Statistics Surveys
Vol. 16 (2022) 137–181
ISSN: 1935-7516
https://doi.org/10.1214/22-SS136

A brief and understandable guide to pseudo-random number generators and specific models for security

Elena Almaraz Luengo

Group of Analysis, Security and Systems (GASS); Department of Statistic and Operational Research, Faculty of Mathematical Science, Universidad Complutense de Madrid (UCM), Spain
e-mail: ealmaraz@ucm.es

Abstract: The generation of random sequences is the basis of simulation and can be used in many different areas such as Statistics, Computer Science, Systems Management and Control, Biology, Particle Physics, Cryptography or Cyber-Security, among others. It is crucial that the numbers generated were random or at least, behave as such. The fundamental statistical properties required for such sequences are randomness and independence and, from a cryptographic perspective, unpredictability. There is a variety of methods to generate these sequences. The main ones are physical and arithmetic methods. In this work, a detailed study of the main arithmetic methods is carried out. On the other hand, the necessity of secure sequence generation will be analyzed and new lines of ongoing research focusing applications in Internet of Things and new generator designs will be described.

MSC2020 subject classifications: Primary 11K45; secondary 65C10.
Keywords and phrases: Arithmetic methods, congruential methods, CSPRNG, pseudo-random numbers, PRNG, random numbers, simulation, TRNG, IoT.

Received July 2021.

Contents

1 Introduction . . . 138
2 Historical development . . . 142
3 Pseudo-random numbers generators . . . 144
3.1 The middle-square method . . . 146
3.2 Lehmer's method . . . 146
3.3 Linear congruential methods . . . 147
3.3.1 Multiplicative congruential method . . . 148
3.3.2 Mixed congruential method . . . 150
3.3.3 Multiple recursive generators . . . 154
3.3.4 Lagged Fibonacci generator . . . 156
3.3.5 Combined multiple recursive generators . . . 157
3.3.6 Matrix congruential generators . . . 160
3.4 Non-linear congruential generators . . . 160

3.4.1 Inversive congruential generators 161
3.4.2 Knuth's non-linear congruential generator 162
3.4.3 More general formulations 163
4 New lines of research: other methods for generating pseudo-random numbers 164
5 Conclusions 168
References 168

1. Introduction

Random numbers are the essential basis of the simulation. In general terms, the randomness involved in a model is obtained through sequences of numbers that pretend to be random (we will go into this aspect in more detail later) and come from a Uniform distribution in the interval (0,1) ($U(0,1)$) and are obtained through various generators. These random numbers are then conveniently transformed to simulate the different probability distributions required in the model.

In the early days of statistical simulation studies, random number tables were used to carry out the analysis. Later this methodology changed with the rise of computers and more efficient ways of obtaining sequences were established. Not only in statistics and applications such as re-sampling or stochastic processes is simulation very useful (mainly due to the complexity of the problem or the impossibility of its treatment by analytical methods), but in other disciplines simulation studies are of vital importance, for example in the study of physical and biological processes and in the area of cryptography and lightweight cryptography, among others. Is in cryptography that the generated sequences become critically important, for example they are used in key distribution scenarios such as Kerberos [140], temporary key generation, secure key generation public key generation [171], encryption, etc. Depending on the field of application, the desired conditions for the sequences of numbers to be used will be different.

A fairly general definition of the term simulation is as follows: reproduction of a real problem in an environment (computer or other device) controlled by the experimenter.

Simulation will be appropriate:

- When you want to analyze complex models, so present today in which there is a great amount of information.
- The observation of irreversible alterations in an environment which can prevent, for example, irreversible damage caused by direct intervention in certain ecosystems.
- To guide the investigation of a phenomenon or even theoretical results by evaluating the feasibility or ineffectiveness of certain alternatives.
- To verify the importance of some variables over others in a given context.
- When it is wanted to prevent possible adverse effects before implementing a certain policy.

- When the direct experimentation in situ is costly or impossible.
- To validate certain analytical solutions and the applied methods.
- When there is no analytical procedure to address a problem or when even if there is one, it is costly to implement.

On the other hand, the simulation will not be suitable when:

- The problem has a clear and simple analytical solution, i.e. there is an analytical procedure for resolution and it can be addressed efficiently.
- The costs of the simulation exceed the possible savings that its implementation could imply.

As we mentioned beforehand, the numbers generated are intended to be random. In this respect, we can find in the literature a widespread classification of the different types of random numbers (see for example [175] or [194]): the so-called "true" random numbers, pseudo-random numbers and quasi-random numbers. True random numbers are sequences generated from phenomena with intrinsic randomness. They do not need any initial seed to be obtained and are expected to show no patterns or correlations between values. Their main drawback is that they are very costly to generate and, in many cases, time and/or hardware resources are limited. Therefore, from a certain amount of random information, it is sought to extend this information and generate very long sequences of "random" numbers in some alternative way. This is how the so-called pseudo-random numbers arise. These are generated from devices or algorithms that, given an input called seed, generate long sequences of random-looking numbers. Because of the way they are generated they can be reproducible. There is also a third category, quasi-random numbers. They are not designed to appear random, but to be uniformly distributed. One of the objectives of these numbers is to reduce and control errors in Monte Carlo simulations (for more details see [168]). In this case, statistical problems arise for the verification of the goodness of fit of the sequences obtained by hypothesis testing as the dimension increases [119]. In Table 1 principal characteristics of these types are described.

There are two main types of generators (classification based on their properties, architecture and type of implementation) that usually appear in the literature (see for example [183]):

- Physical generators (true random number generators, TRNGs): these are physical devices that use external sources to generate random numbers (hardware or natural phenomena). The most commonly used are based on electrical circuits equipped with a noise source (often a resistor or a semiconductor diode) that is amplified, sampled and compared with a reference signal to produce bit streams. These random bits are joined together to form bytes, integers or real numbers as required. One observation about the source is that it must be chosen carefully to ensure effective randomness (e.g., do not use a pulse source that has some kind of periodic pattern). All of them have the fundamental characteristic that they produce unrepeatable sequences. The advantages and disadvantages of TRNG can be seen in Table 2. The output sequences of TRNGs can be used directly

Table 1
Types of random numbers.

True random numbers	• based on physical sources • do not need an initial sequence (seed) • they are expected to have no periodic pattern or correlation between the obtained data
Pseudo-random numbers	• they are generated in a deterministic way, that is, they are constructed through generation algorithms and an initial seed • they have the appearance of being values that come from independent realizations of a $U(0,1)$ random variable (r.v.) • the way they are generated makes them reproducible
Quasi-random numbers	• are obtained through specific algorithms • the obtained sequences are distributed uniformly in the square or in the unit cube • its main disadvantage is that as the dimension increases there are no specific hypothesis tests to assess the goodness of the obtained sequences [119].

Table 2
Pros and cons of the use of a TRNG.

Pros	Once a sequence has been generated, it can never be recreated by anyone else, even if they have the same device that first created it. This feature is very convenient, for example, to protect the secrecy of communications. Once the sequence has been generated it is stored and distributed only to two correspondents, who can use it to communicate secretly using stream ciphering.
Cons	The difficulty of distributing the encryption key to the two correspondents using a secure channel. The necessity to have a physical element that is more or less bulky, expensive and difficult to produce. It is very difficult to design a device or a program that produces a bit-stream free of biases and correlations. In statistical simulation: non-reproducibility.

as random sequences or can be used as input to a pseudo-random number generator.

- Arithmetic generators (pseudo-random number generators, PRNGs): these are deterministic algorithms that are executed in computers. There are two main sub-types, linear and non-linear. The generated sequences present period (the length depends on the type of generator and on the selection of the parameters involved in the equation or equations of the generator) In this work we will focus in such generators.

Table 3 shows the principal differences between TRNGs and PRNGs.

The generation of (or pseudo-random) numbers also makes it possible to generate values of other random variables by means of certain transformations. In fact, if U is a $U(0,1)$ random variable and $F(x)$ is a distribution function, then $X = F^{-1}(U)$ is a random variable with distribution function $F(x)$, being

$$F^{-1}(u) = \inf\{x : F(x) \geq u\} \tag{1}$$

Indeed, if we denote by $F_X(x)$ to the distribution function of X, then by the

Table 3
Principal differences between TRNGs and PRNGs.

Characteristic	***TRNG***	***PRNG***
Way of generation	External source of entropy	Mathematic algorithm
Efficiency	X	$\surd$
Deterministic	X	$\surd$
Periodicity	X	$\surd$
Reproducibility	X	$\surd$

definition of distribution function: $F_X(x) = P(X \leq x)$, now as $X = F^{-1}(U)$ we have $F_X(x) = P(F^{-1}(U) \leq x) = P(U \leq F(x))$ and as U is a uniform random variable in the interval $(0, 1)$ and $F(x) \in (0, 1)$ we obtain that $F_X(x) = F(x)$ as we wanted to prove. So if we can simulate a $U(0, 1)$ random variable it is possible (at least from a theoretic point of view) to simulate any other random variable.

It is also possible to obtain sequences of identically and independently distributed (i.i.d.) random variables by using sequences of i.i.d. random bits, see for example Chapter XV of [40]. So then as before, the problem of obtaining random sequences of numbers can be reduced to the obtainment of random bits sequences.

The importance and necessity of working with the concept of randomness, random and pseudo-random numbers is evident in many areas of knowledge and has been for a long time. From a theoretical/conceptual point of view regarding random and pseudo-random numbers, we could highlight different investigations. For example, the works of [31], [93], [96], [128] and [65] on the concept of randomness, random sequences and their construction are noteworthy. Papers [77] and [195] on the generation of equidistributed or uniform numbers or papers [79] and [129] on the generation of random numbers and algorithms are also worth mentioning. Finally, as a sample, we could point out the works on the generation of pseudo-random numbers, their concept and analysis [84], [37], [198], [139], [177], [158], [43] and [16] among others.

There is also a wide variety of works showing tables of random numbers, which we will discuss in the next section, for example: [196], [56], [95], [162], [78], [189], [32], [173], [174], [169], [151], [138], [170] or [167].

There are also a wide variety of papers that analyze tables or show extraction methods, for example: [28], [24], [25], [26], [27] or [154] among others.

It is also possible to find a large number of papers related to a certain method of generating random and pseudo-random sequences, which will be discussed in the following sections of this paper with special attention to secure random numbers. Finally, it is possible to find in the literature numerous works related to statistical hypothesis tests applicable to check the properties of the sequences obtained with different procedures, in this sense a work that describes the existing statistical batteries as well as the definitions of the tests involved and the software that can be used to perform the verifications is [4].

The principal types of studies in randomness context are:

- Theoretical/conceptual perspective about random and pseudo-random numbers.
- Tables of random numbers, its analysis and extraction methods.
- Methods of generation of random and pseudo-random sequences.
- Methods for testing random numbers.
- Applications of random numbers and security

The aim of this work is to:

- Clarify the concept of randomness and the need to work with random sequences in different contexts. It will clearly distinguish what requirements are placed on the sequences generated depending on the context in which they are used, whether from a purely statistical point of view with applications in simulation and computation or from a cryptographic point of view.
- Describe in detail the most important models of PRNGs emphasizing their properties and pointing out the context in which they are the most suitable for use.
- Point out the strengths and weaknesses of the generators described to describe new designs and current lines of research.

This paper is organized as follows: Section 2 gives a historical development of random and pseudo-random number generation emphasizing the fact that arithmetic generators will be analyzed, Section 3 explains the different types of pseudo-random number generators (classical generation methods not currently used, linear and non-linear congruential methods) together with their fundamental properties. Section 4 describes the new designs of PRNGs and new lines of investigation and finally Section 5 gives the main results of this research.

2. Historical development

When working with a simulation, cryptography or system security problem (among others) it is necessary to include a source of randomness. During the first half of the 20th century, various physical procedures were used such as coin tosses, dice, experiments with cards, urns or mechanical procedures such as spinners to extract numbers (or, in general, samples) randomly or electrical circuits based on vacuum tubes with random pulses. During the second half of the 20th century a large number of works appeared proposing physical generators of random numbers. In some cases the random numbers were published in table form. One of the first papers in this line can be found in [196]. Tippet proposed a table of non-uniform random numbers, consisting of 41600 digits arranged in 10400 four digited numbers. The table proposed by Tippet has been analyzed in different works such as [211], [42], [64] and more recently in [179]. Other well-known tables in the literature are for example those proposed by Fisher and Yates in 1938 [56], analyzed in [178] or in [180] among others. In Table 4 a fragment of Table XXXIII of Random Numbers from [57] (page 134) is shown.

Table 4
Fragment of a table of Random Numbers

03	47	43	73	86	36	96	47	36	61	46	98	63	71	62	33	26	16	80	45	60	11	14	10	95
97	74	24	67	62	42	81	14	57	20	42	53	32	37	32	27	07	36	07	51	24	51	79	89	73
16	76	62	27	66	56	50	26	71	07	32	90	79	78	53	13	55	38	58	59	88	97	54	14	10
12	56	85	99	26	96	96	68	27	31	05	03	72	93	15	57	12	10	14	21	88	26	49	81	76
55	59	56	35	64	38	54	82	46	22	31	62	43	09	90	06	18	44	32	53	23	83	01	30	30

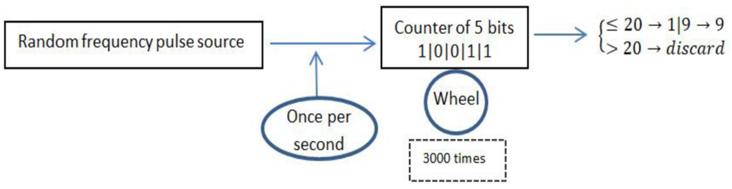

Fig 1. *Random digit generation using the RAND corporation method.*

The table proposed by Kendall and Babington Smith in 1938 [94] analyzed in [181] among others, or the table of one million random numbers of the Rand Corporation built in 1955 [32] and analyzed for example in [182]. One of the advantages of Rand Corporation's table compared to those existing up to then was precisely its large size. The digits in the table were obtained from an electrical circuit whose operation was similar to that of a roulette wheel. The circuit consisted of a random frequency pulse source providing an average of 100000 pulses per second. Once per second this train of randomly spaced pulses was connected to a 5-bit counter. The counting was done during a constant time interval of less than one second. Once the counter had reached its maximum value (31), the next pulse would take the value 0, then 1 and so on. This cyclic behavior is similar to that of a 32-number roulette wheel that spins repeatedly through all the numbers until it stops at one of them. The time during which the pulses were counted was calculated so that on average, the counter was spinning approximately 3000 times. Once the pulse counting time interval was over, the number stored in the counter ($0 \leq n \leq 31$) was converted to a decimal base. When $n > 20$ it was discarded, if $n \leq 20$, the least significant digit of the number was the random digit. This digit was stored on a perforated card. The circuits had to be adjusted several times until sequences with suitable statistical properties were obtained. In Figure 1 it is shown the random digit generation using the RAND corporation method.

