and u be a periodic sequence in A with period length ρ. Let $u_N, \cdots, u_{N+\rho-1}$ be a full period. Then u is uniformly distributed if and only if

$$|\{N \le n < N+\rho \mid u_n = a\}| = \frac{\rho}{|A|}\ , \quad \text{for all } a \in A$$

This means that the period length is a multiple of the cardinality of the base set: $\rho = k \cdot |A|$ with some $k \in \mathbb{Z}$. Then, every element of A appears exactly k times in a full period.

Remark 3. According to Remark 1, a sequence has good uniform properties if the value of the discrepancy approaches zero rapidly. Hence, the faster it converges to zero, the better it behaves, for instance, in numerical approximations. However, if a uniformly distributed sequence is periodic, the discrepancy will be zero or very low for one period. It implies a strong geometrical structure of the sequence. Fundamentally, the fact that the discrepancy sometimes approaches zero means that the sequence is excessively regular in that segment. The repeated occurrence of a zero discrepancy indicates some pattern in the sequence, which makes it poor [5].

A sequence defined by a finite recursion over a finite set is always periodic. Therefore, if the base set is some algebraic structure, the sequence satisfies several homogeneous linear recursion, e.g., $x_{n+\rho} = x_n$. The order of the minimal linear recursion can be a unit of measure for the randomness of the sequence.

Remark 4. For a given sequence according to the definition, there are infinitely many possibilities for the values of the preperiod (ρ_0) and the period length. For the sake of simplicity, we will assume that both the preperiod and the period length are the unique minimal.

There are several further measures of pseudorandomness. Among others, A. Sárközy and C. Mauduit in [6] introduced some for binary sequences. Based on their notions, they settle a condition on pseudorandomness which shows some similarity to the one defined by A. Kolmogorov.

2.2 Theoretical random number sequences

Theoretical RNSs usually have the remarkable feature that their properties can be proven via mathematical instruments. Based on these proven properties, the approximate (practical) sequences can be qualified and their proper usage can be verified.

1. Weyl-sequence
   The general Weyl-sequence [7] is defined in the form $w_n = \{n \cdot \alpha\}$, where $\{x\}$ denotes the fractional part of x. One can prove that w_n is uniformly distributed in $[0,1]$, if and only if α is an irrational number. It is also known [see p122, Kuipers-Niederreiter] that the discrepancy of w_n depends on some arithmetic properties of α. If α has a bad rational approximation, then the corresponding Weyl-sequence has a better discrepancy. More precisely,

   $$D_N(w) = O\left(N^{-\frac{1}{\eta}+\varepsilon}\right) \qquad \text{for all } \varepsilon > 0\ ,$$

   where η is the type of α, i.e. α can be approximated by rationals in the form $\frac{p}{q}$ better than $\frac{1}{q^\eta}$.

2. Van der Corput sequence

   Van der Corput in [8] defined a sequence of rational numbers based on digit reflection. Let $2 \le b \in \mathbb{N}$. If the digit expansion of $n \in \mathbb{N}$ is

   $$n = \sum_{i=0}^{k} a_i b^i \tag{1}$$

   in base b, where $0 \le a_i < b$ for all $j \ge 0$, we will denote the radical-inverse of n in base b by

   $$\bar{n} = \sum_{i=0}^{k} a_i b^{-i-1}\ .$$

Thus $\bar{n}$ is obtained from n by a symmetric reflection of the expansion (1) to the fractional point. Clearly $\bar{n} \in [0,1)$ for all $n \in \mathbb{N}$.

For an integer $b \geq 2$, the van der Corput sequence in base b is the sequence $(x)_b$ with $x_n = \bar{n}$ for all $n \in \mathbb{N}$. If S_b is the van der Corput sequence in base b, then $D_N^*(S_b) = O(N^{-1}\log N)$ for all $N \geq 2$ with an implied constant depending only on b. This follows from a more general result shown in [5].

3. Sequences generated from the digits of irrational numbers

   Let α be an irrational number, $b > 1$ an integer. The digit expansion of α in base b is an infinite sequence. An α is called normal to base b if the relative frequency of each word of length k in the digit expansion of α is asymptotically b^{-k}. M. Borel [10] proved that almost all real numbers are normal with respect to the Lebesgue measure. However, it is not obvious whether an arbitrary number is normal in base b. Normality is not known in the case of π, but the experiments so far do not contradict its possibility.

4. de Bruijn sequence

   The de Bruijn sequence of type (n,k) [11] is the shortest sequence containing all words of length k over an alphabet A with cardinality n. It can be proven that minimality means that such a sequence contains every word of length k exactly once. This yields that the k-block words are uniformly distributed.

   For example, let $A = 0,1$, that means $n = 2$, and let $k = 3$. All two-letter words of three lengths: $000,001,010,011,100,101,110,111$. The words 0001011100 and 0001110100 all satisfy the definition, so they are both $(2,3)$-type De Bruijn-words.

   It is clear that the length of the (n,k)-type de Bruijn sequence is $n^k + k - 1$ and the prefix of the sequence of length $n^k + t - 1$ will contain each word above A of length t exactly n^{k-t} times. In such a manner, this sequence is strongly uniformly distributed.

3 Definitions and classification of PRNGs

In this section, we present a non-exhaustive list of some of the most well-known or historically important generators. As a primary point of view, the investigated generators can be classified by their definition. The generators, in practice, use some recursive relation with bounded depth. In the implementations, the recursion is usually replaced by a seed. This seed is modified iteratively and the random sample is extracted from it. Due to the above properties, such sequences are obviously periodic. Intrinsically, all of the observed PRNGs operate based on some form of recursion. In each case, there is a seed and one can calculate the random values extracted from it. Using this, we can classify PRNGs according to the properties of the recursion.

1. PRNGs based on recursions with modular arithmetic

   A large class of PRNGs uses modular arithmetic as a base operation for computing elements in the sequences.

