Article

Improving the Statistical Qualities of Pseudo Random Number Generators

Rafael Álvarez *, **Francisco Martínez** and **Antonio Zamora**

Department of Computer Science and Artificial Intelligence (DCCIA), University of Alicante, 03690 Alicante, Spain; fmartine@dccia.ua.es (F.M.); zamora@dccia.ua.es (A.Z.)
* Correspondence: ralvarez@dccia.ua.es

Abstract: Pseudo random and true random sequence generators are important components in many scientific and technical fields, playing a fundamental role in the application of the Monte Carlo methods and stochastic simulation. Unfortunately, the quality of the sequences produced by these generators are not always ideal in terms of randomness for many applications. We present a new nonlinear filter design that improves the output sequences of common pseudo random generators in terms of statistical randomness. Taking inspiration from techniques employed in symmetric ciphers, it is based on four seed-dependent substitution boxes, an evolving internal state register, and the combination of different types of operations with the aim of diffusing nonrandom patterns in the input sequence. For statistical analysis we employ a custom initial battery of tests and well-regarded comprehensive packages such as TestU01 and PractRand. Analysis results show that our proposal achieves excellent randomness characteristics and can even transform nonrandom sources (such as a simple counter generator) into perfectly usable pseudo random sequences. Furthermore, performance is excellent while storage consumption is moderate, enabling its implementation in embedded or low power computational platforms.

Keywords: random; pseudorandom; nonlinear; filter; PRNG; s-box

Citation: Álvarez, R.; Martínez, F.; Zamora, A. Improving the Statistical Qualities of Pseudo Random Number Generators. *Symmetry* **2022**, *14*, 269. https://doi.org/10.3390/sym14020269

Academic Editor: Chin-Ling Chen

Received: 31 October 2021
Accepted: 16 December 2021
Published: 29 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Random number and bit sequence generation is important in the development of many fields in science and technology, such as physics [1], biology [2], computational finance [3], nanostructures [4], semiconductors [5], security [6], data science [7], etc. It plays a fundamental role in the application of the Monte Carlo methods in those environments where a random behavior is apparent and stochastic simulation becomes essential [8–10].

Random number generators can produce number sequences based on sources of randomness, which can be either internal or external to the computer implementing such random generator. In the latter case, one or several natural physical phenomena can be sampled, combined, and processed, including sources such as particle duration in radioactive processes, thermal noise in an electrical resistor, noise from inactive microphones or cameras, frequency instability in an oscillator, etc. When the source of randomness is contained within the computer, resources such as the system clock, the frequency of key presses, the contents of input/output buffers, operating system statistics including processor load or memory usage, and even hard drive or network device latency are combined to distill enough randomness [11].

Despite their name, true random generators can have some drawbacks, such as the sequence being significantly skewed, having excessive autocorrelation and, especially, that the generated sequence cannot be easily reproduced. In the case of stochastic simulation and Monte Carlo methods, it is much more convenient to employ deterministic algorithms that produce sequences that are, in practice, indistinguishable from true random sequences and can be perfectly reproduced as a function of the input or seed; these are known as pseudo random generators (or PRNGs) and, although the generated sequences are not genuinely random, they behave as such for all practical purposes [12–14].

When employing pseudo random generators in simulation and Monte Carlo methods, the sequences produced must present long periods and a good statistical distribution, to be unpredictable. If an algebraic description of the properties of the sequences generated by a pseudo random generator is not possible (which is a fairly common occurrence), we have to resort to several statistical suites (batteries of tests) that can help detect possible uniformity deviations or dependencies [15,16]. One of the most popular and complete libraries is TestU01, that extends upon classical suites and includes several preconfigured groups of tests called SmallCrush, Crush and BigCrush [17]. In a similar fashion, PracRand [18] is also very popular and comprehensive.

The main contributions of this paper are based around the proposal of a new nonlinear filter design that can significantly improve the randomness characteristics of the sequences generated by common pseudo random generators. The results obtained by this new filter are excellent, presenting negligible performance overhead and completely correcting all defects present in the unfiltered sequences; even when using a simple counter (a completely nonrandom source) as a pseudo random generator. Due to its evolving internal state, the period of the filtered sequence is much longer than that of the input, which can be useful in cases where very long sequences are required. Furthermore, although our main focus in this paper are pseudo random sequences, the proposed filter can be also successfully applied to true random sequences without any modification.