Another physical generator of random numbers, in this particular case of random (independent) bits, is due to Vincent in 1970 [199]. The method he proposes consists of counting the number of randomly generated pulses during a certain time interval. If the sum is even, the value 1 is selected and if it is odd, the value 0 is selected. Pulses can be obtained in various ways such as measurement through a radioactive source detector or through a circuit that detects peaks above a certain preset threshold connected to the output of a noise source.

Vincent's generator machine satisfies that the number of pulses k, counted in a time interval s, is distributed according to a Poisson distribution with average λs:

$$P\left[N(t+s)-N(t)=k\right]=\frac{e^{\lambda s}(\lambda s)^k}{k!} \tag{2}$$

The difference between the probability that the number of pulses is even, and the probability that the number of pulses is odd, is equal to:

$$P\left[k \text{ even}\right]-P\left[k \text{ odd}\right]=\sum_{k=0}^{\infty}(-1)^k\frac{e^{\lambda s}(\lambda s)^k}{k!}=e^{-2\lambda s} \tag{3}$$

Since $e^{-2\lambda s}\simeq 10^{-13}$ for a value of s as small as 15, the probability of getting a 0 is practically the same as getting a 1, which makes Vincent's method very satisfactory.

Other noise sources that have been used for physical random number generators include a polarized Zener diode at the elbow of its characteristic curve, decomposition of radioactive sources, using radio signals received when tuning a radio receptor to a frequency at which it is not transmitting, etc.

The interest in building powerful physical random number generators that produce appropriate sequences is still strong at present. In fact, the sequences obtained with this type of generators are unpredictable and aperiodic. Given the characteristics of the sequences obtained with this type of generator, its application is of special interest in areas such as cryptography (data generation, key encryption, etc.). However, the application of physical generators in simulation studies has some disadvantages such as the necessity of high capacity memories or the slowness of the procedure. This is why other techniques have been developed to help overcome these undesirable effects. In fact, since the middle of the 20th century, and in parallel with the development of tables, other types of arithmetic (deterministic) generators were introduced, which have the advantage of being sequential and faster procedures.

3. Pseudo-random numbers generators

The most suitable and reliable method of generating random numbers is to use deterministic algorithms that have some solid mathematical basis. These algorithms produce a sequence of numbers that resembles that of a sequence of realizations of independent and identically distributed random variables according to a $U(0,1)$ random variable, although it is not really so. That is why such numbers are called pseudo-random and the algorithm that produces them is called pseudo-random number generator.

A good pseudo-random number generator should have some important properties that are described in Table 5.

Most arithmetic generators are usually very fast and require little data storage capacity. However, there are certain generators that do not satisfy the properties of uniformity and independence of the obtained sequences.

Table 5
Properties of a good PRNG.

P1	The sequence of values it provides should resemble a sequence of independent realizations of a $U(0,1)$ r.v.
P2	From the point of view of statistical simulation: - the results must be reproducible.[1] From the point of view of cryptographic applications: - the sequences must be unpredictable.[2]
P3	The sequence of generated values should have a non-repetitive cycle as long as possible.
P4	The generator - must be fast and - must occupy a small amount of internal memory.
P5	It is desirable that the generator be portable.

[1] That is, starting with the same initial conditions, the same sequence must be obtained. This would allow to debug possible failures of the model or to simulate different alternatives of the model in the same conditions;
[2] We are emphasizing the necessary security of the systems that make use of the generated sequences. See for example [186], [15] or [68]). In [19] and in [99] the problem of predicting the output of a pseudo-random number generator is considered.

There are several hypothesis tests that can be used to verify the statistical properties of randomness (autocorrelation test, test of streaks, etc.) and uniformity (for example, the Chi-Square goodness-of-fit test or the Kolmogomorov-Smirnov test). In the literature we can find different sets or groupings of these tests that are called test batteries or test suites. Among the best known are NIST SP 800-22 [176], Diehard [127], Dieharder [21], ENT [201], TestU01 [113], among others. For a detailed analysis of the most popular ones see [4].

In the case of cryptographic applications, it is also necessary that the sequence generators are able to escape from severe attacks, even if part of their initial or current state is available to an attacker. We will discuss later what additional conditions are required in this case when discussing Cryptographically Secure Pseudo-random Number Generator.

There are several well-known books in which the topic of random and pseudo-random numbers is treated, for example Chapter 3 of [97], Chapter 1 of [67], Chapter 7 of [6], Chapter 3 of [109], Chapter 7 of [101] among others as well as many papers that will be discussed in next sections.

Below we will describe the best known generators in the literature.

3.1. The middle-square method

This is due to von Neumann. It was originally presented by the author in 1949 at a conference and published in 1951 [200]. It is fundamentally only of historical interest since it is not currently used due to the weaknesses of the sequences that are obtained. These have very short periods, if the procedure is repeated a sufficient number of times the method will repeatedly generate the same number or it will go to a number previously obtained in the sequence and it will be repeated indefinitely or even degenerate to 0.

The algorithm consist of the following steps:

- take a positive integer x_0 with $2n$ digits.
- it is squared to obtain a $4n$-digit number (if it would be necessary, the number should be completed with zeros on the left).
- remove the middle digits of the resulting number, let x_1 be this number (it will be considered the *random number*).
- use that number as the seed for the next iteration.

The pseudo-random numbers are obtained by dividing the sequence by 10^{2n}.

The disadvantage of this procedure is that the generated numbers can be repeated cyclically after a short cycle and even degenerate to 0 very fast. Another important drawback is that the sequence obtained is not “random” because it can be predicted directly from the seed.

This type of generator has been discarded today and more sophisticated systems are used.

Example 1 *Let us consider $n = 2$ and $x_0 = 4122$.*

- $x_0^2 = 16990884$.
- $x_1 = 9908$, $x_1^2 = 98168464$.
- $x_2 = 1648$, $x_2^2 = 2835856$.
- $x_3 = 8358$, $x_3^2 = 69856164$.
- $x_4 = 8561$, $x_4^2 = 73290721$.
- $x_5 = 2907\ldots$

We obtain the following sequence of pseudo-random numbers: $u_0 = 0.4122$, $u_1 = 0.9908$, $u_2 = 0.1684$, $u_3 = 0.8358$, $u_4 = 0.5861$, $u_5 = 0.2907$, *...*

On the other hand, we have noted that this method can degenerate to 0 very fast (see [35]). For example, if we take $x_0 = 1009$ *the obtained sequence is:* $x_1 = 180$, $x_2 = 324$, $x_3 = 1049$, $x_4 = 1004$, $x_5 = 80$, $x_6 = 64$, $x_7 = 40$, $x_8 = 16$, $x_9 = 2$, $x_{10} = 0$.

3.2. Lehmer's method

This method consists of the following steps:

- An integer number x_0 of n digits, is taken as a seed.
- Another integer, c, of k digits is taken. Usually $k < n$.

- Calculate $x_0 \cdot c$, number of, at most, $n+k$ digits.
- The k digits on the left are separated from $x_0 \cdot c$ and the number formed by the remaining n digits is subtracted from the one formed by the k digits on the left, resulting in x_1.
- This process is repeated as many times as necessary.

The sequence of pseudo-random numbers is obtained as follows: $u_i = x_i/10^{2n}$.

This method has some drawbacks, the most noteworthy are:

- Possible appearance of negative iterants.
- Also short cycles often occur (in particular, zero is an absorbing value of this generator).

Due to these weaknesses, this type of generator is not used in practice and other methods have been designed to avoid these problems.

Example 2 *Let us consider* $x_0 = 2136$ *and* $c = 53$*, in this case* $n = 4$ *and* $k = 2$*.*

- $x_0c = 113208$, $3208 - 11 = 3197 = x_1$.
- $x_1c = 169441$, $9441 - 16 = 9425 = x_2$.
- $x_2c = 499525$, $9525 - 49 = 9476 = x_3$.
- $x_3c = 502228$, $2228 - 50 = 2178 = x_4$, *etc.*

Example 3 *Let us select the following values:* $x_0 = 2000$ *and* $c = 50$*, then* $x_0c = 100000$ *so* $x_1 = 0000 - 10 = -10 < 0$*, which is not possible. This is a simple example in which a negative iterant appears.*

3.3. Linear congruential methods

These methods are due to Lehmer (1951) [115]. The methods of generating pseudo-random numbers called congruentials are based on the mathematical concept of congruent numbers.

Two numbers a and b are said to be congruent modulo m if m divides $a - b$, this is represented by $a \equiv b \mod m$. Congruence is an equivalence relationship, indeed:

- It satisfies the symmetric property: if $a \equiv b \mod m$ then $b \equiv a \mod m$
- It satisfies the reflexive property: $a \equiv a \mod m$, for all a and $m \neq 0$.
- It satisfies the transitive property: if $a \equiv b \mod m$ and $b \equiv c \mod m$ then $a \equiv c \mod m$

Although these congruent generators have limited capacity to produce very long number sequences that can pass as independent value sequences from a $U(0,1)$ random variable, they are considered a basic element in other more efficient generators.

In this type of generators the most popular are the linear congruential methods but also non-linear methods can be found in the literature. From the cryptographic point of view linear congruency generators are easily predictable (see

[166]), it can be possible to obtain the values of the parameters in polynomial time given a sufficiently long string of generated numbers. Therefore, this type of generator is not recommended for cryptographic applications [156]. However, it can be used as an intermediate element in more complex generator designs, which may allow its use in cryptography as we will see later.

Within this type of methods we can distinguish the multiplicative congruential methods and mixed congruential methods.

3.3.1. Multiplicative congruential method

This method starts with an initial positive value x_0 called seed and two positive integer values a (multiplier) and m (modulus) with $0 < a < m$ and $x_0 < m$. The following numbers of the sequence are then obtained by means of the expression:

$$x_{n+1} \equiv ax_n \mod m, n \geq 1 \tag{4}$$

Each $x_n \in \{0, 1, ..., m-1\}$ and $u_n = x_n/m$ is called pseudo-random number that is taken as a value of a $U(0,1)$ random variable.

The expression (4) for the integers is equivalent to:

$$u_{n+1} \equiv au_n \mod 1, n \geq 1, 0 < u_n < 1 \tag{5}$$

There exist many papers in the literature that study these generators with their principal properties: see for example [45], [59] or [76] among others.

The performance of multiplicative congruential generator depends on the selections of its parameters. Generally a and m are chosen in such a way that the following conditions (conditions 3.3.1) are satisfied:

- For any initial seed, the resulting sequence will appear to be a sequence of i.i.d. values according to a $U(0,1)$ random variable (that is, the sequence passes the goodness-of-fit test when the theoretical distribution is $U(0,1)$ and the values are independent, they pass the independence test(s)).
- The sequence must have a long period.
- The values must be obtained in an efficient way from a computer point of view.

Some well-known selections of the parameters are described in Table 6.

As it can be seen in Table 6 there is a lot of research about the possible values of the parameters. Other interesting reference in this context is [107] in which several tables of parameters (depending on the size) with good performance with respect to the spectral test, are given.

Another important aspect is to know the period of the generated sequence. In regard to this point, the following result is relevant:

Theorem 1 (Knuth [97]) *The multiplicative congruential method* $x_{n+1} \equiv ax_n \mod m$ *has maximum period* $m-1$ *only if:*

- *m is a prime number.*

Table 6. *Some well know values of the parameters in a multiplicative congruential generator*

a	m	***References and comments***
23	10^8+1	Original parameters in Lehmer's 1948 model.
5^{2q+1}	2^s	[8]. In this model $q \in \mathbb{Z}^+$ and s is the number of binary digits desirable in the generated number.
65539	2^{31}	RANDU: proposed by IBM in 1960. It is not used nowadays because it fails many essential statistic tests such as spectral test.
λ	2^q or 10^q	[33]. For a q-place binary or decimal machine. In this case $\lambda \in \mathbb{Z}$ is prime to the modulus.
5	2^{35}	[11]. An algorithm is proposed for generating pseudo-random numbers with these parameters.
a_p	p	[83]. The modulus is the largest prime p within accumulator capacity and a_p is a primitive root of p.
a	2^k	[7] (also see [80] and [100]). The modulus is selected in this way because the binary base of most digital computers.
a	p_m	For example [83] or [172]. p_m is the large prime number that can be fitted to the computer word size.
a^*	m^*	[191]. m^* and a^* are the values that maximize the period and minimize the correlation of the generated sequence.
$125, 1289, 1381$	$2^{13}, 2^{11}$	[100]. Particular case of [7].
$a_1 = 16807$ $a_2 = 63036016$	$2^{31}-1$ $2^{31}-1$	For a 32-bit machine. The case with a_1 is faster and has less risk of memory overflow, it was proposed in [117] and is widely used. The case with a_2 has better statistical properties but computationally it gives more problems, especially of memory overflow. A selection that is used occasionally is the Mersenne prime $2^{61}-1$.
630360016	$2^{31}-1$	[160]. Used by some FORTRAN.
a	p	[187]. The modulus is a prime number.
69069	2^{32}	These selections are very popular and has been recommended by Marsaglia in 1972. Used in RN32 [85].
1664525	2^{32}	Used in the INMOS Transputer Development System (IMS D700D).
16807	$2^{31}-1$	[156]. Used by APL, IMSL and SIMPL/I.
742938285 40014 40692	$2^{31}-1$ 2147483563 2147483399	In [103] it is stated that these are the best selection of the parameters but this selection does not provide a portable generator. Portable selection. Portable selection.
48271	$2^{31}-1$	[157]. The authors make some comments about this selection instead of $a = 16807$
$2^{q_1} \pm 2^{q_2}$ $m - 2^{q_1} \pm 2^{q_2}$ with $q_1 > q_2$ Some particular cases: $2^{15}-2^{10}, 2^{16}-2^{21} \rightarrow$ $2^{30}-2^{19}, 2^{42}-2^{31} \rightarrow$	2^p-1 2^p-1 $2^{31}-1$ $2^{61}-1$	[208], the author states that the proposed generators with these values for m and a are very fast and pass several statistic tests. Paper [208] was revised in [112]. It is shown the weakness of the generator proposed in [208]: it is stated that if $a = \pm 2^{q_1} \pm 2^{q_2}$ the number of ones of the binary representation of x_{n-1} and x_n, (their Hamming weights) turn out to be very dependent and, therefore, this dependence extends to the bits of the generated u_n. The sequences generated according to [208] do not pass the Hamming weights test.