   (a) Homogeneous linear methods

   In these methods, the sequences are defined by a linear recurrence relation with no constant terms in their definition.

   i. D. H. Lehmer's Multiplicative Linear Congruential Generator (MLCG)
   MLCG is also called Power Residue Method (PRM). The classical definition is

$$u_{n+1} \equiv x \cdot u_n \pmod{m} \quad (n = 0,1,\cdots),$$

   which can be expressed in the form

$$u_{n+1} \equiv x^n \cdot u_0 \pmod{m} \quad (n = 0,1,\cdots).$$

D. H. Lehmer developed this generator for ENIAC with parameters $x = 23, m = 10^8 + 1, u_0 = 47594118$. The period length was 5882352. When choosing the coefficient x, one has to try to find a large order element, which means a long period. The order of an element $\mathrm{ord}(x)$ is the smallest positive e, such that $x^e \equiv 1 \pmod{m}$ [12, 13].

ii. Generalized Feedback Shift Register Generator (GFSR)
This generator was introduced by T.G. Lewis and W.H. Payne in 1973. GFSR is a widely used PRNG based on the linearly recurring equation

$$x_{n+k} = x_{n+l} \oplus x_n \quad (n = 0, 1, \cdots),$$

where $k > l > 0$, $s > 0$, $x_n \in \{0,1\}^s$ and $\oplus$ is the bitwise exclusive-or operation. With carefully chosen k and l, the period length is $2^k - 1$, which is a theoretical upper bound. The generator is very fast and using the same parameters with different word sizes, one can generate a different sequence [14].

iii. Linear Recurrence Sequence Generator (LRS)
LRS is a generalization of LCG. One can prove that with properly chosen coefficients $a_0, \cdots, a_{k-1}$, and initial values $u_0, \cdots, u_{k-1}$, the sequence defined by the recurrence relation

$$u_{n+k} \equiv a_{k-1}u_{n+k-1} + \cdots + a_0 u_n \pmod{m} \quad (n = 0, 1, \cdots)$$

is uniformly distributed. [9] describes a construction for the coefficients when $m = 2^t$ with some $t \in \mathbb{N}$. It is shown that the coefficients can be chosen from the set $\{0,1\}$, with the exception $a_0, a_1 \in \{0,1,2,3\}$. Furthermore, the number of nonzero coefficients can be reduced to six, and the period length can be 2^{s+k}.

iv. Twisted Generalized Feedback Shift Register Generator (TGFSR)
TGFSR was developed by M. Matsumoto and Y. Kurita in 1992. An improvement of the GFSR: before the xor operation, one of the words is "twisted."

$$x_{n+k} = x_{n+l} \oplus x_n A \quad (n = 0, 1, \cdots),$$

where $A \in \{0,1\}^{s \times s}$ is a square matrix. The period of the generated sequence attains the theoretical upper bound $2^{ks} - 1$. With this modification, the memory requirements are significantly reduced for the same period length [15, 16].

(b) Non-homogeneous linear method
In this method, the sequence is defined by a linear recurrence relation with a constant term in its definition.

i. Linear Congruential Generator (LCG)
This generator is the generalization of MLCG. LCG was published in 1958 by W. E. Thomson and independently by A. Rotenberg in 1960 [17, 18]. It is one of the most influential and studied generators. LCG is defined by the recurrence relation:

$$x_{n+1} \equiv (ax_n + c) \pmod{m} \quad (n = 0, 1, \cdots),$$

where a, c, m and x_0 can be chosen such that the generator has a full period of length m. Large values of m enable to have long cycles. It is a fast generator with small memory consumption, but one that fails on many tests [12].
If there is a non-homogeneous linear recursion, that is first-order, which means that it depends on only one next element, it can be transformed into a homogeneous second-order one. In theory, there are non-homogeneous linear generators from which the constant term (non-homogeneous term) can be expressed and homogenized as a whole. It may not be worth looking for homogeneous methods, as in the final result, all of them can be equated to a homogeneous method.

(c) Hybrid methods

In these methods, the sequences are defined by linear methods, but the outputs are modified by some additional operations.

i. Mersenne Twister (MT)
MT was developed in 1997 by M. Matsumoto and T. Nishimura. It is a further improvement of TGFSR.

$$x_{n+k} = x_{n+l} \oplus y_n A \quad (n = 0, 1, \cdots) \,,$$

where y_n is the concatenation of the upper $s-r$ bits of x_n and the lower r bits of x_{n+1} (r is a properly chosen parameter). The period length can be arbitrary, but with the suggested – and widely used – parameters, it is $2^{19937}-1$, which is a Mersenne prime. They perform quite well on statistical tests, but it has a low order linear dependency, which yields easy prediction [19].

ii. Well Equidistributed Long-period Linear Generator (WELL)
WELL was developed by F. Panneton, P. L'Ecuyer, and M. Matsumoto in 2006. Based on similar approaches as MT, but fine-tuned for performance and better statistical properties, it is much faster than MT. The period length of the sequence can supersede the period length of MT [20].

iii. Wichmann-Hill algorithm (WH)
Described by B. A. Wichmann and I. D. Hill in 1982. The idea is a special combination of three "independent" LCGs. Let x_n, y_n, and z_n be pseudo-random sequences generated by different LCGs. The corresponding moduli are m_x, m_y, and m_z. Then the sequence

$$u_n = \left\{ \frac{x_n}{m_x} + \frac{y_n}{m_y} + \frac{z_n}{m_z} \right\}$$

is uniformly distributed, here $\{x\}$ yields the fractional part of x.
The period length is the product of the period length of the three base sequences. Their suggested example is $2.7 \cdot 10^{13}$ [21, 22].

iv. Additive Congruential Random Number Generator (ACORN)
This generator was introduced by R. S. Wikramaratna in 1988. Theoretically, one generator provides an infinite set of PRN sequences $x^{(k)}$, where $k \in \mathbb{N}$. For all k, the sequence has an initial value $x_0^{(k)}$. The recurrence is given by

$$x_{n+1}^{(0)} = x_n^{(0)} \quad \text{and} \quad x_{n+1}^{(k)} = \left\{ x_n^{(k)} + x_{n+1}^{(k-1)} \right\} \,.$$

The main advantages of ACORN are the speed of execution, long period length and simplicity of coding.
Remark 5. WH and ACORN provide random numbers from the interval $[0, 1]$, but since all computations can be executed with fix-point representation, the generated numbers can be transformed into integers by a simple digit shift [23].

v. Permuted Congruential Generator (PCG)
PCGs were introduced and observed by M. E. O'Neill in 2014. The idea is to apply an output permutation on a simple but good-quality LCG. The statistical properties are improved, and the efficiency and speed remain at the same magnitude [24].