The proposed design, although nonlinear and complex in nature, remains symmetric (balanced) in terms of the probability of each bit of the sequence, guaranteeing that all output values are equally probable. Moreover, its design employs certain concepts borrowed from symmetric ciphers [19], such as an initialization stage, seed (or key) dependent construction and the combination of different types of operations.

The main motivation of the proposed filter design is improving the quality of pseudo random sequences, not just in terms of statistical randomness, but also in terms of unpredictability and nonlinearity, which are extremely important for applications in the fields of cryptography and information security. There are known filtering algorithms that can correct for simple statistics but introduce problems in higher order metrics (a problem commonly described as resilience). The comprehensive testing performed ensures that this does not happen with our proposal, improving on the unpredictability of simple (and otherwise unsuitable in practice) generators by virtue of its highly nonlinear design.

Perhaps, the best known algorithm that employs random s-boxes is the RC4 stream cipher, which showed some problems in recent years [20–22] in terms of certain statistical biases. Our proposal differs from RC4 in several ways, mainly in its use of four 8×32 s-boxes that, once constructed, do not evolve with further filtering, unlike in RC4 that employs a single 8×8 s-box that is constantly evolving. Furthermore, the output sequence (*keystream*) generated by RC4 is directly sourced from its s-box values, while our proposal filters an external sequence and its output is not the direct result of any of the four s-boxes. Also, its internal structure is very different to RC4, with different operations and s-box implementation, wider data paths, and additional registers and variables.

S-boxes and their applications are a current and very active field of research, comprising an abundant body of related work, from which we can highlight the following research:

Regarding pseudo randomness, chaotic systems, and the construction of s-boxes, Özkaynak [23] has studied the relevance of chaotic systems on the construction of s-boxes, proposing with Tanyildizi [24] an s-box construction method based on one-dimensional chaotic maps. Hussain et al. [25] designed an alternative s-box for the Advanced Encryption Standard [26] that is constructed from a chaotic logistic map. Lu et al. [27] proposed an algorithm for the construction of s-boxes based on a new compound chaotic system, together with an efficient image encryption scheme based on the LSS chaotic map [28]. Wang et al. [29] proposed a chaotic s-box construction method based on a memorable simulated annealing algorithm. Jiang et al. [30] created an s-box with excellent properties based on chaos theory and bent functions. Lambi [31] applied a discrete-space chaotic map to s-box design. Zhou et al. [32] proposed a chaos-based random s-box generation algorithm based on a 2D mixed pseudo random coupling PS map lattice.

S-boxes are also frequently linked to the subject of image encryption, with Yang et al. [33] designing a 2D multiple collapse chaotic map with the aim of deriving an s-box and diffusion map that improve efficiency and security. Haq et al. [34] proposed a 12×12 s-box that is employed in color image encryption, improving statistical confusion. Zhang [35] studied a fast image cryptosystem based on a piecewise linear chaotic map and a cubic s-box, and Wang et al. [36] analyzed image encryption based on s-boxes constructed upon a chaotic system without equilibrium.

Machine learning is an active field of research, with applications to s-boxes. More specifically, Idris et al. [37] studied the prediction of active s-boxes on reduced Generalized Feistel Ciphers employing deep learning techniques. Among others, Zu et al. [38], Kim et al. [39], Mishra et al. [40], and Bolufé-Röhler et al. [41] also applied deep learning to s-box cryptoanalysis, design, and optimization in different types of ciphers.

This paper is divided as follows: Section 2 introduces the concepts required to understand the specific characteristics of the pseudo random generators employed to generate the sequences to be filtered. Section 3 describes the design of the filter, including the initialization and iteration (filtering) phases, and provides information regarding the motivation for such design. Section 4 details the testing results and procedures in terms of randomness and performance. Finally, some conclusions are given in Section 5.

2. Preliminaries

In the following, we detail the main characteristics of the three PRNGs that we tested with our proposed filter.

2.1. Linear-Feedback Shift Registers

Sequences generated by Feedback Shift Registers are well known since the beginning of electronics and are commonly employed in many disciplines, so there exists abundant literature regarding the subject [19,42–44]. We can identify two components in this type of generators: a shift register and a feedback function.