- *$a \not\equiv 0 \mod m$ and $a^{(m-1)/q} \not\equiv 1 \mod m$ for each prime factor q of $m-1$.*

Example 4 *Let be the following multiplicative congruential method:*

$$x_{n+1} \equiv 6x_n \mod 13, n \geq 1 \tag{6}$$

with seed $x_0 = 1$. In this case the maximum period $m-1 = 12$ is reached because $m = 13$ is a prime number and $m - 1 = 12 = 2^2 \cdot 3$ so $q = 2, 3$ and

$$a^{(m-1)/q} = \begin{cases} 6^4 = 1296 \not\equiv 1 \mod 13 \\ 6^6 = 46656 \not\equiv 1 \mod 13 \end{cases}$$

But if the method were:

$$x_{n+1} \equiv 7x_n \mod 31, n \geq 1$$

with seed $x_0 = 19$, the period is 15 and maximum period is not reached because $m - 1 = 30 = 3 \cdot 5 \cdot 2$, so $q = 2, 3, 5$ and $a^{(m-1)/2} = 7^{15} \equiv 1 \mod 31$.

In [97] another result related with the period of a multiplicative congruential generator for the case of $m = 2^\beta$ for any positive integer β, is explained:

Theorem 2 (Knuth [97]) *The multiplicative congruential method $x_{n+1} \equiv ax_n \mod 2^\beta$ has maximum period $m/4$ attained if and only if x_0 is odd and $a \equiv 3 \mod 8$ or $a \equiv 5 \mod 8$.*

In practice, it is recommended a period of at least 10^9, so at least the modulus that would need to be fixed would be at least 10^9. However, this amount is still not adequate, as with the speed of computers, this period can be reached very quickly.

Some references about the period of these types of generators are [17] where the properties of the period in a multiplicative congruential generator $a^\delta \equiv 1 \mod n$ are studied or [23] where the properties about the period of the generator $a^{\theta(m)} \equiv 1 \mod m$ with a and m relatively primes are studied.

3.3.2. Mixed congruential method

The expression in this case is more general:

$$x_{n+1} \equiv ax_n + b \mod m \tag{7}$$

where $a, m \in \mathbb{Z}^+$ with $0 < a < m, 0 \leq b < m$ and $x_0 < m$. Here b is called increment.

Note that as in the previous situation (multiplicative congruential method) the numbers that are obtained with expression (7) are completely determined by m, a, b, and x_0. In fact, it is verified that:

$$x_i \equiv \left[a^i x_0 + \frac{(a^i - 1) + b}{a - 1}\right] \mod m \tag{8}$$

TABLE 7
Some well know values of the parameter in a mixed congruential generator

a	m	b	***References and comments***
λ	p	μ	[72], section 2 selection of a, b and m in order to get maximum period.
2^k+1	2^l	b	[161]. Here $k \in \mathbb{Z}$, $l \in \mathbb{N}$.
a	m	b	[3], several cases for different selection of the parameters are studied.
a	m	b	[82], selection of the parameters in case of being working in a binary machine.
5^{15}	2^{35}	1	[34].
314159269	2^{31}	453805245	[98]. Has a maximum period.
25214903917	2^{48}	11	drand48 (48-bit PRNG of C language under UNIX).
1103515245	2^{31}	12345	Unix rand generator
69069	2^{32}	1	Vax generator MTH$RANDOM

but with appropriate selection of these parameters the sequence can be considered to be formed by realizations of a $U(0,1)$ random variable (that is, the generated sequence passes the goodness-of-fit test when the theoretical distribution is $U(0,1)$).

As in the previous case, it is important to know the period of the generated sequence. In this regard, we highlight the following result:

Theorem 3 (Hull, T. E. and Dobell, A. R. [81]) *The simultaneous verification of the following three conditions is necessary and sufficient for a congruential generator to have maximum period $h = m$:*

- $\gcd(b, m) = 1$ *(where* $\gcd(b, m)$ *is the greatest common divisor of* b *and* m*).*
- $a - 1$ *is a multiple of each prime number that divides* m*.*
- *If* m *is multiple of 4, then* $a - 1$ *is also multiple of 4.*

Example 5 *Let us consider the following mixed congruential method:*

$$x_{n+1} \equiv 5x_n + 7 \mod 200 \tag{9}$$

with seed $x_0 = 3$*. This generator has period* 8 *but maximum period is not reached because* $m = 200 = 2^3 \cdot 5^2$ *so* $q = 2, 5$ *and* $a - 1 = 4$ *is not a multiple of 5.*

It has been proved that a good selection for b can be $b \mod 8 \equiv 5$ if you work in binary system or $b \mod 200 \equiv 21$ if you work in decimal system. More specifically, b must be an odd integer and relative prime with m.

As for the selection of the seed, this is irrelevant (in general, see Theorem 4) in the sense that the value of this parameter does not seem to affect the statistical properties of the generated sequences.

There are several works that analyze parameters in a mixed (linear) congruential generator. Some of the most well known values are given in Table 7.

Others well known references that analyzes this issue are the following. In [142] a method is given to search systematically for multipliers which are optimal with respect to statistical independence of pairs. In [18] a systematic search method is used in order to obtain the values of the multipliers that are optimal with respect to statistical independence of pairs of successive pseudo-random numbers. An interesting paper involving multiplicative congruential generators is [58] where a method of an exhaustive search to find optimal full period multipliers for the multiplicative congruential generator with $m = 2^{31} - 1$ is described. In this work, the concept of optimal multiplier is related to the concept of optimal distance. In [39] a method based in Stern sequences [190] and continued fractions is described to find optimal multipliers for linear congruential pseudo-random number generators. In [22] two systematic search methods are employed to find multipliers which are optimal with respect to an upper bound for the discrepancy of pairs of successive pseudo-random numbers.

Theorem 4 resumes some of the principal properties that should be verified in order to get periods as long as possible.

Theorem 4 (Law [101]) *Given a linear congruential generator $x_{n+1} \equiv ax_n + b \mod m$:*

- *If $m = 2^c$ and $b \neq 0$ the longest possible period is m, which is achieved whenever b is relatively prime to m (i.e. the greatest common factor of b and m is 1) and $a = 1 + 4k$, $k \in \mathbb{Z}$.*
- *If $m = 2^c$ and $b = 0$ the longest possible period is $m/4 = 2^{c-2}$, which is achieved if the seed x_0 is odd and if $a = 3 + 8k$ or $a = 5 + 8k$, for some $k = 0, 1, 2, \dots$*
- *For m a prime number and $b = 0$, the longest period is $m - 1$, which is achieved whenever a has the property that the smallest integer k such that $a^k - 1$ is divisible by m is $k = m - 1$.*

As we have commented before, it is very important to work with generators that have long periods. One possibility to increase the period of a congruential linear generator is by shuffling. In this context in [121] it is suggested to shuffle the output of the congruential generator using another simpler generator. In [9] an algorithm to Shuffling of Uniform Deviates is presented. The principal steps of the algorithm are:

- Set k as the length of the table T. Fill the table with with $x_1, \dots, x_k$. Put $i = 1$. Generate x_{k+i}. Set $y_i = x_{k+i}$.
- Generate j from y_i.
- Set $i = i + 1$ and put $y_i = T(j)$.
- Generate x_{k+i}. Refresh $T(j)$ with x_{k+i}.

An inherent property of congruential generators is that they produce a lattice structure as can be seen in Figure 2.

In this grid structure a series of lines can be identified where all the pairs in the series are located. Depending on the distance between these lines, the pairs are distributed more or less evenly in the plane. Generally, the greater the

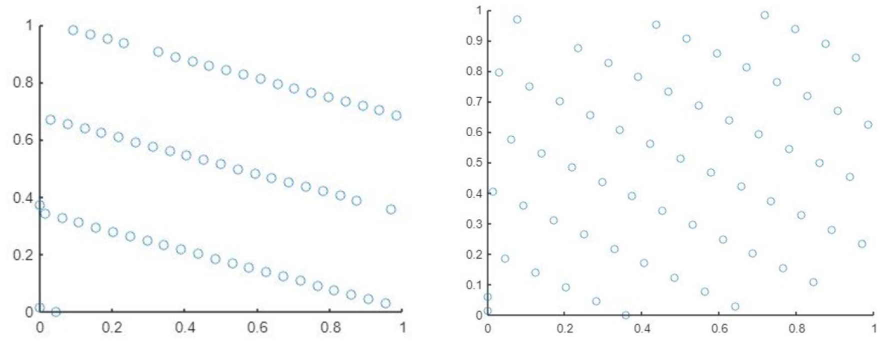
(a) $a = 21, b = 1, m = 64$ (b) $a = 25, b = 1, m = 64$

FIG 2. *Representation of* (u_i, u_{i+1}) *for two examples of mixed linear congruential generators.*

distance, the worse the generator. This distance is determined by the parameter a.

This structure can be detected by the spectral test [34] or by the lattice test [126]. Interesting references which studies the lattice structure in the case of a multiplicative congruential generator are [125] or [12]. Some interesting papers where the correlation of the generated values of a mixed congruential generator is studied are [33], [72] or [41].

With the previously mentioned IBM generator, generator all the triplets of consecutive numbers of the series $(5 - 10^{10})$ fall in only 15 planes. Marsaglia demonstrated in 1968 that the maximum number of parallel hyperplanes that can produce a linear multiplicative congruential generator is $(n!m)^{1/n}$, where n is the number of consecutive numbers of the subsections considered. Note that the number of hyperplanes decreases rapidly as the dimension of the space increases n.

With a shuffling procedure as in [121] is described, it is possible to break up the lattice structure.

There are several articles in the literature that study the fundamental properties of linear congruential generators, for example [141], [142] or [143].

If different random number sequences are available, each must be used for a random parameter [102]. In this case, the length of the string must be taken into account to avoid overlaps. In [102] different seeds for strings of the mentioned generator separated by 100000 pseudo-random numbers can be found.

Although congruential generators are the most used in practice, other types of generators have been developed with the objective of obtaining longer periods and better statistical properties. Often, however, a congruential generator with properly chosen parameters can work as well as more complicated alternatives.

3.3.3. Multiple recursive generators

Congruential generators can be generalized to higher order linear recursions, considering the following relationship:

$$x_n \equiv (a_1 x_{n-1} + \ldots + a_k x_{n-k}) \mod m \tag{10}$$

where $k, m \in \mathbb{Z}^+$ and $a_j \in \mathbb{Z}_m$, then the sequence of pseudo-random numbers is taken as $u_n = x_n/m$. (See [104] and [110]).

The study of this type of generator is associated with the study of the characteristic polynomial:

$$P(z) = z^k - a_1 z^{k-1} - \ldots - a_k \tag{11}$$

about the finite field $\mathbb{Z}_m$. When m is prime and the polynomial is primitive over $\mathbb{Z}_m$, the period of the generator is $m^k - 1$ (maximum possible period in this class of generators) (See [97]).

It is possible to generalize the previous expression a little more (10) by adding an addend $b \in \mathbb{Z}_m$ with the result:

$$x_n \equiv (a_1 x_{n-1} + \ldots + a_k x_{n-k} + b) \mod m \tag{12}$$

In [110] the authors study multiple recursive generators to find those with good properties related to structure and computational efficiency, in their work show various possibilities depending on the parameter values.

In [87], different multiple recursive generators of orders one, two and three that possess good properties of randomness and homogeneity are studied. The authors consider different values of the generator parameters and apply statistical tests such as serial autocorrelation, runs test or chi-square test among others to verify these properties.

The computational efficiency of multiple recursive generators can be improved by choosing some $a_j = 0, 1, -1$. In [38] it is proposed a portable Fast Multiple Recursive Generator (FMRG) and it is claimed that its properties are better than those of classical linear congruential generators.

A particular case of this type of generator is that in which $m = 2$ and is based on a sequence of zeros and ones that are generated according to the recursive formula (Linear Feedback Shift Register Generator-LFSR-):

$$x_n \equiv (a_1 x_{n-1} + \ldots + a_k x_{n-k}) \mod 2 \tag{13}$$

where $a_i \in \{0, 1\}$ for all $i = 1, ..., k$, in this case, x_i represent bits that constitute the binary representation of an integer. This generator was introduced by Tausworthe in 1965 [193]. Its properties and characteristics have been widely studied in the literature (see for example [205], [55], [159], [197] or [69] among others). In relation to its use, this is very remarkable in areas such as communications or in cryptography, even though they are not cryptographically secure in principle. To overcome this problem, they are used as building blocks in more sophisticated constructions of PRNGs. The most common techniques in this context are:

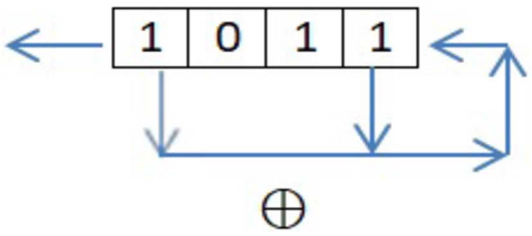

FIG 3. *Diagram of the LFSR in Example 6.*

- Combine the output of several LFSRs by using a nonlinear function.
- Use the output of one LFSR (or a combination of several LFSRs) to clock one (or a combination of) other LFSR(s).