(d) PRNGs based on non-linear recursion

In these methods, the sequences are defined by some general modular recursion.

i. Quadratic Congruential Generator (QCG)

The linear congruential method can be generalized to such as a quadratic congruential method:

$$x_{n+1} \equiv (dx_n^2 + ax_n + c) \pmod{m}$$

where a, c, m and x_0 can be chosen such that the QCG has a full period of length m. The restrictions are not much more severe than in the linear method. The block distribution properties of the sequence are weaker than those of the LCG [12].

ii. Blum-Blum-Shub (BBS)
This generator is a nonlinear congruential PRNG proposed in 1986 by L. Blum, M. Blum, and M. Shub. It can be regarded as a simplified QCG. The seed of the sequence satisfies the recursion

$$x_{n+1} \equiv x_n^2 \pmod{m} \quad (n = 0, 1, \cdots),$$

where m is the product of two large primes, from the value x_n, a single bit b_n is obtained by some extraction (e.g., least significant bit). BBS was the first theoretically proven cryptographically secure PRNG. Since the computation of a single bit requires the multiplication and division of large integers, it is slower than most generators [25].

iii. Multiplicative Fibonacci sequence (MFS)
The MFS is a modified method of the original Fibonacci sequence. The Fibonacci sequence is similar to the LCG. The next sequence element is equal to the sum of the two previous sequence elements: $F_n = F_{n-1} + F_{n-2}$. This method can be used in the form of multiplication: $F_n = F_{n-1} \cdot F_{n-2}$. This is the Multiplicative Fibonacci method. The recurrence is defined by the quadratic polynomial $p(x, y) = x \cdot y$, with the substitution of the previous values of the sequence.

iv. Power Congruential Generator (PowCG)
It is a modification of the standard LCG. We replace here the arithmetic operations $+$ and $\cdot$ with the corresponding next-level operations $\cdot$ and x^y.

$$x_{n+1} \equiv c \cdot (x_n)^a \pmod{m} \quad (n = 0, 1, \cdots),$$

where a, c, m and x_0 can be chosen such that the generator has a full period of length m.

v. Inverse Congruential Generator (ICG)
This is another modification of LCG. It was the first proposed by J. Eichenauer and J. Lehn in [26]. The authors called Inversive Congruential Generator (ICG). Now, instead of the linear $a \cdot x + c$, we define the recurrence, by the function $a \cdot x^{-1} + c$.

$$x_{n+1} \equiv \left(a \cdot (x_n)^{-1} + c\right) \pmod{m} \quad (n = 0, 1, \cdots),$$

where a, c, m and x_0 can be chosen such that the generator has a full period of length $\varphi(m)$, where $\varphi(.)$ is the Euler's totient function.

2. PRNGs based on other recursion methods

PRNGs usually have their seed from a finite domain and use a particular mapping to generate the new seed from the previous ones. The best is if we can prove the uniform behavior of this mapping, but in many cases, it is hard to do so. This class details some generators where the next seed is calculated based on a special function not listed before. These functions' uniformity - or, equivalently, the randomization property - is not always proven.

(a) MSM-based generators

i. John von Neumann's Middle-Square Method (NMSM)
Neumann's MSM is an interesting way to construct uniformly distributed PRNG since this was the first practical random number generator. In 1946 John von Neumann introduced the method (first published in [27]). It was simple and fast to execute with ENIAC. He used a recursive definition where the initial value x_0 is some $2k$-digit decimal number. For $n > 0$ he defined $x_n \equiv \lfloor x_{n-1}^2/10^k \rfloor \pmod{10^{2k}}$. The period length depends on the initial value. In general, the longest period has a length at most 8^{2k}, but very often, it is much shorter. Practically it is a rather weak generator. If the seed becomes 0, it is 0 for all consecutive members [12].

ii. N. Metropolis Middle-Square Method in Binary Number Systems (MMSM)
In the early '50-s, Nicholas Metropolis investigated the MSM in binary number systems. He showed that in the case of 20-bit numbers, there are only 13 different cycles. The longest period amongst the 13 cases is 142, which is rather short. It is not obvious to recognize this short period because of the long preperiod [12].

iii. R. R. Coveyou's Simplified Middle-Square Method (CSMSM)
Essentially it is a degenerate double-precision MSM. The initial value x_0 is chosen such that $2 \equiv x_0 \pmod 4$ and $x_{n+1} \equiv x_n(x_n+1) \pmod{2^k}$ for $n > 0$. The period length is typically around 2^{k-1} [12].

iv. Generalized Middle-Square Method (GMSM)
This generator is the generalization of John von Neumann's MSM to canonical number systems (CNS). Let $p(x) \in \mathbb{Z}[x]$ be an irreducible polynomial of degree n, and with coefficients $1 = a_n \le a_{n-1} \le \cdots \le a_0 = 2$. The corresponding CNS has only 2 digits: 0 and 1. For the sake of simplicity, one can call the digits bits and the digit representation of algebraic integers in $\mathbb{Z}[\alpha]$ a binary representation.
In the design of the generator, one can use a seed of $m \in \mathbb{N}$ bits. Similarly, as it is done in the original construction, let u be a sequence over $\mathbb{Z}[\alpha]$ defined by the following:

$u_0 \in$ is a random m-bit number;

if $k > 0$, let

$$u_{k-1}^2 = \sum_{i=0}^{h} b_i \alpha^i \ , \text{ with } b_h \neq 0 \ , \ t = \left\lfloor \frac{h-m}{2} \right\rfloor \text{ and}$$
$$u_k = \sum_{i=0}^{m-1} b_{i+t+1} \alpha^i \ .$$

The value of m should be chosen to be large enough, in particular, such that $2m + C_3 > m$, i.e., $m > -C_3$. Another approach is if $t = \lfloor \frac{m}{2} \rfloor$, but then $\frac{m}{2} > -C_3$ should hold [28].

v. Middle-Square Weyl Sequence (MSWS)
To improve the original MSM generator, a simple modification can be applied. As we already know, we can prove that if $GCD(m,a) = 1$, then $x_n \equiv n \cdot a \pmod m$ is uniformly distributed as well. Thus, the modified MSM sequence is the non-discrete version of the original sequence: $y_n \equiv \lfloor y_{n-1}^2/2^k \rfloor + x_n \pmod{2^{2k}}$ [29].