A Feedback Shift Register (FSR) is composed of an n-bit register, in which a sequential ordering of the bits was established, obtaining a structure that is similar to a FIFO (first in, first out) queue: when a new bit is introduced from the left to become the most significant bit, the remaining bits are shifted one position to the right and the least significant bit is extracted from the register as output. Also, the new bit that is introduced from the left is obtained as the result of a feedback function that takes the current state of the register as input. In this way, starting from an initial register value that acts as the seed and iterating the feedback process, each successive extracted bit conforms the output binary sequence.

The FSR generators with best characteristics are those that employ a linear feedback function and are known as Linear Feedback Shift Registers (LFSR). An n-bit LFSR can be characterized by a polynomial of $n-1$ degree in $\mathbb{Z}_2[x]$, achieving the highest possible period ($2^n - 1$) when this polynomial is primitive.

Algorithms based on LFSRs are popular for generating pseudo random sequences since they present the following desirable characteristics:

- They can be easily analyzed as they have known mathematical properties.
- Their hardware implementation is simple and efficient and, although not ideal, they can also be adequately implemented in software.
- As long as the feedback function is chosen from a primitive polynomial, the period of the sequence is guaranteed to be maximum.
- The generated sequences have very good randomness properties in terms of basic statistics.

In our tests, we chose a 48-bit LFSR with a feedback function constructed from the following primitive polynomial to obtain a maximal period for the given register length ($2^{48} - 1$).

$$x^{48} + x^{47} + x^{21} + x^{20} + 1,$$

and we show the corresponding LFSR diagram in Figure 1.

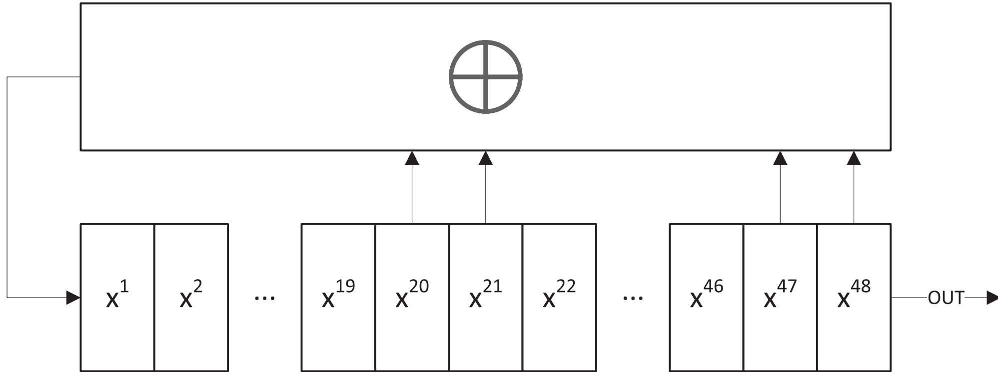

Figure 1. Schematic for the tested 48-bit LFSR generator.

2.2. Linear Congruential Generators

Introduced by Lehmer [45] in 1951, this algorithm is widely employed as it is used by default in many programming languages. It is based on the following expression:

$$X_{n+1} = (aX_n + c) \bmod m,$$

where the previous value (X_n) is multiplied by a coefficient (a), an offset (c) is added to it and the result modulo m constitutes the next value of the sequence (X_{n+1}). The name linear congruential refers to the expression of a line (linear) in modular arithmetic (congruential).

Despite generating sequences that, from the point of view of statistical randomness, are quite good and can have a fairly long period if the parameters are chosen properly, it is unsuitable for use in cryptography or security as it is predictable. Four values of the sequence are enough to be able to determine the seed and the necessary parameters (a, c and m) with which to reproduce the whole sequence. Unfortunately, because it is so popular, it is often used by inexperienced programmers in security tasks, allowing relatively simple attacks.

We chose a common 32-bit LCG implementation in many programming languages and standard libraries, with the following values:

$$a = 69069,$$
$$c = 1,$$
$$m = 2^{32} - 1.$$

2.3. Simple Counters

A sequence generated by a simple counter is, obviously, neither random nor pseudo random, quite the opposite. Nevertheless, the fact that it is a predictable sequence constitutes a very challenging test for the proposed filter to successfully correct such an unsuitable sequence and provide statistically acceptable results in terms of randomness.

Furthermore, counters have interesting properties when considered as part of PRNGs. They have maximal periods for a given register size (2^n possible values, being n the number of bits) and provide every possible output value once and only once, exhibiting certain statistical symmetry or output bit balance.

They are also very fast, providing a great baseline to establish maximum filter performance. Moreover, a counter feeding a filter is a very similar construct to that employed in the CTR (counter) mode of block ciphers [44].