However, the design of such a combination has to be done with great care and must be analyzed with the cryptanalytic techniques.

Example 6 *Let us consider the following generator:*

$$x_n \equiv x_{n-4} + x_{n-1} \mod 2$$

with $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$, $a_1 = a_4 = 1$ *and* $a_2 = a_3 = 0$.

The first 12 elements of the sequence are: $x_5 = 0$, $x_6 = 0$, $x_7 = 1$, $x_8 = 0$, $x_9 = 0$, $x_{10} = 0$, $x_{11} = 1$, $x_{12} = 1$, $x_{13} = 1$, $x_{14} = 1$, $x_{15} = 0$, *etc. These values correspond with the following scheme:*

For best results in terms of computational efficiency it is recommended taking most of the $a_i = 0$ in (13).

A particular case of (13) and that appears frequently in the literature is:

$$x_n \equiv (x_{n-k+q} + x_{n-k}) \mod 2 \tag{14}$$

resulting from a trinomial.

A generalization of (13) was made by Lewis and Paine in 1973 [118] and they called it as Generalized Feedback Shift Register pseudo-random number Generator (GFSR). The authors claim that the GFSR is capable of producing multidimensional pseudo-random numbers, of arbitrarily long period and at higher speed than other pseudo-random generators such as Lehmner's or Kendall's. In their work, they also carry out a detailed study of the generator paying special attention to the period and statistical properties on correlations, among others.

Other authors have studied this type of generator such as [63], [131], [132] or [133]. In this last paper ([133]), the pseudo-random number generator known as the "Mersenne twister" is defined, and is known for its quality. Its name is because that the length of the period of the generated sequence corresponds to a Mersenne prime number. There are at least two variants of this algorithm, differing only in the size of Mersenne primes used. The most recent and most widely used is the Mersenne Twister MT19937, with a word size of 32-bit. Its period is $2^{19937}-1$. There is another variant with 64-bit words, the MT19937-64, which generates another sequence. Two interesting references which study such generators are [155] and [111].

Mersenne twister in its primitive form is not recommended for cryptographic applications (it is based on a linear recursion and any sequence of pseudo-random numbers generated by a linear recursion is insecure since from a sufficiently long sub-sequence of the outputs, the rest of the outputs can be predicted), but for statistical simulations it is.

3.3.4. Lagged Fibonacci generator

The lagged Fibonacci generator is a particular case of multiple recursive generator. It is defined as:

$$x_i \equiv (x_{i-j} + x_{i-k}) \mod m \tag{15}$$

In the particular case in which $j = 1$, $k = 2$ the generator is known as the Old Fibonacci generator. This generator tends to have a period greater than m, but is unacceptable from a statistical point of view: the following arrangement of three consecutive output values $u_{i-2} < u_i < u_{i-1}$ can never be produced whereas such a structure should occur with probability $1/6$ in the case of a "perfect" random number generator. [20]

However, under certain selection of the parameters generator (15) can perform well. For example, if m is a prime number and $k > j > 0$, then the length of the cycle can be as long as $m^k - 1$ (see [5]) and if $m = 2^p$, the maximum period can be $(2^k - 1)2^{p-1}$. An example of this generator is:

$$x_i \equiv (x_{i-37} + x_{i-100}) \mod 2^{30} \tag{16}$$

with an approximate period 2^{129}.

Some references where this type of generator is studied are for example: [73] where the additive generators are studied with its properties, [202] and [122], where the properties concerning the period of the Fibonacci generator are studied, [71] where it is studied the case $x_i \equiv x_{i-1} + x_{i-n} \mod 1$ with $j > n$ and its equivalent statement for binary machines $x_i \equiv x_{i-1} + x_{i-n} \mod 2^r$, $j > n$, $0 \leq x_i < 2^r$ being r the number of bits used to encode each fixed point number and [137] where the additive congruential method is studied for the case $j = 2$, $k = 3$ and m a prime number.

A common way in which this generator is generalized is using some binary operator $\circ$:

$$x_i \equiv (x_{i-j} \circ x_{i-k}) \tag{17}$$

with $0 \leq x_i < m$ and $0 < j < k < i$.

A particular and very used operator is the XOR ($\oplus$) operator:

$$t_i = \begin{cases} 0, & \text{if } t_{i-j} = t_{i-k} \\ 1, & \text{if } t_{i-j} \neq t_{i-k} \end{cases} \tag{18}$$

and it is denoted by:

$$t_i \equiv (t_{i-j} \oplus t_{i-k}) \tag{19}$$

Then a sequence of binary integers x_1, x_2,... is defined as follows:

$$\begin{cases} x_1 = t_1 t_2 ... t_l \\ x_i = t_{(i-1)l+1} t_{(i-1)l+2} ... t_{il}, i = 2, 3, ... \end{cases} \tag{20}$$

That is, l consecutive t_i are stringed to form x_i as a number in base 2. The recurrence for the x_i is the same that the recurrence for the t_i, that is:

$$x_i \equiv (x_{i-j} \oplus x_{i-k}) \tag{21}$$

where the $\oplus$ operation is performed bitwise. Then, the pseudo-random number u_i which is taken as a value of a $U(0, 1)$ random variable is:

$$u_i = \frac{x_i}{2^l}, i = 1, 2, ... \tag{22}$$

3.3.5. Combined multiple recursive generators

Linear congruential generators with modulus around 2^{31} may be insufficiently long for certain applications. In fact, this length can be used up in a few minutes on a PC. However, congruential algorithms can be combined to increase the period of the generation cycle.

The principal idea is to combine two or more multiplicative congruential generators in order to obtain longer sequences with "good" statistical properties.

The disadvantage of using a combined generator is the associated computational cost, which is higher than that required for a simple congruential generator.

In [103] it is shown the following result that suggests how this can be done:

Lemma 1 (L'ecuyer [103]) *Let W_i, $i = 1, ..., l$ be l independent discrete random variables where W_1 is a discrete uniform random variable between 0 and $d - 1$, $d \in \mathbb{Z}^+$. Then:*

$$W = \sum_{i=1}^{l} W_i \mod d \tag{23}$$

follows a discrete uniform law between 0 and $d - 1$.

This lemma can be applied to form combined generators in this way: let $X_{i,j}$, $j = 1, ..., l$ the i-th output from l different multiplicative congruential generators, where the j-th generator has prime modulus m_j and period $m_j - 1$. Then the j-th generator produces integers $x_{i,j}$ that can be considered approximately as realizations of a discrete uniform random variable $X_{i,j}$, on the integers from 1 to $m_j - 1$ and $W_{i,j} = X_{i,j} - 1$ is approximately uniformly distributed on the integers from 0 to $m_j - 2$. In [103] the following combined generator is proposed:

$$X_i \equiv \sum_{j=1}^{l} (-1)^{j-1} X_{i,j} \mod m_1 - 1 \tag{24}$$

with

$$U_i = \begin{cases} X_i/m_1 & \text{if } X_i > 0 \\ (m_1-1)/m_1 & \text{if } X_i = 0 \end{cases} \tag{25}$$

The maximum period for this generator is:

$$P = \frac{\prod_{j=1}^{l}(m_j-1)}{2^{k-1}} \tag{26}$$

In [103] the properties of generator in (24) are studied paying attention to the period and the possible values that can be selected for the multipliers and modulus in order to achieve periods as long as possible verifying the desired statistical properties. In addition, the proposed generators are tested with Knuth's statistical test battery.

In [121] it is discussed the testing methods for generating pseudo-random numbers in case of using linear (multiplicative and mixed) conguential methods and other methods, in particular the authors proposed the following sequence $\{x_i\}$ built in the following way:

$$u_{i+1} \equiv (2^{17}+3)u_i \mod 2^{35} \tag{27}$$

$$v_{i+1} \equiv (2^{7}+1)v_i + 1 \mod 2^{35} \tag{28}$$

and $u_0 = 1$, $v_0 = 0$. A table of 128 values of the sequence of $\{u_i\}$ is generated and then to generate a value of x_i it is used the first seven bits of v_i as an index to get x_i from the table. Another interesting reference in which a method of generating pseudo-random uniform numbers based on the combination of two congruential generators is described is [204].

In [10] the following generator is analyzed:

$$u_{i+1} \equiv ku_i \mod p \tag{29}$$

$$v_{i+1} \equiv lv_i + 1 \mod q \tag{30}$$

where k, l, p, q, u_i and v_i are positive integers. The sequence $\{x_i\}$ is generated by combining the sequences $\{u_i\}$ and $\{v_i\}$. In this paper, also several statistic tests are applied to the resulting $\{x_i\}$ sequence.

Other well-known combined multiple generator is due to Wichmann and Hill ([206], [207]). It has a period of order 10^{12} and is defined as:

$$x_i \equiv 171x_{i-1} \mod 30269 \tag{31}$$

$$y_i \equiv 172y_{i-1} \mod 30307 \tag{32}$$

$$z_i \equiv 170z_{i-1} \mod 30323 \tag{33}$$

And take:

$$u_i = \left(\frac{x_i}{30269} + \frac{y_i}{30307} + \frac{z_i}{30323}\right) - \left\lfloor \frac{x_i}{30269} + \frac{y_i}{30307} + \frac{z_i}{30323} \right\rfloor \tag{34}$$

This combination is based on the following results:

- If $U_1, \ldots U_k$ are independent an identically distributed $U(0,1)$ random variables, then the fractional part of $U_1 + \ldots + U_k$ also follows a $U(0,1)$ distribution and $U_1 + \ldots + U_k - \lfloor U_1 + \ldots + U_k \rfloor \sim U(0,1)$.
- If $u_1, \ldots, u_k$ are generated by congruential methods with periods $c_1, \ldots, c_k$, respectively, then the fractional part of $u_1 + \ldots + u_k$ has a cycle of period least common multiple of $c_1, \ldots, c_k$.

Generator (31) has a long period (but not as long as stated in [207]) it is portable and efficient although the code can present some numerical difficulties in some particular computer architectures (see [134]).

In [105] and in [108] another design combined generator is presented. I different multiple recursive generators of the form $z_{i,j} \equiv a_1 z_{i-1,j} + a_2 z_{i-2,j} + \ldots + a_q z_{i-q,j} \mod m_i$, $i = 1, ..., I$ are considered. Given $\delta_1, ..., \delta_I$ specific constants it is built:

$$y_j \equiv \delta_1 z_{1,j} + ... + \delta_I z_{I,j} \mod m_1 \tag{35}$$

And taking $u_j = y_j / m_1$.

The values of this type of generator must be chosen very carefully. It is possible to obtain long periods and sequences verifying good statistical properties. An interesting study concerning this last point can be seen in [108].

In [114] it is studied the following combined generator:

$$x_{1,n} \equiv (a_1 x_{1,n-2} - b_1 x_{1,n-3}) \mod m_1 \tag{36}$$

$$x_{2,n} \equiv (a_2 x_{2,n-1} - b_2 x_{2,n-3}) \mod m_2 \tag{37}$$

$$z_n \equiv (x_{1,n} - x_{2,n}) \mod 4294967087 \tag{38}$$

and

$$u_n = \begin{cases} z_n/4294967088 & \text{if } z_n > 0 \\ 4294967087/4294967088 & \text{if } z_n = 0 \end{cases} \tag{39}$$

being $a_1 = 1403580$, $b_1 = 810728$, $m_1 = 2^{32} - 209$, $a_2 = 527612$, $b_2 = 1370589$, $m_2 = 2^{32} - 22853$.

These types of generators allow the simultaneous acquisition of multiple strings, each of which can be divided into many consecutive long sub-strings. The objective is to obtain sequences whose statistical properties are better than those obtained from each of the independent generators that compose it.

The length of this generator is far superior to those of the previous ones:

$$l = \frac{(m_1^3 - 1)(m_2^3 - 1)}{2} \tag{40}$$

To generate different strings and sub-strings, two positive integers v and w are chosen, and $z = v + w$ is defined. Then, the cycle l is divided into contiguous strings of length $Z = 2^z$ and each in turn is divided into $V = 2^v$ sub-strings of length $W = 2^w$. Suitable values are $v = 51$ and $w = 76$, so $W = 2^{76}$ and $Z = 2^{127}$. For this particular generator the following values can be used as initial default seeds $(x_{1,n-3}, x_{1,n-2}, x_{1,n-1}, x_{2,n-3}, x_{2,n-2}, x_{2,n1}) =$

$(12345, 12345, 12345, 12345, 12345, 12345)$. Other parameters suitable for this type of generators can be found in [108].

Specific attention should be paid to the generation of random numbers to be used in parallel calculations [62].

3.3.6. Matrix congruential generators

It is possible to design other types of generators that produce better sequences than those obtained by classical congruential generators by means of a matrix design. Thus, the so-called Matrix Congruential Generators are developed. These are defined as:

$$x_n \equiv (Ax_{n-1} + B) \mod m \tag{41}$$

where x_n is a d-dimensional vector and A and B are matrices $d \times d$. The elements of the vectors and the arrays are integers $z \in \{1, \dots, m-1\}$.

The case $B = 0$, $x_0 \neq 0$ and m a prime number was studied by [61], [74] and [144].

The lattice structure of the sequences generated by this type of generator is studied in [2].

Of particular interest is the study in [38] that looks at this type of generator and proposes a fast generation method that the authors call Fast Matrix Congruential Generator (FMCG). It is claimed that FMCGs produce better sequences than classical linear congruential generators and are easily implemented and computationally efficient.

It is possible to generalize the expression of a multiple recursive generator (10) for higher dimensions and so it is defined the multiple recursive matrix random number generator as:

$$x_n \equiv (A_1x_{n-1} + ... + A_kx_{n-k}) \mod m \tag{42}$$

where x_n is a d-dimensional vector and A_i, $i = 1, ..., k$ are matrices $d \times d$.