(b) Shuffling methods

In these methods, the sequences are defined by permutations.

i. Knuth's Algorithm M (KnM) (randomizing by shuffling)
Let x_n and y_n be two sequences generated by some algorithms and assume $0 \le y_n < m$ with some m. The output sequence of the generator is z_n. Let k be some (not too large) positive integer, and the seed of the generator is $v = (v_0, \cdots, v_{k-1})$. Initially let $v_n = x_n$ for $n = 0, \cdots, k-1$, then

for all $n \geq k$

$$\begin{aligned} j &= \lfloor k \cdot y_n / m \rfloor \\ z_{n-k} &= v_j \\ v_j &= x_n \ . \end{aligned}$$

The period length is the least common multiple of the period length of x_n and y_n. The computational time is the sum of the computational time of the two base sequences [12].

ii. Knuth's Algorithm B (KnB) (randomizing by shuffling)
It is similar to Algorithm M but uses only one base sequence x_n with assumption $0 \leq x_n < m$. The initial seed v is as before. Then the new sequence $y_0 = x_k$ and for all $n > 0$ [12]

$$\begin{aligned} j &= \lfloor k \cdot y_{n-1} / m \rfloor \\ y_n &= v_j \\ v_j &= x_{n+k} \ . \end{aligned}$$

iii. RC4 based generators
Ron Rivests algorithm for stream cipher key generation was designed in 1987 but kept secret until 1994. It is efficient and uses simple operations. Because of its simplicity and efficiency, it is the basis for several cryptographic protocols, such as WEP (Wired Equivalent Privacy) and WPA (Wireless Protected Access), used in WiFi communications. The seed is an array S of 256 entries of 8 bit values.

$$\begin{aligned} &i \equiv (i+1) \pmod{256} \\ &j \equiv (j + S[i]) \pmod{256} \\ &\text{swap values of } S[i] \text{ and } S[j] \\ &K = S[(S[i] + S[j]) \pmod{256}] \\ &\text{output } K. \end{aligned}$$

[30]

(c) Miscellaneous methods

i. XorShift (XSH)
This is a class of PRNGs – also called shift-register generators – which were described by G. Marsaglia in 2003. The period length is $2^k - 1$, where $k = 32, 64, 96, 128, 160, 192$ in the original paper. It uses a 4-word seed: (x, y, z, w) and three parameters for shifting a, b, c. The elements of the seed are shifted and xor'ed to generate new seed members and the output. The operations are the following:

$$\begin{aligned} tmp &= (x \oplus (x \text{ shl } a)) \\ x &= y \\ y &= z \\ z &= w \\ w &= (w \oplus (w \text{ shr } b)) \oplus (tmp \oplus (tmp \text{ shr } c)) \end{aligned}$$

here shl and shr are the bitwise left and right shifts, respectively. Very fast generator [31].

ii. Legendre symbol (LeS)
The Legendre symbol has become one of the most useful methods in number theory. It can have three values: $1, -1$, or 0. It has a prime number in its denominator. The value of the Legendre symbol is $\frac{a}{b}$ and its value is 1 if the squared remainder is a mod b. The value of a is -1 if it is not a quadratic remainder and 0 if a is divisible by b. There is exactly the same number of squared residuals as non-squared residuals. From a theoretical point of view, it is a finite sequence with a uniform distribution [6].

A number sequence generated using the Legendre symbol is a good source of PRNs. Mathematicians have been using this random number sequence for quite a long time and it has been regarded as a good source of random numbers. Not only is it a good random number source from a practical point of view, but one can also say it is a cryptographically secure generator [32].

iii. Encryption-based generators (EncG)
Let Enc : $\mathscr{P} \times \mathscr{K} \to \mathscr{P}$, be a cryptographic function (encryption), where $\mathscr{P} = \{0,1\}^n$, and $\mathscr{K} = \{0,1\}^m$ with some $n, m \in \mathbb{N}$ and construct a sequence of words by the recursion $m_{n+1} = \text{Enc}(m_n, k)$, where $k \in \mathscr{K}$ is the secret key - in our case a properly chosen parameter. A typical application of such sequences is, for instance, the Output Feedback Mode of Operation of symmetric encryption functions. (See, FIPS PUB 81 [33].) Assuming the expected good quality of the mapping Enc, the derived sequences usually have very good statistical and cryptographic properties. The only disadvantage of these generators is their relatively low speed.

4 Analytical properties of PRNGs

We have already discussed theoretical measures of PRNGs in Subsection 2.1. However, we have seen that a periodic sequence does not perform very well on these. Since the practical generators are always periodic, they don't have good global quality scores. Nevertheless, periodic sequences may still deliver satisfactory local results on the mentioned observations – in particular, if they have a large period length.

Below, we consider further analytical attributes of the generators, such as period length, computational complexity and memory usage. The properties of the PRNGs of Section 3 are presented in Table 1.

Since there is a wide range of platforms they are implemented on, it is hard to compare the speed of the generators. Some perform better on constrained devices, while others are more efficient on CISC (Complex Instruction Set Computer) processors. According to this duality, instead of the speed of the algorithms, we highlighted the number of complex (slow) and simple (fast) operations. One can find available implementations for most of the generators of Section 3. Those without released implementations are either weak or very simple.

The entries of the table are expressed in terms of the parameters of the generators:

- s: the length (bit size) of the data (seed). If the generator under consideration is based on modular arithmetics, then for the used modulus $m \approx 2^s$.
- d: the decimal length of the data. Some of the early generators are defined in decimal representation.
- k: the farthermost previous seed value used to determine the next one or the dimension of the seed vector.

In Table 1, we assumed the following:

in the column *Period length*, the best (largest) possible values are shown;

the column *Operations* displays the complexity of the algorithms, i.e., the number of necessary operations to calculate a single random sample. Operations use the given size of data (s bit, or d decimal digit).

The complexity of the LRS generator can be reduced to 5 additions by choosing the proper parameters.

With the widely used parameters, the period length of the MT is $2^{19937-1}$.

With the widely used parameters, the period length of the WELL is 2^{44497}.

Neumann's MSM is originally defined for decimal numbers.

The original construction of Metropolis with 20-bit numbers has maximal period length 142.

The time complexity depends on the underlying cryptographic algorithms.