For our testing, we have chosen a simple 32-bit counter (since that is a common integer variable size in modern implementations) that is set to an initial value (seed) and incremented accordingly:

$$X_0 = IV$$
$$X_{n+1} = (X_n + 1) \bmod 2^{32}$$

3. Description

Substitution boxes (or s-boxes) can be thought of conceptually as lookup tables that assign an output value for each possible input value. Although they vary in size, usually they are 8×8 bits (input and output values are a byte) or 8×32 bits (input values are a byte and output values are a four-byte word). They provide the required nonlinearity characteristics in PRNG or cryptographic algorithms, introducing complexity while maintaining high performance.

There are two main classes regarding their design:

- Static or constructed s-boxes. These are chosen carefully to achieve certain characteristics but usually employ some kind of generator function. Their main advantage is that they can achieve optimum values in certain metrics, but a weak point might be found since their data or underlying structure can be analyzed offline.
- Dynamic or key-derived s-boxes. Unlike static s-boxes, the results obtained by random s-boxes cannot be guaranteed to optimal in relevant metrics, but the fact they are not based on any known underlying structure can be a definite advantage. Furthermore, when pseudo random s-boxes are dynamically generated and derived from the seed then different s-boxes can be used for each different sequence, possibly providing even better results.

Our filter proposal is based on four 8×32 key-derived s-boxes that approximate a complete 32×32 substitution box, which would be unfeasible for this application in terms of computational and storage costs. This is a technique based on previous work [46,47]. It also includes an additional 64-bit internal state that is sometimes used as two 32-bit registers.

It is composed of two distinct phases: an initialization phase where the different s-boxes are constructed and a filtering phase where the input sequence is transformed into the output sequence. Both of these phases are described in the following.

3.1. Initialization

To construct the set of four 8×32 s-boxes, the initialization algorithm is based on the multiple states of a single 8×8 s-box that are concatenated to generate wider s-boxes. We define this s-box as $S = (S_0, S_1, \ldots, S_{255})$, with S_i denoting the i-th position in S, and 0 being the first position and 255 the last (as per computing standards).

This s-box is evolved during initialization using a set of values called the *keystream*, or K, that is a vector of 256 byte values that acts as the seed for the filter. This means that there are 2^{2048} different filter constructions possible and, since the initialization algorithm is deterministic, for a given seed value of $K = (K_0, K_1, \ldots, K_{255})$ we always obtain the same filter. As before, K_i denotes the i-th position in K, with 0 being the first position and 255 the last.

The filter can be seeded either from an external source or directly from the first 256 bytes of the input sequence before generating any output. The latter technique is employed throughout this paper.

For simplicity, we identify the byte values within K or S as the associated integer numbers.

Initially, S and integers h and l are

$$
\begin{aligned}
S_i &= i, i = 0, 1, \ldots, 255, \\
h &= 0, \\
l &= 0.
\end{aligned}
$$

Then, S is evolved in terms of K following

$$\begin{aligned} j &= (h + S_i) \bmod 256, \\ h &= (l + S_j + K_i) \bmod 256, \\ l &= l + i + K_h, \\ &\text{swap}(S_i, S_h). \end{aligned}$$

This process is repeated 256 times, swapping S_i and S_h values for $i = 0, 1, \ldots, 255$ to obtain an evolution state of S. We denote these states by a superscript, $S^n = (S^n_0, S^n_1, \ldots, S^n_{255})$ with n being the n-th evolution state of S and 0 the initial state of S. Therefore S^n_i denotes the i-th position of the n-th evolution state of S.

We can now define four 8×32 s-boxes, $M^q = (M^q_0, M^q_1, \ldots, M^q_{255})$, $q = 0, 1, 2, 3$, where M^q_i represents the i-th position in M^q, with 0 being the first position and 255 the last.

Notice that unlike S or K, positions in M^q contain 32-bit values (or four byte words) rather than bytes. These four s-boxes are constructed by the concatenation of four different evolutions of each position of S:

$$\begin{aligned} M^q_i &= (S^{4q+4}_i, S^{4q+3}_i, S^{4q+2}_i, S^{4q+1}_i), \\ i &= 0, 1, 2, \ldots, 255. \end{aligned}$$

This process is repeated for $q = 0, 1, 2, 3$ to construct all four 8×32 s-boxes and requires the S^1 to S^{16} evolution states of S. It is graphically represented in Figure 2.