Some interesting references about the matrix congruential generators are [147] or [148].

3.4. Non-linear congruential generators

The linear congruential generators have some weaknesses related to regularity, which is why other non-linear congruential methods have been developed to try to overcome these deficiencies. While it is true that their generation requires more computational effort than linear models, with technological progress and the existence of increasingly better, more efficient computers with more memory, this is not a major problem.

The generic expression of a non-lineal congruential generator is:

$$x_n \equiv g(x_{n-1}, x_{n-2}, \dots) \mod m \tag{43}$$

where g is a non-linear deterministic function. As in the case of linear congruential generators, the sequences obtained with this formula, are integers between 0 and $m-1$. From these, the sequence of values in $(0,1)$ is obtained taking $u_n = x_n/m$.

Next, we will discuss some of the most well-known ones.

3.4.1. Inversive congruential generators

They were introduced in [53]) and three fundamental types can be distinguished according to whether the modulus is a prime number, a power of 2 or an odd prime number (the latter introduced in [51]).

They are defined as:

$$x_i \equiv (a x_{i-1}^{-} + b) \mod m \qquad (44)$$

with $0 \leq x_i < m$, and

$$x^{-} = \begin{cases} \text{multiplicative inverse of } x \text{ modulo } m, \text{ if it exists} \\ 0 \text{ in other case} \end{cases}$$

The sequence of values adjusted in the interval $(0,1)$ are obtained by the quotient: x_i/m.

As in the case of linear congruential generators, it is important to detect the length of the sequences that can be obtained in this type of generator. In fact, in [53] the authors describe an efficient method for the calculation of the period in these generators when the module is a prime number p. In particular, they give the following result:

Theorem 5 (Eichenauer, J. and Lehn, J. [53]) *Let a and b be the coefficients of the inversive non-linear congruential method choosing in order to verify that $z^2 - bz - a$ were a primitive polynomial over $\mathbb{Z}_p$. Then, the sequence x_i/p has maximum period and its value is p.*

If the inversive congruential method has a maximum period, then the uniformity test for equidistribution is passed in $[0,1)$ as shown in [142]. This aspect related to the uniformity properties of the sequences obtained using this type of generators has also been studied in [53] and [52]. In relation to the independence of the obtained values, a detailed analysis is made in [145] and [146].

In the case that the modulus m were a power of 2, the period of the generated sequence has also been studied. In this respect in [54] the keys are given in the following theorem:

Theorem 6 (Eichenauer-Herrmann, J. et. al. [54]) *An inversive congruential method with modulus $m = 2^k$, $k \geq 3$, will give sequences with maximum period $m/2$ if and only if:*

- $a \equiv 1 \mod 4$ *and*
- $b \equiv 2 \mod 4$

In relation to the independence property tested with the serial test, some results detailed in [146] and [50] can be found in which, (among others), some disadvantages are shown that this type of generators with a power of two modulus have in detriment of those with a prime modulus. Another disadvantage is pointed out in [49] and refers to the existence of regular structures within the generated points.

As it was explained above, the last type of inversive congruential generator, those with modulus is a power of a prime number, was introduced in 1990. For this type it also has been studied the period pointing out the theorem of [47].

Theorem 7 (Eichenauer-Herrmann, J. [47]) *Let be considered an inversive congruential generator with modulus* $m = p^k$, $p \geq 3$ *and* $k \geq 2$. *Let* $\lambda \geq 2$ *and* η *be an integer such that* $y_\lambda \equiv y_0 + p\eta \mod p^2$. *For* α, β *integers let us consider a sequence of integers* $(y'_n)_{n\geq 0}$ *such that* $y'_0 \equiv y_0 \mod p$ *and* $y'_{n+1} \equiv (a + p\alpha)y_n^- + b + p\beta \mod m$, $n \geq 0$. *Then the inversive congruential sequence* $(y'_n)_{n\geq 0}$ *has maximal period* l *if and only if*

$$\lambda(y_0^2 - by_0 - a)(b\alpha - 2a\beta) + \eta a(4a + b^2) \not\equiv 0 \mod p$$

The properties of uniformity and independence can be studied in [46].

An interesting survey about this type of generators is [48].

With regard to the use of this type of generator, it should be noted that it is not very widespread. In the case of inversive congruential generators, there are some means of accelerating the calculations (see [70]). The randomness of the sequences is better than in the case of linear congruential generators, although their performance in passing the tests based on spacings was not as good, as stated by L'Ecuyer [106], and therefore their use is not recommended in most tests.

3.4.2. Knuth's non-linear congruential generator

It was proposed by Knuth in 1998 [97] it is defined as:

$$x_i \equiv (dx_{i-1}^2 + ax_{i-1} + b) \mod m \tag{45}$$

with $0 \leq x_i < m$.

The method can be generalized to higher order polynomials although in practice, there seems to be no advantage in doing so. The particular case of $a = 0$ and $b = 0$ with m the product of two different large prime numbers has been studied in [13] and [14], in these articles the generator is defined and its main characteristics are shown. Blum, Blum, and Shub demonstrated the following result about the unpredictability of their generator: if $m = p_1p_2$, where p_1 and p_2 are primes $p_1 \neq p_2$ that verify $p_i \equiv 3 \mod 4$, $i = 1, 2$, then the sequence provided by this generator is not predictable in polynomial time without knowing the values of p_i, $i = 1, 2$. This result has important applications in the field of cryptography because of the computational difficulty of distinguishing quadratic residues $\mod m$ non-quadratic residues modulo $\mod m$. On the contrary, for statistical purposes, this generator is not recommended because the non-linearity of the generator cannot be performed in an efficient way (from the perspective of computational efficiency). An interesting reference that studies the properties of this generator is [36].

Another particular case of this generator is when the parameters values are: $d = a = 1$, $c = 0$ and $m = 2^k, k \in \mathbb{Z}^+$. In this case the generator turns out to be very similar to the one defined by the middle-square method although it has better statistical properties. The period of this quadratic generator is at most m.

3.4.3. More general formulations

In addition to the above-mentioned generators, formulations in terms of more general g functions can be found in the literature. It can be considered general non-linear functions or mixtures in which linear and inversive addends are considered. With respect to the first case, the formulation would be:

$$x_i \equiv g(x_{i-1}) \mod m \tag{46}$$

This type of generators with non-linear functions g and with modulus m being a prime number was studied in detail in [52].

Another example is the case in which linear and inversive addends are combined, this type was studied in [89] and can be defined as:

$$x_i \equiv (a\bar{x}_{i-1} + cx_{i-1} + b) \mod m \tag{47}$$

with $0 \leq x_i < m$, $a, b, c \in Z_m$ and $m = 2^k$, $k \geq 3$. The sequence of pseudo-random numbers is obtained by x_i/m.

In [89] the authors study the period of the obtained sequences and formulate the next theorem:

Theorem 8 (Kato, T. et. al. [89]) *Let us consider $m = 2^k, k \geq 3$ and the generator defined in (47), then the sequence x_i/m es purely periodic with period $m/2$ if and only if $a + c \equiv 1 \mod 4$ and $b \equiv 2 \mod 4$.*

An important case of this type of non-linear generators is the Non-linear (feedback) Shift Register (NLFSR) generator. The equation of this model is:

$$x_{i+1} = f(x_i, \ldots, x_{i-L+1}) \tag{48}$$

where f can be any non-linear function in L variables. For computational purposes the most recommended case is the binary case, in which each cell contains a bit, and f is a Boolean function.

It is known that any binary sequence generated by a NLFSR has a period that can be at most equal to 2^L (sequences that reached this period are known as De Brujin sequences), and that any periodic sequence can be produced by such a register.

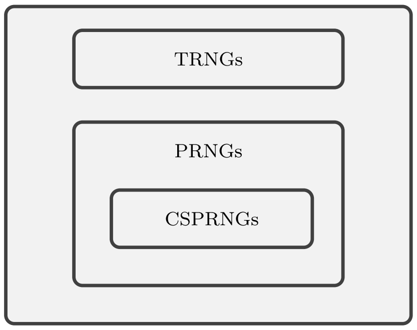

Fig 4. *Types of generators*

Example 7 *Let us consider the NLFSR defined by:*

$$x_{i+1} = x_i \wedge x_{i-1} \oplus x_i \oplus x_{i-2} \tag{49}$$

where $\wedge$ represents the AND operator and $\oplus$ the XOR operator, and the initial conditions $x_0 = 0$, $x_1 = 1$ and $x_2 = 0$. In this case the obtained sequence is $x_3 = 0$, $x_4 = 1$, $x_5 = 0$, $x_6 = 0$, $x_7 = 1$, $x_8 = 1$, $x_9 = 0$, $x_{10} = 1$, $x_{11} = 0$, $x_{12} = 0$, $x_{13} = 1$, $x_{14} = 1$, etc.

4. New lines of research: other methods for generating pseudo-random numbers

As before-mentioned, one of the fundamental areas in which the generation of pseudo-random numbers is crucial is cryptography and lightweight cryptography. The latter responds to security requirements in environments with limited hardware and software resources, for example the Internet of Things (IoT). Applications in key security and encryption [184], [188] are noteworthy.

From a cryptographic point of view, it is essential that the used generators are secure. Recalling the properties that a good PRNG must fulfill (see Table 5), the unpredictability property must be verified. This implies that knowledge of any subsequence of a generated sequence does not imply that the complete sequence can be calculated or estimated, and that knowledge of some internal state does not imply that the preceding or subsequent numbers can be calculated. In this sense, it is possible to find in the literature the definition of Cryptographically Secure Pseudo-random Number Generator (CSPRNGs). CSPRNGs are a special type of PRNG with the property of unpredictability. This means that given n consecutive bits of the key, there is no algorithm in polynomial time that can predict the next bit with a probability of success greater than 50 %.

The requirements of an ordinary CSPRNG also satisfy those of a PRNG, but not vice versa. CSPRNGs must also satisfy that: (1) their statistical properties are good (pass statistical randomness tests) and (2) that they can come out successful under severe attacks, even if part of their initial state or current state is available to an attacker.

Every CSPRNG:

- Must satisfy the next-bit test: given the first k bits of a random sequence, there is no polynomial time algorithm that can predict the $(k + 1)$-th bit with a probability of success greater than 0.5 [91]. A generator that passes the next bit test will pass any other statistical test of randomness in polynomial time [210].
- Must support state compromise extensions: if some or all of its state information is revealed, it must be impossible to reconstruct a sequence of random numbers generated prior to state attainment. Moreover, if there is entropy input while running, it should be impossible to use knowledge of the input state to predict future state conditions.

As described in the previous sections, certain PRNGs are not suitable for cryptographic uses. Although there are designs that are theoretically proved to be secure, see [43], [44] or [1] among others. However, most of the generators implemented in operating systems and cryptographic libraries are not based on the main security models of theoretical PRNGs and present weaknesses that make them vulnerable against known attacks (as can be the case of Linux OS PRNGs, OpenSSL, Android, OpenJDK, Bouncycastle and IBM). For example, certain PRNGs are vulnerable to machine learning attacks, in [86] it is described how long-short term memory (LSTM) turns out to be efficient to decrypt a normal PRNG.

The main attacks that PRNGs can suffer from are (see [92]):

- Direct cryptanalytic attack: it is carried out when the sequence generated by the PRNG is not completely indistinguishable from a truly random sequence. In this case, a hacker could deduce what type of PRNG has been used and maybe what is the key that governs it. This attack is only feasible when a certain number of the generated sequence numbers can be observed directly or if the sequence can be found out indirectly. If the PRNG were used exclusively to generate keys for other secure encryption algorithms, such as TDEA or AES, it would not be possible to deduce these keys so it would be impossible to attack the PRNG that generated them.
- Entry-based attack: this attack occurs when an attacker is able to use knowledge about the PRNG input sequence to cryptanalyze it. This attack can be implemented in several ways: (i) known input attack, (ii) chosen input attack and (iii) repeated input attack.
- Attacks by extension of a jeopardized state: attempts to extend the benefits of a previous successful attack where a state S of the generator has been regained.

When working with PRNGs it is recommended using a cryptographic digest function (hash function) applied to the output of a PRNG if this is vulnerable to direct cryptanalytical attacks. This may increase the security of such a sequence, but decreases the speed of the generator. Other recommendation is to apply a summary function to the entry with a counter before using it. This helps prevent most of the attacks per chosen input. Inputs should be concatenated mod 2 bit by bit with time stamping and then summarized, before loading into the PRNG. It is also possible to generate from time to time a new PRNG input status. For PRNG generators such as ANSI X9.17, which leave a large part of their state unchanged after initialization, it is desirable to generate a new input state from the current one in the PRNG. In this way, the PRNG can reinitialize itself, with sufficient time and entropy. Finally, it is essential to pay special attention to the start and seed points of PRNGs guarantee the privacy (confidentiality) of the state of the generator.

In this context, new techniques have been developed that combine the methods explained in the previous section with others that use Deep-learning or new designs (that also incorporate TRNGs or uses other algorithms such Genetic Algorithm, etc.).

In IoT applications, the PRNGs used often have security vulnerabilities, i.e. they are not CSPRNGs. This is precisely due to hardware and software limitations. The generation of CSPRNGs in this context is a powerful and growing line of research. Table 8 shows some designs of cryptographically secure PRNGs together with a brief description and references. As can be seen, their mathematical basis is founded on some of the models discussed in the previous section, improved with coupling operations, incorporation of several interconnected generators, XOR operations or combinations with TRNGs among others.

As an example of the use of Deep-learning techniques, we could highlight [86] or [203]. In [86], a method for generating pseudo-random numbers is proposed that uses neural networks, in particular recurrent neural networks with long short-term memory. On the other hand, in [203] it is proposed a design that combines two PRNGs and a Physical unclonable function (PUF) for the generation of pseudo-random number sequences that are robust against LSTM attacks.