Table 1 General properties of the generators

Generators	Period length[1]	Operations[2]		Memory (bits)
		Slow	Fast	
MLCG	2^s	1 mul, 1 mod		$2s$
GFSR	2^k		s-bit xor	$k \cdot s$
TGFSR	2^{ks}	$s \times s$-bit m.mul	s-bit xor	$k \cdot s + s^2$
LCG	2^s	1 mul, 1 mod	1 sum	$4s$
LRS[3]	2^{k+s}		k sum	$k \cdot s$
MT[4]	2^{ks}	$s \times s$-bit m.mul	s-bit xor	$k \cdot s + s^2$
WELL[5]	2^{ks}	8 $s \times s$-bit m.mul	9 s-bit op	
WH	2^{3s}	3 LCG, 3 div	2 sum	$12s$
ACORN	2^{ks}	k mod	k sum	$k \cdot s$
PCG	2^{128}	mul	s-bit perm,xor	$4s$
QCG	s^s	4 mul, 1 mod	2 sum	$4s$
BBS	2^s	1 mul, 1 mod		$2s$
MFS	2^{2s}	1 mul, 1 mod		$3s$
PowCG	2^s	s mul, 1 mod		$4s$
ICG	2^s	1 mul, 1 inv, 1 mod	1 sum	$4s$
NMSM[6]	8^d	1 mul	1 xor, 1 shr	$4d$
MMSM[7]	2^s	1 mul	1 xor, 1 shr	s
CSMSM	2^s	1 mul		s
GMSM	2^s	1 mul	1 xor, 1 shr	s
MSWS	2^{2s}	1 LCG, 1 mul	1 sum	$5s$
KnM	2^{2s}	2 gen, 1 mul, 1 div		$2 \cdot * + k \cdot s$
KnB	2^s	1 gen, 1 mul, 1 div		$* + k \cdot s$
RC4	2^{1024}		7 byte op	256 bytes
XSH	2^s		1 xor, 1 shr	s
LeS	2^s	s mod		s
EncG[8]	2^s	*	*	$2s$

The following Table 2 explains the operations referred in Table 1.

Table 2 Explanation of the operations

Slow operations		Fast operations	
mul	multiplication	sum	addition
mod	modular reduction	xor	bitwise xor
div	division	shl, shr	bitwise left and right shift
m.mul	multiplication of a vector by a matrix	op	memory operations
inv	modular inversion	perm	bitwise permutation
gen	sample from an embedded generator		
LCG	sample from an LCG generator		

5 Statistical tests

Before using a PRNG, one needs to know whether it is suitable for the given purpose or not. Knowing the underlying algorithm is usually not informative enough. The users often do not even care about the technical details. This is why it is important to have general performance observations, which can be realized by standard statistical tests. The different tests express the properties from different points of view. But it is still a question of which properties are expected for the application concerned.

Statistical tests of PRNG are usually applications of the general statistical tests for some expected properties of an imaginary perfect uniformly distributed true random sequence. In many cases, the tests count the relative frequency of the appearance of some – in general numerically expressed – property and determine the probability of the given event. For instance, one of the most simple tests is the frequency test. We assume that the observed pseudo random bit sequence is a sequence of independent samples with probability $1/2$ for both the 0s and 1s. The number of 1s (or 0s) of a sequence of length n is expected to have a binomial distribution, with mean $n/2$. Because of the desired distribution, the actual value of the number of 1s has a well defined probability. Setting a level of significance, we accept or reject our hypothesis (i.e., the sequence is uniformly distributed). If we have an initial test – such as the previous frequency test –, we may define an advanced one: testing m different sequence of length n, we get m probabilities. These probabilities should follow a well defined distribution, for which we may execute another statistical test. This provides a probability, as before, and depending on the preset significance level, we accept or reject our hypothesis again. We may continue the described iteration to obtain further higher-level tests. In most cases, however, the practical tests apply only first or second-level tests. There are several ways to execute tests, but the two most commonly used are the Chi-Square and Kolmogorov-Smirnov tests. The chi-square test is a hypothesis test designed to test a statistically significant relationship between nominal and ordinal variables arranged in a binary table. In other words, it shows whether two variables are independent. For sequences with continuous distribution functions. In this test, the random number generators' distribution function $F_n(x)$ is compared to the theoretical cumulative distribution function $F(x)$ [12, 34].

Some of the most well-known test batteries are D. E. Knuths tests [12], the Diehard test suite from G. Marsaglia (1995), the Dieharder written by Robert G. Brown [35] and TestU01 published by P. LEcuyer and R. Simard in 2007 [36].

In the following, we execute a statistical test suite for random and pseudorandom number generators for cryptographic applications. This is designed and recommended by the National Institute of Standards and Technology (NIST).

5.1 NIST statistical test suite

The NIST Test Suite is a statistical package consisting of 15 tests that were developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software-based cryptographic pseudorandom number generators ([37] and [38]). These tests focus on a variety of different types of non-randomness

that could exist in a sequence. The 15 tests can be grouped into five main classes. The 15 tests are the following:

1. Frequency tests (monobit, block)

   Monobit test: Observes the relative frequency of the zeroes and ones. In a true random uniformly distributed sequence, the proportion of the zeroes and ones should be close to $1/2$. The test estimates the distance of the input sequence to a completely equidistributed one.

   Block test: It splits the input sequence into blocks of equal length. The test observes the relative frequency of the zeroes and ones in M-bit blocks. It is a generalization of the monobit test and determines whether the average frequency of ones in an M-bit block is approximately $M/2$.

   Serial Test: The relative frequency of all m-bit overlapping patterns. It is another generalization of the frequency test. Applying the test, one can determine whether the number of the relative frequencies of the different m-bit overlapping patterns is nearly as one would expect in the case of a true random sequence. In an ideal random sequence, the probability of the appearance of an m-bit pattern depends only on m.

2. Extremal behavior tests (runs)

   Run test: Counts the number of consecutive identical bits of length $\geq k$. In general, a run is a subsequence of consecutive identical values in a sequence. The test determines whether the number of runs of different lengths is as expected for a true random sequence. If the number of runs is high, then the sequence behaves regularly.

   Runs of ones in blocks test: The focus of the test is the longest run of ones within M-bit blocks. Similar to the previous test, but it does not count all runs. The longest run in an ideal random sequence has a well-described distribution. Theoretically, there is no difference between the longest run of ones and the longest run of zeros, so there is no need for separate tests for the different bits.

3. Linear dependency and periodicity tests

   Binary Matrix Rank Test: Tests the distribution of the rank of a random matrix.

   Linear Complexity Test: Tests the shortest LFSR expression satisfied by the sequence. Any periodic sequence satisfies a linear recurrence and the goal of the test is to determine the shortest one the sequence satisfies. If it is large enough, the sequence can be regarded as true random.

   Discrete Fourier Transform Test: It is a spectral test of the input sequence. Applying the Discrete Fourier Transform, we transform the sequence of signals to the frequency space and observe the behavior there.

4. Pattern tests (template matching)

   Non-overlapping Template Matching Test: Tests the relative frequency of predefined patterns. With the test, one can detect whether a sequence contains an irregular amount of occurrences of a given non-periodic pattern. During the test, an m-bit window shifted to identify a specific m-bit pattern. In the case of a missing pattern, the window slides a one-bit position. Otherwise, the window shifted exactly by m-bit.