For simplicity, we identify the 32-bit values (or four bytes) within M^q_i as a vector composed from the four integer numbers associated to these four bytes.

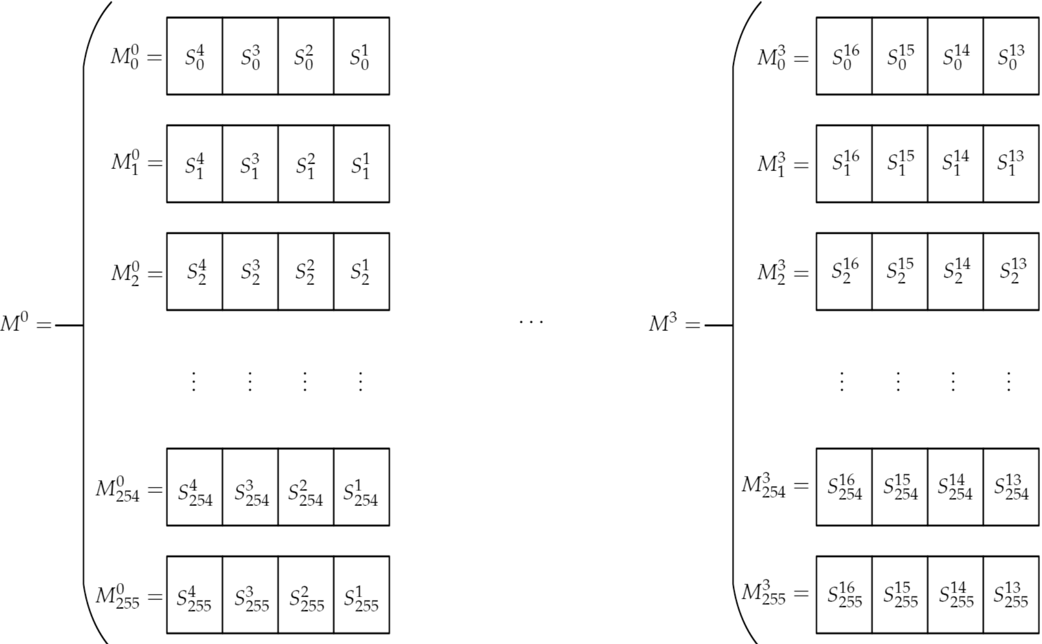

Figure 2. Construction of M^0 to M^3 taking byte values from S^1 to S^{16} evolutions states of S.

3.2. Filtering

Besides the four M^q s-boxes, the filter also employs a single 64-bit value $T = (T^h, T^l)$ that can also be addressed as two 32-bit halves, T^l (low 32-bit half) and T^h (high 32-bit half). Furthermore, we can also denote each byte of each half by subscript, with T^h_i denoting the i-th byte of the high half of T, with 0 being the least significant byte and 3 the most significant, this is $T^h = (T^h_3, T^h_2, T^h_1, T^h_0)$ and $T^l = (T^l_3, T^l_2, T^l_1, T^l_0)$.

Since the filter is designed to process a 32-bit word at a time, we can define I as the 32-bit word corresponding to the filter input and O as the 32-bit word corresponding to the filter output each iteration. So, for each 32 bits of input

$$\begin{aligned} T^h &= I, \\ T^l &= T^l \oplus M^0_{T^h_0} \oplus M^1_{T^h_1} \oplus M^2_{T^h_2} \oplus M^3_{T^h_3}, \\ O &= T^l, \\ T &= \mathrm{rol}_{31}(T). \end{aligned}$$

With $\mathrm{rol}_{31}(T)$ denoting the bitwise rotation of T 31 bits to the left, and $\oplus$ the bitwise exclusive-or between 32-bit operands.

Before filtering starts, T is initialized with K_0 to K_7

$$\begin{aligned} T^l_i &= K_i, \\ T^h_i &= K_{i+4}, \\ i &= 0, 1, 2, 3. \end{aligned}$$

The component architecture for each filter iteration is shown in Figure 3. The storage requirements of this design are only 4104 bytes (4KiB for the four 8×32 s-boxes and 8 bytes for the 64-bit T register).