Examples of research that focus on designing generators that combine TRNGs and PRNGs include [66] or [88] among others. In [66] a generator is designed that combines a TRNG and a PRNG. The authors opt for this methodology because according to their design, they manage to use the advantages that both types of generators provide and in this way they obtain sequences that pass the usual statistical tests. They use a true random number as a seed for the application of the PRNG. Specifically, they start by extracting neuro-signals (using a low-resolution, cheap and portable encephalogram) that are considered seeds to apply a Linear congruential generator. A similar problem is addressed in [88]. The authors design a PRNG generator that uses as seed a (nearly) random sources produced from computer.

As an example of research in which it is combined a PRNG with Genetic Algorithm is [192]. In particular, it is used a LFSR.

Table 8. *New PRNGs designs for cryptographic purposes.*

PRNG	***Year***	***Description***	***Tested by***	***Reference***
Coupled-variable input LCG (CVLCG)	2020	Coupling of two variable input linear congruential generators for generating pseudo-random bit at every iteration	NIST	[153]
Gupta and Panda's method	2020	Three consecutive modules: → BiBiSeG module → FileShuffle module → RandKeyGen module which generate cryptographically secure key bit sequences of the desired length (say 128, 256, 512 or 1024)	NIST	[75]
Modified dual-CLCG	2019	Uses the congruential mod 2 addition of two different coupled LCG outputs To perform the modulo-2 addition operation, it takes only single XOR logic	NIST	[152]
Arrow	2017	Lightweight PRNG (from the family of Trifork PRNGs) It is based on combining, via bitwise XOR, the outputs of two simple Lagged Fibonacci generators	NIST Diehard	[149]
Warbler PRNG family	2016	Combination of modified de Bruijn blocks together with a Welch-Gong (WG) NLFSR Problem: almost every member of this family is vulnerable to linear attacks.(see [120])	NIST	[124]
Modified models of J3Gen	2015	Two PRNGs based on a linear feedback shift register (LFSR) configured with a multiple-polynomial tap architecture fed by a physical source of randomness	EPC Gen2 standard NIST	[30]
Warbler PRNG family	2015	New implementation of the Warbler family Two designs are presented for the counter: a binary counter and a LFSR counter		[209]
Warbler PRNG family	2013	Nonlinear feedback shift register PRNG Problem: vulnerable to linear attacks	EPC C1 Gen2 standard NIST	[123]
MeTuLR	2013	Mersenne Twister generator for RFID tags that uses random numbers as seeds, generated by using a property of the tags memories	EPC Gen2 standard NIST	[212]
J3Gen	2013	Combination of: → a TRNG (thermalnoise) and → a Dynamic Linear Feedback Shift Register (DLFSR) with multiple primitive polynomials Problem: Sensitive to certain cryptographic attacks(probabilistic and deterministic). See [163]	EPC C1 Gen2 standard	[135]
AKARI (AKARI-1, AKARI-2)	2011	Design that involves a non-linear filter function and a mapping of the form: $x \rightarrow x + (x_2 \vee C) \mod 2m$ The non-linear filter allows to overcome the problem that the mapping is not cryptographically secure	ENT, Diehard NIST	[130]
Trifork	2010	Combination of three coupled Perturbed Lagged Fibonacci Generators	Diehard NIST	[150]
Meliá-Seguí et. al. PRNG	2010	Improved design of the Che et al.'s PRNG, but by employing multiple primitive polynomials instead of one in the LFSR	EPC C1 Gen2 standard	[136]
Peris-López et. al. PRNG	2010	Blum-Blum-Shub (BBS) PRNG implemented for security levels 160 bit and 512 bits		[165]
TinyOS's PRNG	2009	Combines two random number generator modules: → RandomMlcgC (based on a multiplicative linear congruential generator) and → RandomLfsrC (built from a linear feedback shift register) Problem: Sensitive to certain cryptographic attacks due to their linear structures		[116]
LAMED	2009	Methodology based on Genetic Programming The update function for the internal state is based on bitwise XOR operator, modular algebra and bit rotations	Diehard NIST ENT SEXTON EPC C1 Gen2 standard	[164]
Katti and Srinivasan's PRBG generator	2009	Pseudo-random bit generator based on the couple of four mixed Linear Congruential Generators	NIST	[90]
Che et al.'s PRNG	2008	PRNG based on a combination of: → oscillator-based TRNG and → a linear feedback shift register (LFSR) with 16 stages Problem: → It is vulnerable due to the linearity of the LFSR → It is possible to obtain the polynomial associated with a few observations from the LFSR output. (See [136])		[29]
TinyRNG	2007	CSPRNG, its source of randomness comes from the received bit errors Implementation based on MICA2 motes		[60]
Low power 8-bit PRNG	2004	PRNG based on a free-running timer. Specifically designed for the i-Bean Network Can be used in other low-power embedded networks Problems: It has significant security concerns: (See [185]) → the exact entropy introduced by the transmitted and received messages in the network are undefined → how frequently it is necessary to apply the re-keying algorithm is not defined → the CRC of the transmitted packets has a very low entropy → the packets can bemodified by an attacker → the generated sequence has a short period → the internal secret state of the generator can be easily recovered by eavesdropping only two consecutive outputs	ENT	[185]

5. Conclusions

The generation of random sequences as discussed above is essential in several areas of knowledge. The essential statistical properties sought in these sequences are randomness, independence and uniformity (in which case any type of sequence can be generated from any other random variable). To the above properties we would add that of unpredictability if we were working in cryptography. From a practical point of view, and assuming that an arithmetic generator is used, it is also desirable that the period was as long as possible.

Focusing on the initial objectives of this research, we can affirm that they have been achieved:

- The concept of randomness and the main types of generators have been explained in detail: TRNGs, PRNGs and the subclass CSPRNGs. It has been emphasized which properties are desirable for the sequences to fulfill depending on the context in which they are to be used.
- This work has analyzed in detail the different types of arithmetic generators, which are characterized by a solid mathematical basis. They constitute a set of algorithms that allow the generation of sequences of pseudorandom numbers. Two main sub-types have been developed: linear and nonlinear. The most important generators in the literature have been defined, from the most rudimentary methods with the worst properties to the most sophisticated ones with better statistical properties and longer periods.
- The pros and cons of each of the types analyzed have also been explained, both from a statistical and computational point of view. Moreover, the new lines of research and new designs of PRNGs have been explained.

References

[1] Abdalla, M., Belaïd, S., Pointcheval, D., Ruhault, S. and Vergnaud, D. (2015). Robust Pseudo-Random Number Generators with Input Secure Against Side-Channel Attacks. Cryptology ePrint Archive, Report 2015/1219. MR3477002

[2] Afflerbach, L. and Grothe, H. (1988). The lattice structure of pseudorandom vectors generated by matrix generators. *Journal of Computational and Applied Mathematics* **23** 127–131. MR0952072

[3] Allard, J. L., Dobell, A. R. and Hull, T. E. (1963). Mixed Congruential Random Number Generators for Decimal Machines. *Journal of the ACM* **10** 131–141. MR0149645

[4] Almaraz Luengo, E. and García Villalba, L. J. (2021). Recommendations on Statistical Randomness Test Batteries for Cryptographic Purposes. *ACM Computing Surveys* **54**.

[5] Altman, N. S. (1988). Bitwise behavior of random number generators. *SIAM Journal on Scientific and Statistical Computing* **9** 941–949. MR0957484

[6] Banks, J., Carson, J. S. II, Nelson, B. L. and Nicol, D. M. (2010). *Discrete-Event System Simulation, 5th ed.* Pearson.

[7] Barnett, V. D. (1962). The behaviour of pseudorandom sequences generated on computers by the multiplicative congruential method. *Mathematics of Computation* **16** 63–69. MR0136046

[8] Bauer, W. F. (1958). The Monte Carlo Method. *Journal of the Society for Industrial and Applied Mathematics* **6** 438–451. MR0098455

[9] Bays, C. and Durham, S. D. (1976). Improving a poor random number generator. *ACM Transactions on Mathematical Software* **2** 59–64.

[10] Beasley, J. D. (1969). Design and testing of the System 4 random number generator. *The Computer Journal* **12** 368–372.

[11] Behrenz, P. G. (1962). Algorithm 133: Random. *Communications of the ACM* **5** 553–553.

[12] Beyer, A., Roof, R. B. and Williamson, D. (1971). The lattice structure of multiplicative congruential pseudorandom vectors. *Mathematics of Computation* **25** 345–363. MR0309263

[13] Blum, L., Blum, M. and Shub, M. (1986). A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing* **154** 364–383. MR0837589

[14] Blum, L., Cucker, F., Shub, M. and Smale, S. (1997). *Complexity and Real Computation*. Springer-Verlag, New York. MR1479636

[15] Blum, M. and Micali, S. (1982). How to generate cryptographically strong sequences of pseudo random bits. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)* 112–117. MR0780388

[16] Blum, M. and Micali, S. (2019). *How to Generate Cryptographically Strong Sequences of Pseudo Random Bits* In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali* 227–240. Association for Computing Machinery. MR0764183

[17] Bofinger, E. and Bofinger, V. J. (1958). On a periodic property of pseudorandom sequences. *Jounal of the ACM* **5** 261–265. MR0131949

[18] Borosh, I. and Niederreiter, H. (1983). Optimal multipliers for pseudo-random number generation by the linear congruential method. *BIT Numerical Mathematics* **23** 65–74. MR0689604

[19] Boyar, J. (1989). Inferring sequences produced by pseudo-random number generators. *Journal of the ACM* **36** 129–141. MR1072416

[20] Bratley, P., Fox, B. L. and Schrage, L. E. (1987). *A Guide to Simulation, 2d ed.* Springer-Verlag, New York.

[21] Brown, R. G., Eddelbuettel, D. and Bauer, D. (2014). Dieharder: A random number test suite (version 3.31.1).

[22] Brunner, D. and Uhl, A. (1999). Optimal Multipliers for Linear Congruential Pseudo-Random Number Generators with Prime Moduli: Parallel Computation and Properties. *Bit Numerical Mathematics* **39** 193–209. MR1697676

[23] Certaine, J. R. (1958). On sequences of pseudo-random numbers of maximal length. *Journal of the ACM* **5** 353–356. MR0127511

[24] Chakrabarty, D. (2016). Drawing of Random Six-Digit Numbers from

Tables of Random Three-Digit Numbers. *International Journal of Advanced Research in Science, Engineering and Technology* **3** 2507–2515.

[25] Chakrabarty, D. (2017). Drawing of Random Nine-Digit Numbers from a Single Table of Random Three-Digit Numbers. *International Journal of Advanced Research in Science, Engineering and Technology* **4** 3566–3577.

[26] Chakrabarty, D. (2017). Drawing of Random Four-Digit Numbers from a Single Table of Random Two-Digit Numbers. *International Journal of Advanced Research in Science, Engineering and Technology* **4** 3377–3387.

[27] Chakrabarty, D. (2018). Drawing of Random Ten-Digit Numbers from Tables of Random Two-Digit and Three-Digit Numbers. *International Journal of Advanced Research in Science, Engineering and Technology* **5** 6415–6420.

[28] Chakrabarty, D. and Sarmah, B. (2015). Examination of Proper Randomness of the Numbers Generated by L.H.C. Tippett (1927). *IOSR Journal of Mathematics* **11** 35–37.

[29] Che, W., Deng, H., Tan, W. and Wang, J. (2008). *A random number generator for application in RFID tags* In *Networked RFID Systems and Lightweight Cryptography* 279–287. Springer, Berlin.

[30] Chen, J., Miyaj, A., Sato, H. and Su, C. (2015). Improved lightweight pseudorandom number generators for the low-cost RFID tags. In *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). Vol.1.* 17–24.

[31] Church, A. (1940). On the concept of a random sequence. *Bulletin of the American Mathematical Society* **46** 130–135. MR0000911

[32] Corporation, R. (1955). *A million random digits.* Glencoe III, The Free Press.

[33] Coveyou, R. R. (1960). Serial Correlation in the Generation of Pseudo-Random Numbers. *Journal of the ACM* **7** 72–74. MR0117869

[34] Coveyou, R. R. and Macpherson, R. D. (1967). Fourier Analysis of Uniform Random Number Generators. *Journal of the ACM* **14** 100–119. MR0221727

[35] Craddock, J. M. and Farmer, S. A. (1971). Two Robust Methods of Random Number Generation. *The Statistician* **20** 55–66.

[36] Cusick, T. W. (1995). Properties of the $x^2 \mod N$ Pseudorandom Number Generator. *IEEE Transactions on Information Theory* **41** 1155–1159. MR1366760

[37] Danilowicz, R. L. (1989). Demonstrating the dangers of Pseudo-random numbers. *SIGCSE Bulletin* **21** 46–48.

[38] Deng, L. Y. and Lin, D. J. L. (2000). Random number generation for the new century. *The American Statistician* **54** 145–150.

[39] Denzer, V. and Ecker, A. (1988). Optimal multipliers for linear congruential pseudo-random number generators with prime moduli. *BIT* **28** 803–808. MR0972805

[40] Devroye, L. (1986). *Non-Uniform Random Variate Generation.* Springer-Verlag, New York. MR0836973

[41] Dieter, U. and Ahrens, J. (1971). An exact determination of serial

correlations of pseudo-random numbers. *Numerische Mathematik* **17** 101–123. MR0286245

[42] Dodd, E. L. (1942). Certain Tests for Randomness Applied to Data Grouped into Small Sets. *Econometrica* **10** 249–257. MR0007239

[43] Dodis, Y., Pointcheval, D., Ruhault, S., Vergniaud, D. and Wichs, D. (2013). Security analysis of pseudo-random number generators with input: /dev/random is not robust. In *CCS 2013 - Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security* 647–658. MR3131030

[44] Dodis, Y., Shamir, A., Stephens-Davidowitz, N. and Wichs, D. (2014). How to Eat Your Entropy and Have It Too – Optimal Recovery Strategies for Compromised RNGs. In *Advances in Cryptology – CRYPTO 2014* 37–54.

[45] Downhamd, D. Y. and Roberts, F. D. K. (1967). Multiplicative congruential pseudorandom number generators. *The Computer Journal* **10** 74–77.