   Overlapping Template Matching Test: Same as before, but allowing overlap in the case when a pattern is found.

   Maurer's Universal Statistical Test: Tests the distance between previously undetermined patterns. It expresses the possibility and ratio of compression of the sequence. According to the theory of Kolmogorov-complexity, a true random sequence can not be significantly compressed.

   Approximate Entropy Test: based on a similar argument as the serial test, but instead of counting relative frequencies, it calculates the empirical entropy of the patterns.

5. Random walk tests (cumulative sum, excursion)

   Cumulative Sums (Cusum) Test: Test the behavior of the cumulative sums of the sequence. The zeros of the sequence are replaced by -1, whence the value of the cumulative sum should be close to zero. The cumulative sum of the modified sequence is equivalent to the distance of the excursion from zero (i.e. the cumulative sum may be considered as a random walk). We observe the values of the cumulative sum of the partial sequences. In a true random sequence, the random walk should oscillate around zero. If the cumulative sum is large, the sequence is regarded as non-random.

   Random Excursions Test: A cycle of a random walk is when the excursion returns to 0. The test counts the number of cycles having exactly K visits in a particular state. The states are $-4, -3, -2, -1$ and $+1, +2, +3, +4$.

   Random Excursions Variant Test: Similar to the previous. It tests the total number of times that a particular state is visited. The number of occurrences of the observed states should follow a particular distribution. The states are $-9, -8, \ldots, -1$ and $+1, +2, \ldots, +9$.

A more detailed description of the tests can be found in [38]

5.2 Experimental results

We implemented the PRNGs discussed in Section 3 in C++ and ran on an Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz with 64GB RAM. We tested the obtained sequences with the NIST test suite in the same environment. The implementations and the detailed results are available at `http://www.prng.hu`. In the tests, 10^9-long $0,1$-bit sequences were used for each generator. Each sequence was split into 1000 equal-length subsequences to have reasonably large independent test cases. All fifteen tests were applied for every sequence. During the test, we used the default setup of the test suite:

1. Block Frequency Test - block length(M): 128
2. NonOverlapping Template Test - block length(m): 9
3. Overlapping Template Test - block length(m): 9
4. Approximate Entropy Test - block length(m): 10
5. Serial Test - block length(m): 16
6. Linear Complexity Test - block length(M): 500

The most essential results are collected in Tables 3 and 4. Table 3 contains the p-values computed by the different tests. The significance level is set to 0.01.

The minimum pass rate for each statistical test – with the exception of the random excursion (variant) test – is approximately 0.981819 for a sample size of 1000 binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately 0.979517 for a sample size of 609 binary sequences.

The generators which did not pass a particular test are marked by an * in the tables. For those tests which stand off several subtests with different parameters – cumulative sums, non-overlapping template, random excursions, random excursions variant, serial – we displayed the average of the results instead.

Table 3 NIST test results: *p*-values

p-values	Freq.	BFreq.	Cum.Sums	Runs	L.Run	Rank	FFT	NOTemp.	OTemp.	Univ.	AEntr.	RExc.	REVar.	Serial	LinComp.
MLCG	0.53	0.15	0.38	0.42	0.18	0.26	0.86	0.43	0.13	0.29	0.05	0.52	0.60	0.32	0.99
GFSR	0.70	0.13	0.91	0.15	0.49	0.71	0.76	0.48	0.50	0.86	0.22	0.40	0.38	0.65	0.51
TGFSR	0.37	0.67	0.57	0.95	0.18	0.12	0.84	0.52	0.51	0.44	0.99	0.44	0.06	0.16	0.71
LCG	0.52	0.59	0.53	0.04	0.11	0.73	0.03	0.42	0.37	0.59	0.99	0.46	0.38	0.47	0.03
LRS	0.33	0.22	0.63	0.68	0.27	0.30	0.55	0.54	0.62	0.47	0.31	0.55	0.37	0.50	0.87
MT	0.76	0.73	0.79	0.23	0.03	0.28	0.67	0.46	0.65	0.87	0.77	0.53	0.50	0.48	0.07
WELL	0.60	0.11	0.77	0.80	0.38	0.65	0.46	0.49	0.01	0.45	0.37	0.61	0.48	0.37	0.03
WH	0.38	0.98	0.07	0.43	0.06	0.49	0.08	0.51	0.24	0.17	0.49	0.65	0.38	0.82	0.69
ACORN	0.27	0.21	0.71	0.62	0.87	0.77	0.01	0.52	0.98	0.80	0.90	0.48	0.54	0.74	0.22
PCG	0.39	0.13	0.39	0.26	0.75	0.85	0.44	0.46	0.26	0.48	0.56	0.65	0.59	0.15	0.98
QCG	0.00*	0.00*	0.00*	0.00*	0.00*	0.13	0.00*	0.00*	0.00*	0.00*	0.00*	0.20	0.09	0.00*	0.00*
BBS	0.00*	0.00*	0.00*	0.00*	0.02	0.03	0.00*	0.00*	0.00*	0.00*	0.00*	0.54	0.48	0.00*	0.90
MFS	0.00*	0.00*	0.00*	0.10	0.02	0.63	0.20	0.05	0.40	0.00*	0.00*	0.31	0.39	0.00*	0.24
PowCG	0.00*	0.00*	0.00*	0.00*	0.00*	0.08	0.01	0.00*	0.00*	0.00*	0.00*	0.51	0.45	0.00*	0.03
ICG	0.90	0.53	0.46	0.52	0.45	0.35	0.04	0.48	0.34	0.09	0.62	0.40	0.58	0.70	0.77
NMSM	0.28	0.78	0.73	0.71	0.59	0.23	0.10	0.46	0.77	0.95	0.57	0.46	0.55	0.73	0.34
MMSM	0.88	0.39	0.40	0.36	0.09	0.15	0.20	0.52	0.14	0.27	0.07	0.59	0.48	0.26	0.18
CSMSM	0.03	0.46	0.01	0.10	0.39	0.68	0.00*	0.49	0.22	0.98	0.49	0.50	0.43	0.20	0.74
GMSM	0.67	0.39	0.64	0.85	0.57	0.80	0.10	0.46	0.36	0.47	0.63	0.54	0.49	0.41	0.76
MSWS	0.21	0.93	0.35	0.72	0.52	0.46	0.56	0.50	0.33	0.13	0.63	0.44	0.46	0.50	0.61
KnM	0.38	0.67	0.48	0.67	0.78	0.43	0.83	0.52	0.32	0.77	0.28	0.67	0.43	0.12	0.32
KnB	0.34	0.37	0.99	0.54	0.47	0.75	0.95	0.56	0.36	0.02	0.53	0.39	0.56	0.16	0.25
RC4	0.29	0.03	0.80	0.95	0.43	0.55	0.17	0.50	0.55	0.87	0.42	0.40	0.46	0.35	0.16
XSH	0.21	0.21	0.52	0.76	0.27	0.87	0.49	0.53	0.88	0.69	0.39	0.59	0.67	0.05	0.00*
LeS	0.04	0.64	0.07	0.05	0.86	0.64	0.06	0.52	0.09	0.27	0.59	0.47	0.39	0.28	0.12