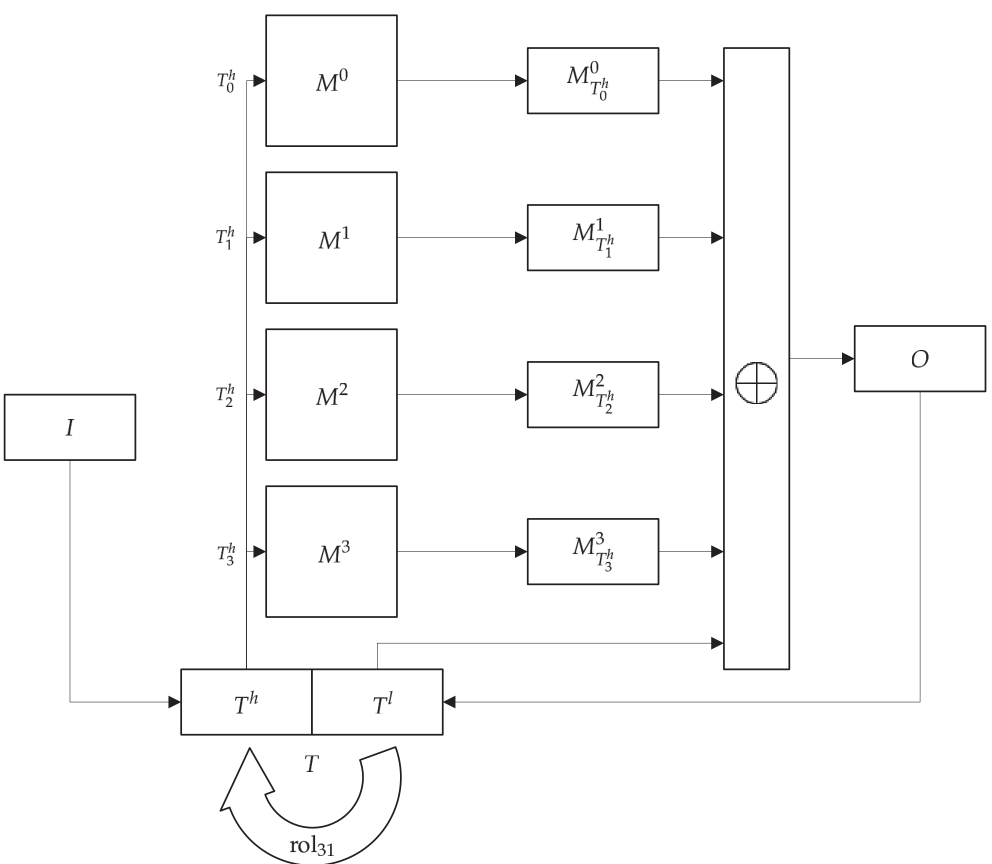

Figure 3. Filter iteration architecture.

3.3. Design Rationale

The proposed design presents several interesting aspects that are motivated in the following.

3.3.1. Data Dependent Construction

This filter follows a seed dependent construction, allowing for many different possible filter constructions. Although this characteristic adds complexity and is not common since most standard sequence filters are fixed or static, it provides better randomness results and broadens the possible applications of the proposed filter not only to simulation, but also to the fields of information security and cryptography. It is, in essence, inspired by the underlying structure of symmetric encryption algorithms such as stream and block ciphers [19].

3.3.2. Evolving Internal State

Although the construction is seed dependent, once the filter is constructed there is no additional evolution within the sets of s-boxes M^0 to M^3. To avoid that repeating bit patterns in the input generate similar output characteristics, the 64-bit register T acts as an evolving internal state that diffuses and decorrelates the input patterns along the output sequence, significantly improving results. As an additional benefit, it helps increase the overall period of the sequence. The fact that this register is 64-bit, while the input and output to the filter are 32-bit words, is also a technique inspired from symmetric ciphers.

3.3.3. Different Operation Types

The filter design involves different types of operations, like the exclusive-or that combines the outputs of different s-boxes, the bitwise rotation of the T register, and the additions to generate the indices in the filter initialization phase. The combination of different types of operations further improves the nonlinearity of the proposed filter.

4. Results

We include analysis of the results of the three generators described (CTR, LCG and LFSR) in terms of randomness and performance. Multiple byte units are expressed following the IEC recommended guidelines (see [48]).