[46] Eichenauer-Herrmann, J. (1991). On the discrepancy of inversive congruential pseudorandom numbers with prime power modulus. *Manuscripta Mathematica* **71** 153–161. MR1101266

[47] Eichenauer-Herrmann, J. (1992). Construction of inversive congruential pseudorandom number generators with maximal period length. *Journal of Computational and Applied Mathematics* **40** 345–349. MR1170913

[48] Eichenauer-Herrmann, J. (1992). Inversive Congruential Pseudorandom Numbers: A Tutorial. *International Statistical Review* **60** 167–176. MR1248899

[49] Eichenauer-Herrmann, J., Grothe, H., Niederreiter, H. and Topuzoglu, A. (1990). On the lattice structure of a nonlinear generator with modulus 2. *Journal of Computational and Applied Mathematics* **31** 81–85. MR1068151

[50] Eichenauer-Herrmann, J. and Niederreiter, H. (1992). Lower bounds for the discrepancy of inversive congruential pseudorandom numbers with power of two modulus. *Mathematics of Computation* **58** 775–779. MR1122066

[51] Eichenauer-Herrmann, J. and Topuzoglu, A. (1990). On the period length of congruential pseudorandom number sequences generated by inversions. *Journal of Computational and Applied Mathematics* **31** 87–96. MR1068152

[52] Eichenauer, J., Grothe, H. and Lehn, J. (1988). Marsaglia's lattice test and non-linear congruential pseudo random number generators. *Metrika* **35** 241–250.

[53] Eichenauer, J. and Lehn, J. (1986). A non-linear congruential pseudorandom number generator. *Statistische Papers* **27** 315–326. MR0877295

[54] Eichenauer, J., Lehn, J. and Topuzoglu, A. (1988). A nonlinear congruential pseudorandom number generator with power of two modulus. *Mathematics of Computation* **51** 757–759. MR0958641

[55] Fellen, M. (1969). An implementation of the Tausworthe generator.

Communications of the ACM **12** 413–413.

[56] Fisher, R. A. and Yates, F. (1938). *Statistical tables for biological, agricultural and medical research.* Oliver& Boyd. MR0030288

[57] Fisher, R. A. and Yates, F. (1963). *Statistical tables for biological, agricultural and medical research, 6th. ed.* Hafner Pub: New York. MR0030288

[58] Fishman, G. S. and Moore, L. S. I. (1986). An exhaustive analysis of multiplicative congruential random number generators with modulus $2^{31} - 1$. *SIAM Journal on Scientific and Statistical Computing* **7** 24–45. MR0819455

[59] Forsythe, A. B. (1968). Random-number generators. *Computers and Biomedical Research* **1** 470–474.

[60] Francillon, A. and Castelluccia, C. (2007). TinyRNG: A cryptographic random number generator for wireless sensors network nodes. In *Proceedings of the IEEE 5th Int. Symp. Modeling Optim. Mobile, Ad Hoc Wireless Netw., Apr. 2007* 1–7.

[61] Franklin, J. N. (1964). Equidistribution of Matrix-Power Residues Modulo One. *Mathematics of Computation* **18** 560–568. MR0172860

[62] Fushimi, M. (1989). Random Number Generation on Parallel Processors. In *Proceedings of the 1989 Winter Simulation Conference* 459–461.

[63] Fushimi, M. (1990). Random number generation with the recursion $x_t = x_{t-3p} - x_{t-3q}$. *Journal of Computational and Applied Mathematics* **31** 105–118. MR1068154

[64] Gage, R. M. S. (1943). Contents of Tippett's 'Random Sampling Numbers'. *Journal of the American Statistical Association* **38** 223–227. MR0007969

[65] Gardner, M. (1968). On the meaning of randomness and some ways of achieving it. *Scientific American* 116–121.

[66] Gavas, R. D., Navalyal, G. U., Stephens-Davidowitz, N. and Wichs, D. (2017). Fast and secure random number generation using low-cost EEG and pseudo random number generator. In *International Conference On Smart Technologies For Smart Nation (SmartTechCon), Bangalore* 369–374.

[67] Gentle, J. E. (2003). *Random Number Generation and Monte Carlo Methods, 2nd. ed.* Springer, New York, USA. MR2151519

[68] Goldreich, O., Goldwasser, S. and Micali, S. (1986). How to Construct Random Functions. *Journal of the ACM* **33** 792–807. MR0860526

[69] Golomb, S. W. (1982). *Shift Register Sequences, second edition.* Aegean Part Press, Laguna Hills, California. MR0242575

[70] Gordon, J. (1989). *Fast multiplicative inverse in modular arithmetic* In *Cryptography and Coding* 269–279. Clarendon Press, Oxford, United Kingdom. MR1030559

[71] Green, B. F., Smith, J. E. K. and Klem, L. (1959). Empirical Tests of an Additive Random Number Generator. *Journal of the ACM* **6** 527–537. MR0107957

[72] Greenberger, M. (1961). An A Priori Determination of Serial Correlation in Computer Generated Random Numbers. *Mathematics of Compu-*

tation **15** 383–389. MR0144489

[73] Gross, O. A. (1958). *Additive generation of pseudorandom numbers by O. Gross and S.M. Johnson.* RAND Corporation.

[74] Grothe, H. (1987). Matrix Generators for Pseudo-Random Vector Generation. *Statistical Papers* **28** 233–238. MR0975918

[75] Gupta, D. N. and Kumar, R. (2020). Generating Random Binary Bit Sequences for Secure Communications between Constraint Devices under the IOT Environment. In *2020 International Conference for Emerging Technology (INCET)Belgaum, India. Jun 5-7, 2020.* 1–6.

[76] Gustavson, F. G. and Liniger, W. (1970). A fast random number generator with good statistical properties. *Computing* **6** 221–226. MR0290520

[77] Haber, S. (1970). Sequences of numbers that are approximately completely equidistributed. *Journal of the ACM* **17** 269–272. MR0323746

[78] Hald, A. (1952). *Table of random numbers* In *A. Hald Statistical Tables and Formulas* 92–97. Wiley. MR0049515

[79] Hammer, C. and Green, L. (1964). Generation of Random Numbers. *Instruments and Control Systems* **37** 149–150.

[80] Hemmerle, W. J. (1969). Generating pseudorandom numbers on a two's complement machine such as the IBM 360. *Communications of the ACM* **12** 382–383.

[81] Hull, T. E. and Dobell, A. R. (1962). Random Number Generators. *SIAM Review* **4** 230–254. MR0148202

[82] Hull, T. E. and Dobell, A. R. (1964). Mixed Congruential Random Number Generators for Binary Machines. *Journal of the ACM* **11** 31–40. MR0161457

[83] Hutchinson, D. W. (1966). A New Uniform Pseudorandom Number Generator. *Communications of the ACM* **9** 432–433. MR0193739

[84] Jagerman, D. L. (1965). Some theorems concerning pseudorandom numbers. *Mathematics of Computation. American Mathematical Society* **19** 418–426. MR0184405

[85] James, F. (1980). Monte Carlo theory and practice. *Reports on Progress in Physics* **43** 1145–1189.

[86] Jeong, Y. S., Oh, K. and Cho, C. K. (2018). Pseudo random number generation using LSTMs and irrational numbers. In *IEEE International Conference on Big Data and Smart Computing (BigComp), Shanghai, China* 541–544.

[87] Kao, C. and Tang, H.-C. (1998). Several Extensively Tested Multiple Recursive Random Number Generators. *Computers & Mathematics with Applications* **36** 129–136. MR1645380

[88] Karimovich, G. S., Turakulovich, K. Z. and Ubaydullayevna, H. I. (2017). Computer's source based (Pseudo) random number generation. In *2017 International Conference on Information Science and Communications Technologies (ICISCT), Tashkent* 1–6.

[89] Kato, T., Wu, L.-M. and Yanagihara, N. (1996). On a nonlinear congruential pseudorandom number generator. *Mathematics of Computation* **65** 227–233. MR1325868

[90] Katti, R. S. and Srinivasan, S. K. (2009). Efficient hardware implementation of a new pseudo-random bit sequence generator. In *IEEE International Symposium on Circuits and Systems* 1393–1396.

[91] Katz, J. and Lindell, Y. (2008). *Introduction to Modern Cryptography.* CRC press. MR4283554

[92] Kelsey, J., Schneier, B., Wagner, D. and Hall, C. (1998). Cryptanalytic Attacks on Pseudorandom Number Generators. In *Fast Software Encryption, Fifth International Workshop Proceedings* (Springer-Verlag, ed.) 168–188.

[93] Kendall, M. G. (1941). A theory of randomness. *Biometrika* **32** 1–15. MR0004392

[94] Kendall, M. G. and Babington Smith, B. (1938). Randomness and Random sampling numbers. *Journal of the Royal Statistical Society* **101** 147–166.

[95] Kendall, M. G. and Babington Smith, B. (1939). A Table of Random Sampling Numbers. *Tracts for Computers* **24**.

[96] Knuth, D. (1965). Construction of a random sequence. *B.I.T.* **5** 246–250. MR0197434

[97] Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2 / Seminumerical Algorithms, 3th ed.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. MR3077153

[98] Kobayashi, H. (1978). *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology.* Addison-Wesley, Massachusetts, USA. MR0554755

[99] Krawczyk, H. (1992). How to predict congruential generators. *Journal of Algorithms* **13** 527–545. MR1187200

[100] Kruskal, J. B. (1969). Extremely portable random number generator. *Communications of the ACM* **12** 93–94.

[101] Law, A. M. (2015). *Simulation Modeling and Analysis, 5th ed.* McGraw-Hill. Education.

[102] Law, A. M. and Kelton, W. D. (2000). *Simulation Modeling and Analysis, 3th ed.* Boston: McGraw-Hill, c2000. MR0630193

[103] L'Ecuyer, P. (1988). Efficient and portable combined random number generators. *Communications of the ACM* **31** 742–751. MR0945034

[104] L'Ecuyer, P. (1990). Random Numbers for Simulation. *Communications of the ACM* **33** 85–97.

[105] L'Ecuyer, P. (1996). Combined Multiple Recursive Random Number Generators. *Operations Research* **44** 816–822.

[106] L'Ecuyer, P. (1997). Tests based on sum-functions of spacings for uniform random numbers. *Journal of Statistical Computation and Simulation* **59** 251–269.

[107] L'Ecuyer, P. (1999). Tables of linear congruential generators of different sizes and good lattice structure. *Mathematics of Computation* **68** 249–260. MR1489972

[108] L'Ecuyer, P. (1999). Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators. *Operations Re-*

search **47** 159–164. MR2114091

[109] L'Ecuyer, P. (2012). *Random Number Generation* In *Handbook of Computational Statistics, 2nd ed.* 35–71. Springer, New York. MR2985395

[110] L'Ecuyer, P., Blouin, F. and Couture, R. (1993). A Search for Good Multiple Recursive Random Number Generators. *ACM Transactions on Modeling and Computer Simulation* **3** 87–98.

[111] L'Ecuyer, P. and Panneton, F. (2009). *F_2 Linear Random Number Generators* In *International Series in Operations Research & Management Science* 169–193. Springer, Boston, MA.

[112] L'Ecuyer, P. and Simard, R. (1999). Beware of linear congruential generators with multipliers of the form $a = \pm 2^q \pm 2^r$. *ACM Transactions on Mathematical Software* **25** 367–374.

[113] L'ecuyer, P. and Simard, R. (2007). TestU01: AC library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)* **33** 1-40. MR2404400

[114] L'Ecuyer, P., Simard, R., Chen, E. J. and Kelton, D. (2002). An Object-Oriented Random-Number Package with Many Long Streams and Substreams. *Operations Research* **50** 1073–1075.

[115] Lehmer, D. H. (1951). Mathematical methods in large-scale computing units. In *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery, Harvard University Press, Cambridge, Massachusetts* 141–146. MR0044899

[116] Levis, P. and Gay, D. (2009). *TinyOS programming*.

[117] Lewis, P. A. W., Goodman, A. S. and Miller, J. M. (1969). A pseudo-random number generator for the System/360. *IBM Systems Journal* **8** 136–146.

[118] Lewis, T. G. and Payne, W. H. (1973). Generalized feedback shift register pseudorandom number algorithm. *Journal of the ACM* **20** 456–468.

[119] Maaranen, H., Miettinen, K. and Mkel, M. M. (2003). *Using Quasi Random Sequences in Genetic Algorithms* In *Optimization and Inverse Problems in Electromagnetism* 33–44. Springer, Dordrecht.

[120] Mabin, J., Gautham, S. and Balasubramanian, R. (2017). Distinguishing Attacks on (Ultra) Lightweight WG Ciphers. *Springer International Publishing* **12** 45–59. MR3647772

[121] MacLaren, M. D. and Marsaglia, G. (1965). Uniform random number generators. *Journal of the ACM* **12** 83–89. MR0170449

[122] Mamangakis, S. E. (1961). Remarks on the Fibonacci Series Modulo m. *The American Mathematical Monthly* **68** 648–649. MR0130206

[123] Mandal, K., Fan, X. and Gong, G. (2013). Warbler: A Lightweight Pseudorandom Number Generator for EPC C1 Gen2 Passive RFID Tags. *International Journal of RFID Security and Cryptography (IJRFIDSC)* **2** 82–91.

[124] Mandal, K., Fan, X. and Gong, G. (2016). Design and Implementation of Warbler Family of Lightweight Pseudorandom Number Generators for Smart Devices. *ACM Transactions on Embedded Computing Systems* **15**

1–28.

[125] Marsaglia, G. (1965). Regularities in congruential random number generators. *Numerische Mathematik* **16** 8–10. MR0273775

[126] Marsaglia, G. (1972). *The structure of linear congruential sequences* In *Applications of Number Theory to Numerical Analysis* 249–286. Edited by S. K. Zaremba, Academic Press, New York.

[127] Marsaglia, G. (1995). The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness.

[128] Martin-Löf, P. (1966). The definition of random sequences. *Information and Control* **9** 602–619. MR0223179

[129] Martin-Löf, P. (1969). Algorithms and randomness. *Review of the ISI* **37** 265–272.