Table 4 NIST test results: proportion

Proportions (%)	Freq.	BFreq.	Cum.Sums	Runs	L.Run	Rank	FFT	NOTemp.	OTemp.	Univ.	AEntr.	RExc.	REVar.	Serial	LinComp.
MLCG	99.4	99.1	99.4	99.3	98.8	98.6	99.0	99.1	98.2	98.6	98.9	98.8	99.0	99.2	99.4
GFSR	99.3	98.9	99.6	99.0	98.8	98.9	98.3	99.0	98.9	98.9	99.2	99.0	99.0	99.2	99.2
TGFSR	98.8	99.1	99.2	99.2	99.3	99.1	98.8	99.0	98.5	98.6	98.5	99.1	99.1	98.9	98.9
LCG	99.6	99.4	99.6	99.4	98.3	99.1	99.3	99.0	98.8	99.0	98.6	98.9	99.4	98.9	98.9
LRS	98.5	99.3	98.6	99.5	98.9	99.4	98.7	99.0	98.6	99.2	98.9	99.0	99.2	98.5	99.4
MT	98.8	99.2	98.8	98.6	98.8	99.1	99.0	99.0	99.0	99.0	98.9	99.0	99.0	99.2	99.0
WELL	98.8	98.8	98.8	99.0	99.2	98.9	99.1	99.0	98.2	99.4	99.2	98.8	98.9	98.7	98.7
WH	99.0	99.2	99.0	98.6	98.7	99.0	98.5	99.0	99.3	98.0*	99.2	99.1	99.1	99.0	98.9
ACORN	99.2	99.6	99.1	99.4	98.8	99.1	99.2	99.0	98.9	98.0*	99.0	98.9	98.9	99.1	99.2
PCG	99.0	98.6	99.2	98.8	98.3	99.2	99.3	99.0	98.5	98.6	98.4	98.7	99.0	99.4	98.9
QCG	100*	99.0	100*	88.0*	0.0*	99.2	0.0*	100*	0.0*	0.0*	0.0*	99.3	99.2	0.0*	98.7
BBS	100*	98.0*	100*	100*	98.8	98.0*	95.3*	98.4*	99.0	98.0*	98.8	98.7	99.4	100*	99.2
MFS	100*	100*	100*	100*	98.8	99.0	98.8	98.8	98.8	99.0	99.0	99.0	98.8	98.8	98.8
PowCG	100*	100*	100*	100*	99.0	99.0	99.3	99.0*	84.0*	100*	100*	97.3	98.0	100*	99.0
ICG	98.7	99.2	98.7	99.3	98.9	99.2	98.7	99.0	99.1	99.2	99.0	99.0	98.9	98.8	99.1
NMSM	98.9	98.8	99.0	99.2	99.0	98.4	98.8	99.0	98.5	98.2	99.0	99.0	99.0	99.4	99.2
MMSM	99.1	99.3	99.3	99.0	99.0	98.2	98.4	99.0	98.7	97.0*	98.7	99.0	99.0	98.7	99.0
CSMSM	99.7	99.1	99.6	99.5	99.1	98.0*	83.0*	99.0	99.0	98.0*	99.5	98.7	99.2	99.3	99.2
GMSM	98.8	98.4	99.2	99.1	98.9	99.3	98.5	99.0	98.7	99.0	98.7	98.7	99.0	98.9	99.6
MSWS	99.0	99.0	99.4	99.1	99.1	99.1	98.9	99.0	98.6	99.0	98.6	98.6	98.7	98.7	98.8
KnM	98.9	98.0*	99.2	98.8	98.9	98.7	99.2	99.0	99.3	98.8	99.5	98.9	99.0	99.2	0.0*
KnB	99.1	99.1	97.9*	99.0	99.0	99.1	99.2	99.0	98.6	98.9	98.3	99.1	99.1	98.7	99.1
RC4	98.6	99.3	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7	98.7
XSH	99.4	99.1	99.0	99.0	99.0	98.8	99.3	99.0	99.2	98.6	99.0	98.8	99.1	99.2	0.0*
LeS	98.6	98.3	97.9*	99.1	99.2	98.4	98.5	99.0	98.9	98.5	99.0	99.1	99.2	99.3	99.4

6 Conclusion

In the present review, we collected some noteworthy pseudo random number generators and a few theoretical ways to observe them. We also analyze the basic properties of the PRNGs, such as period length, computational speed and memory usage. We found that the period length has a strong relationship with the size of the seed. Typically, it is an inversely proportional relation. We implemented the discussed PRNGs (http://www.prng.hu) and we executed the NIST test suite on the generated sequences. The exact parameters of the generators can be found in the implementations.

As a result of our observations, most of the discussed PRNGs passed all tests of the test battery. In particular cases, however, we found interesting behavior. The most surprising is that LCG and Lehmer's MLCG show a good score, while both of them are proven to be cryptographically insecure. One can find an opposite behavior for the BBS, which performed weak on the tests but has proven to be a secure one. For the generators based on von Neumann's middle square method, we have to increase the number of digits of their seeds considerably to achieve sequences passing the tests. (For instance, the seed of the original NMSM generator was extended to 72-digit.) Compared to it, the improved GMSM generator, which is defined over algebraic number fields [28], requires only a 92-bit seed with similar properties, even for a relatively simple algebraic extension.

7 Declarations

7.1 Conflict of interest:

The authors hereby declare that there is no conflict of interests regarding the publication of this paper.

7.2 Funding:

Not applicable.

7.3 Authors Contribution:

V.P.- Investigation, Formal analysis, Methodology and Writing-Original Draft; T.H.-Resources, Validation, Writing-Review and Editing. All authors contributed to manuscript revision, read and approved the submitted version.