[130] Martín, H., Millan, E. S., Entrena, L., López, P. P. and Hernández-Castro, J. C. (2011). AKARI-X: A pseudorandom number generator for secure lightweight systems. In *11th IEEE International On-Line Testing Symposium* 228–233.

[131] Matsumoto, M. and Kurita, Y. (1992). Twisted GFSR generators. *ACM Transactions on Modeling and Computer Simulation* **2** 179–184.

[132] Matsumoto, M. and Kurita, Y. (1994). Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation* **4** 245–266.

[133] Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random generator. *ACM Transactions on Modeling and Computer Simulation* **8** 3–30.

[134] McLeod, A. I. (1985). Remark AS R58: A Remark on Algorithm AS 183. An Efficient and Portable Pseudo-Random Number Generator. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **34** 198–200. MR0842077

[135] Meliá-Seguí, J., García-Alfaro, J. and Herrera-Joancomartí, J. (2013). J3Gen: A PRNG for low-cost passive RFID. *Sensors* **13** 3816–3830.

[136] Meliá-Seguí, J., J., G.-A. and J., H.-J. (2010). *Analysis and Improvement of a Pseudorandom Number Generator for EPC Gen2 Tags* In *Financial Cryptography and Data Security. FC 2010. Lecture Notes in Computer Science, vol. 6054* 34–46. Springer, Berlin, Heidelberg.

[137] Miller, J. C. P. (1963). Additive congruential pseudo-random number generators. *The Computer Journal* **11** 341–346. MR0235693

[138] Moses, L. and Oakford, R. (1963). Tables of Random Permutations. *Journal of Pharmaceutical Sciences* **52** 511–511.

[139] Naor, M. and Reingold, O. (2004). Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM* **51** 231–262. MR2145654

[140] Neuman, B. C. and Ts'o, O. (1994). Kerberos: An authentication service for computer networks. *IEEE Communications magazine* **32** 33–38.

[141] Niederreiter, H. (1977). Pseudo-random numbers and optimal coefficients. *Advances in Mathematics* **26** 99–181. MR0476679

[142] Niederreiter, H. (1978). Quasi-Monte Carlo methods and pseudo-

random numbers. *Bulletin of the American Mathematical Society* **84** 957–1041. MR0508447

[143] Niederreiter, H. (1985). The serial test for pseudo-random numbers generated by the linear congruential method. *Numerische Mathematik* **46** 51–68. MR0777824

[144] Niederreiter, H. (1986). A Pseudorandom Vector Generator Based on Finite Field Arithmetic. *Mathematica Japonica* **31** 759–774. MR0872797

[145] Niederreiter, H. (1988). Statistical independence of nonlinear congruential pseudorandom numbers. *Monatshefte fur Mathematik* **106** 149–159. MR0968332

[146] Niederreiter, H. (1989). The serial test for congruential pseudorandom numbers generated by inversions. *Mathematics of Computation* **52** 135–144. MR0971407

[147] Niederreiter, H. (1995). The multiple-recursive matrix method for pseudorandom number generation. *Finite Fields and Their Applications* **1** 3–30. MR1334623

[148] Niederreiter, H. (1995). Pseudorandom vector generation by the multiplerecursive matrix method. *Mathematics of Computation* **64** 279–294. MR1265018

[149] Orue López, A. B., Hernández Encinas, L., Martín Munoz, A. and Montoya Vitini, F. (2017). A Lightweight Pseudorandom Number Generator for Securing the Internet of Things. *IEEE Access* **5** 27800–27806.

[150] Orue López, A. B., Montoya, F. and Hernández Encinas, L. (2010). Trifork, a New Pseudorandom Number Generator Based on Lagged Fibonacci Maps. *Journal of Computer Science and Engineering* **2** 46–51.

[151] Owen, D. B. (1962). *Random numbers* In *Handbook of Statistical Tables* 517–538. Addison-Wesley series in statistics. MR0161401

[152] Panda, A. K. and Ray, C. K. (2019). Modified dual-CLCG method and its VLSI architecture for pseudorandom bit generation. *IEEE Transactions on Circuits and Systems I. Regular Papers* **66** 989–1002.

[153] Panda, A. K. and Ray, C. K. (2020). A Coupled Variable Input LCG Method and its VLSI Architecture for Pseudorandom Bit Generation. *IEEE Transactions on Instrumentation and Measurement* **69** 1011–1019.

[154] Pandit, P. and Bakshi, B. (2019). Comparative Study on the Degree of Randomness of Few Popular Random Number Tables. *Asian Journal of Probability and Statistics* **3** 1–8.

[155] Panneton, F., L'Ecuyer, P. and Matsumoto, M. (2006). Improved Long-Period Generators Based on Linear Recurrences Modulo 2. *ACM Transactions on Mathematical Software* **32** 1–16. MR2272349

[156] Park, S. K. and Miller, K. W. (1988). Random number generators: good ones are hard to find. *Communications of the ACM* **31** 1192–1201. MR1022039

[157] Park, S. K., Miller, K. W. and Stockmeyer, P. K. (1993). Technical Correspondence: Response. *Communications of the ACM* **36** 108–110.

[158] Passerat-Palmbach, J., Mazel, C. and Hill, D. R. C. (2011). Pseudo-Random Number Generation on GP-GPU. In *Principles of Advanced and Distributed Simulation (PADS), IEEE Workshop on Principles of Advanced and Distributed Simulation, Nice, 2011* 1–8.

[159] Payne, W. H. (1970). FORTRAN Tausworthe pseudorandom number generator. *Communications of the ACM* **13** 57–57.

[160] Payne, W. H., Rabung, J. R. and Bogyo, T. P. (1969). Coding the Lehmer Pseudorandom Number Generator. *Communications of the ACM* **12** 85–86.

[161] Peach, P. (1961). Bias in Pseudo-Random Numbers. *Journal of the American Statistical Association* **56** 610–618. MR0134452

[162] Peatman, J. G. and Shafer, R. (1942). A table of random numbers from Selective Service numbers. *Journal of Psychology* **14** 295–305.

[163] Peinado, A., Munilla, J. and Fúster-Sabater, A. (2014). EPCGen2 pseudorandom number generators: Analysis of J3Gen. *Sensors* **14** 6500–6515.

[164] Peris-López, P., Hernández-Castro, J. C., Estévez-Tapiador, J. M. and Ribagorda, A. (2009). LAMED a PRNG for EPC class-1 generation-2 RFID specification. *Computer Standards & Interfaces* **31** 88–97.

[165] Peris-López, P., San Millán, E., van der Lubbe, J. C. A. and Entrena, L. A. (2010). Cryptographically secure pseudo-random bit generator for RFID tags. In *International Conference for Internet Technology and Secured Transactions, London, UK.* 1–6.

[166] Plumstead, J. B. (1982). Inferring a sequence generated by a linear congruence. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium* 153–159. MR0780393

[167] Postelnicu, T. (1970). Rohlf, F. J., and R. R. Sokal: Statistical Tables. W. H. Freeman & Comp., San Francisco 1969. XI + 253 S., 5 Abb., Preis 641-. *Biometrische Zeitschrift* **12** 192–193.

[168] Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press. MR1201159

[169] Quenouille, M. (1959). Tables of Random Observations from Standard Distributions. *Biometrika* **46** 178–202. MR0102142

[170] Rao, C. R., Mitra, S. and Matthat, A. (1968). Formulae and Tables for Statistical Work. *Journal of the American Statistical Association* **63** 1064–1065.

[171] Rivest, R. L., Shamir, A. and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **21** 120–126. MR0700103

[172] Ross, S. M. (2006). *Simulation, 4th edition.* Elsevier. MR3294208

[173] Royo, J. and Ferrer, S. (1954). Tables of random numbers obtained from numbers in the Spanish National Lottery. *Trabajos de Estadistica* **5** 247–256.

[174] Royo, J. and Ferrer, S. (1954). *A table of random numbers* In *Tablas*

Estadisticas 3–127. Instituto de Investigaciones Estadisticas, Madrid.

[175] Rubinstein, R. Y. and Kroese, D. P. (2016). *Simulation and the Monte Carlo Method, 3th edition.* John Wiley & Sons, Incorporated. MR3617204

[176] Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J. and Vo, S. (2010). SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications Technical Report, Gaithersburg, MD, United States.

[177] Rusu, F. and Dobra, A. (2007). Pseudo-random number generation for sketch-based estimations. *ACM Transactions on Database Systems* **32** 1–48.

[178] Sarmah, B. K. and Chakrabarty, D. (2014). Testing of Randomness of Number generated by Fisher and Yates. *International Journal of Engineering Sciences & Research Technology* **3** 632–636.

[179] Sarmah, B. K. and Chakrabarty, D. (2015). Examination of Proper Randomness of the Numbers generated by L.H.C. Tippett (1927). *IOSR Journal of Mathematics* **11** 35–37.

[180] Sarmah, B. K. and Chakrabarty, D. (2015). Testing of proper Randomness of the numbers generated by Fisher and Yates (Applying t-test). *ABJMI Aryabhatta Journal of Mathematics & Informatics* **7** 87–90.

[181] Sarmah, B. K. and Chakrabarty, D. (2015). Examination of Proper Randomness of the Number Generated by Kendall and Babington Smith. *International Journal of Engineering Sciences & Research Technology* **4** 260–282.

[182] Sarmah, B. K., Chakrabarty, D. and Barman, N. (2015). Testing of Proper Randomness of the Table of Number Generated by Rand Corporation (1955). *International Journal of Engineering Sciences & Management* **5** 97–119.

[183] Schindler, W. (2009). *Random Number generators for cryptographic applications* In *Cryptographic Engineering* 5–23. Springer-Verlag, Boston, MA.

[184] Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd. ed.* John Wiley & Sons. MR3587912

[185] Seetharam, D. and Rhee, S. (2004). An efficient pseudo random number generator for low power sensor networks. In *29th Annual IEEE International Conference on Local Computer Networks* 560–562.

[186] Shamir, A. (1981). On the generation of cryptographically strong pseudorandom sequences. In *Automata, Languages and Programming* (S. Even and O. Kariv, eds.) 544–550. Springer Berlin Heidelberg, Berlin, Heidelberg.

[187] Smith, C. S. (1971). Multiplicative Pseudo-Random Number Generators with Prime Modulus. *Joumal of the ACM* **18** 586–593. MR0295522

[188] Stallings, W. (2011). *Cryptography and Network Security, Principles and Practices, 5th.ed.* Pearson.

[189] Steinhaus, H. (1954). Table of shuffled four-digit numbers. (In Polish, Russian and English.). *Rozprawy Matematyczne* **6** 1–46. MR0060185

[190] Stern, M. A. (1958). Ober eine zahlentheoretische Funktion. *Journal für die reine und angewandte Mathematik* **55** 193–220. MR1579066
[191] Strome, W. M. (1967). Algorithm 294: uniform random number. *Communications of the ACM* **10**.
[192] Sudeepa, K. B., Aitha, G., Rajinikanth, V. and Satapathy, S. C. (2020). Genetic algorithm based key sequence generation for cipher system. *Pattern Recognition Letters* **133** 341–348.
[193] Tausworthe, R. C. (1965). Random numbers generated by linear recurrence modulo two. *Mathematics of Computation* **19** 201–209. MR0184406
[194] Tayfur, A. and Melamed, B. (2007). *Simulation Modeling and Analysis with ARENA*. Elsevier Science & Technology.
[195] Thomas, D. B. and Luk, W. (2008). PGA-Optimised High-Quality Uniform Random Number Generators. In *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays* 235–244.
[196] Tippett, L. H. C. (1927). *Random number tables. Tracts for computer, No.-15.* Cambridge University Press.
[197] Tootill, J. P., Robison, W. D. and Adams, A. G. (1971). The runs up and down performance of Tausworthe pseudorandom number generators. *Journal of the ACM* **18** 381–399.
[198] Umans, C. (2003). Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences* **67** 419–440. MR2022839
[199] Vincent, C. H. (1970). The generation of truly random binary numbers. *Journal of Physics E: Scientific Instruments* **3** 594–598.
[200] von Neumann, J. (1951). *Various techniques used in connection with random digits* In *Monte Carlo Method, National Bureau of Standards Applied Mathematics Series, vol. 12* 36–38. U.S. Government Printing Office, Washington, D.C.
[201] Walker, J. (2008). ENT: A Pseudorandom Number Sequence Test Program.
[202] Wall, D. D. (1960). Fibonacci Series Modulo *m*. *The American Mathematical Monthly* **67** 525–532.
[203] Wen, Y. and Yu, W. (2019). Machine learning-resistant pseudo-random number generator. *Electronics Letters* **55** 515–517.
[204] Westlake, W. J. (1967). A Uniform Random Number Generator Based on the Combination of Two Congruential Generators. *Journal of the ACM* **14** 337–340.
[205] Whittlesey, J. R. (1968). A comparison of the correlational behavior of random number generators for the IBM 360. *Communications of the ACM* **11** 641–644.
[206] Wichmann, B. A. and Hill, I. D. (1982). Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **2** 188–190.
[207] Wichmann, B. A. and Hill, I. D. (1984). Correction: Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **33** 188–190.

[208] Wu, P.-C. (1997). Multiplicative, congruential random-number generators with multiplier $\pm 2^{k_1} \pm 2^{k_2}$ and modulus $2^p - 1$. *ACM Transactions on Mathematical Software* **23** 255–265.

[209] Yang, G., Aagaard, M. and Gong, G. (2015). Efficient hardware implementations of the Warbler pseudorandom number generator. *IACR Cryptology ePrint Archive* 1–13.

[210] Yao, A. (1982). Theory and applications of trapdoor functions. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science* 80–91.

[211] Yule, G. U. (1938). A Test of Tippett's Random Sampling Numbers. *Journal of the Royal Statistical Society* **101** 167–172.

[212] Özcanhan, M., Dalkilii, G. and Gürle, M. (2013). *An ultra-light PRNG for RFID tags* In *Computer and Information Sciences III* 231–238. Springer.