7.4 Acknowledgement:

† The first researcher has been partially supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund.

‡ The second author has been supported by the SETIT Project (no. 2018-1.2.1-NKP-2018-00004), which has been implemented with the support provided by the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

7.5 Data availability statement:

All data that support the findings of this study are included within the article (and any supplementary files).

References

[1] Kolmogorov A.N., On tables of random numbers, Theoretical Computer Science, 207(2), 387-395, 1998.
[2] Kolmogorov A.N., Three approaches to the quantitative definition of information, Problems of Information Transmission, 1(1), 1-7, 1965.

[3] Chaitin G.J., On the simplicity and speed of programs for computing infinite sets of natural numbers, Journal of the ACM, 16(3), 407-422, 1969.
[4] Solomono R., A formal theory of inductive inference: Part I and Part II, Information and Control, 7(1) and (2), 1-22 and 224-254, 1964.
[5] Niederreiter H., Random Number Generation and Quasi-Monte Carlo Methods, Society for Industrial and Applied Mathematics (SIAM), USA, 1992.
[6] Mauduit C., Sárközy A., On finite pseudorandom binary sequences I: Measure of pseudorandomness, the Legendre symbol, Acta Arithmetica, 82(4), 365-377, 1997.
[7] Weyl H., Üeber die Gleichverteilung von Zahlen mod. eins, Mathematische Annalen, 77, 313-352, 1916.
[8] Van Der Corput J.G., Verteilungsfunktionen (erste mitteilung), Proceedings of the Koninklijke Akademie van Wetenschappen te Amsterdam, 38, 813-821, 1935.
[9] Herendi T., Construction of uniformly distributed linear recurring sequences modulo powers of 2, Uniform Distribution Theory, 13(1), 109-129, 2018.
[10] Borel M.E., Les probabilités dénombrables et leurs applications arithmétiques, Rendiconti del Circolo Matematico di Palermo, 27, 247-271, 1909.
[11] Bruijn N.G., A combinatorial problem, Koninklijke Nederlandse Akademie v. Wetenschappen, 49, 758-764, 1946.
[12] Knuth D.E., The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison Wesley, 1981.
[13] Lehmer D.H., Mathematical methods in large-scale computing units, Proceedings of a Second Symposium on Large-Scale Digital Calculating, 26, 141-146, 1951.
[14] Lewis T.G., Payne W.H., Generalized Feedback Shift Register Pseudorandom Number Algorithm, Journal of the ACM, 20(3), 456-468, 1973.
[15] Kurita Y., Matsumoto M., Twisted GFSR generators, ACM Transactions on Modeling and Computer Simulation, 2(3), 179-194, 1992.
[16] Kurita Y., Matsumoto M., Twisted GFSR generators II. ACM Transactions on Modeling and Computer Simulation, 4(3), 245-466, 1994.
[17] Thomson W.E., A Modified Congruence Method of Generating Pseudo-random Numbers, The Computer Journal, 1(2), 83, 1958.
[18] Rotenberg A., A New Pseudo-Random Number Generator, Journal of the ACM, 7(1), 75-77, 1960.
[19] Matsumoto M., Nishimura T., Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator, ACM Transactions on Modeling and Computer Simulation, 8(1), 3-30, 1998.
[20] Panneton F., LEcuyer P., Matsumoto M., Improved long-period generators based on linear recurrences modulo 2, ACM Transactions on Mathematical Software, 32(1), 1-16, 2006.
[21] Wichmann B.A., Hill I.D., Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator, Journal of the Royal Statistical Society: Series C (Applied Statistics), 30(2), 188-190, 1982.
[22] Wichmann B.A., Hill I.D., Generating good pseudo-random numbers, Computational Statistics Data Analysis, 51(3), 1614-1622, 2006.
[23] Wikramaratna R.S., ACORN-A new method for generating sequences of uniformly distributed Pseudo-Random Numbers, Journal of Computational Physics, 83(1), 16-31, 1989.
[24] ONeill M.E., PCG: A family of simple fast space-efficient statistically good algorithms for random number generation, Harvey Mudd College, HMC-CS-2014-0905, 2014.
[25] Blum L., Blum M., Shub M., A Simple Unpredictable Pseudo-Random Number Generator, SIAM Journal on Computing, 15(2), 364-383, 1986.
[26] Eichenauer J., Lehn J., A non-linear congruential pseudo random number generator, Statistische Hefte, 27, 315-326, 1986.
[27] Neumann J.V., Various techniques used in connection with random digits, Applied Mathematics Series, Notes by G.E. Forsythe, in National Bureau of Standards, 12, 36-38, 1951.
[28] Padányi V., Herendi T., Generalized Middle-Square Method, Annales Mathematicae et Informaticae, 56, 95-108, 2022.
[29] Widynski B., Middle-Square weyl sequence RNG, arXiv:1704.00358 [cs.CR], 2022.
[30] Schneier B., Applied Cryptography: Second Edition: Protocols, Algorithms and Source Code in C (Cloth), John Wiley Sons, 1996.
[31] Marsaglia G., Xorshift RNGs, Journal of Statistical Software, 8(14), 1-6, 2003.
[32] Damgård I.B., On the randomness of legendre and jacobi sequences, Conference on the Theory and Application of Cryptography CRYPTO 1988: Advances in Cryptology CRYPTO 88, 163-172, Part of the Lecture Notes in Computer Science, 403, 163-172, 1990.
[33] FIPS-81. DES Modes of Operation. https://csrc.nist.gov/csrc/media/publications/fips/81/archive/1980-12-02/documents/fips81.pdf, 1980.
[34] Akopov N.Z., Martirosyan N.H., The optimal approach for Kolmogorov-Smirnov test calculation in high dimensional space, Mathematical Problems of Computer Science, 44, 138-144, 2021.
[35] Dieharder.https://webhome.phy.duke.edu/ rgb/General/dieharder.php. Website.

[36] Lecuyer P., Simard R., TestU01: A C library for empirical testing of random number generators, ACM Transactions on Mathematical Software, 33(4), 1-40, 2007.
[37] National Institute of Standards and Technology. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications: NIST SP 800-822, 2012.
[38] Rukhin A.L., Soto J., Nechvatal J.R., Smid M.E., Barker E.B., Leigh S.D., Levenson M., Vangel M., Banks D.L., Heckert N.A., Dray J.F., Vo S.C., A statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST Special Publication, 800-822, 2001.