4.1. Randomness Testing

As shown in Table 1, we performed initial statistical testing using a custom suite ([49]) that comprises the following randomness tests extracted from [19]:

- *Monobit*, related to the frequency of 0 and 1 bits in the sequence.
- *Serial*, frequency analysis of pairs of bits (00, 01, 10, and 11) in the sequence.
- *Poker8*, analyzing the frequency of groups of 8 bits.
- *Poker16*, like Poker8 but analyzing groups of 16bits.
- *Runs*, considering groups of contiguous 0 bits (gaps) and 1 bits (runs).

In this initial suite, a test is considered as passed when its result is below the correction value (with a significance level of $\alpha = 0.1$).

Both LFSR and LCG are good PRNG in terms of basic statistics, so they pass all tests regardless of filtering. Although this might lead to the belief that they are excellent PRNG for all applications, they are still predictable and completely linear, therefore unsuitable for cryptography or information security applications and will be deemed not random in the more complex testing suites that follow.

The simple counter (CTR) is a very different case, since it is not a random algorithm, so it performs very poorly in all tests until the proposed filter greatly improves the sequence to much better results.

Table 1. Initial statistical PRNG analysis.

	Monobit	Serial	Poker8	Poker16	Runs
LFSR (unf.)	0.23	0.44	245	65,632	20.9
LFSR (filt.)	0.36	2.28	220	65,376	25.8
LCG (unf.)	0.02	2.05	196	64,448	11.2
LCG (filt.)	0.01	0.58	254	64,896	10.8
CTR (unf.)	245	299	273,348	67,137,536	4151
CTR (filt.)	0.69	2.62	257	65,344	25.9
Correction	2.71	4.61	284	65,999	28.4

Further testing is performed with the TestU01 suite, specifically the included BigCrush battery that is comprised of 160 statistics [17]; this is a much more stringent set of tests and considered among the better tools for randomness testing, since it includes and further expands on the tests proposed by NIST [15] and those by Marsaglia [16].

We can see in Table 2 that all three algorithms fail a certain number of tests, while the proposed filter is able to correct the output and pass all tests; even in the extreme case of the simple counter that failed all tests without filtering.

Table 2. PRNG analysis with TestU01.

	Total	Failed
LFSR (unf.)	160	30
LFSR (filt.)	160	All passed
LCG (unf.)	160	101
LCG (filt.)	160	All passed
CTR (unf.)	160	All failed
CTR (filt.)	160	All passed

Another well-regarded randomness testing suite is PractRand (see [18]). This is a more modern set of tools than TestU01 but also comprehensive while supporting parallelism and iterative testing, therefore making it significantly more convenient and efficient to use.

The results in Table 3 show that the unfiltered results are poor, with the LFSR generator failing 18 tests out of a total of 130 after testing a sequence of 128 MiB; the LCG generator failing 89 tests out of 103 after testing a sequence of 64 MiB; and the counter failing almost all tests for a testing sequence of 32 MiB. The differing sequence lengths is due to the fact that PractRand performs tests incrementally, presenting results for successive length intervals, and stops testing when results are too poor to continue. This correlates with the results obtained in the previous testing suites.

As before, filtering improves results dramatically, passing all tests for all three algorithms even after testing a full terabyte (1 TiB) sequence, which exemplifies the great qualities of the proposed filter even for longer sequences, being capable of significantly extending the period of the original pseudo random sequences.

Table 3. PRNG analysis with PractRand.

	Length	Total	Failed
LFSR (unf.)	128 MiB	130	18
LFSR (filt.)	1 TiB	290	All passed
LCG (unf.)	64 MiB	103	89
LCG (filt.)	1 TiB	290	All passed
CTR (unf.)	32 MiB	112	106
CTR (filt.)	1 TiB	290	All passed

4.2. Performance

We include performance data for all three PRNG algorithms, both unfiltered and processed with the proposed filter. These performance values were taken as the minimum value of 10 runs generating a 10 GiB sequence, to minimize system variability; and were performed on a Windows 10 64 bit computer with an Intel Core i7-5960X 3.5 GHz CPU and 32 GB of RAM. The implementation was in pure C programming language over a single computing thread.

Figure 4 shows the time taken for all three PRNG to generate a sequence of the aforementioned length (10 GiB) with and without filtering, while Figure 5 presents the real-time bandwidth performance of each algorithm with and without filtering.

To properly interpret the performance results from Figures 4 and 5, we must consider the significance of each of the tested PRNG algorithms.

Taking into account the randomness of the sequences generated, a counter (CTR in our results) is not really a pseudo random generator since its sequences are always increasing values, and therefore easily predictable, but it is used here as a benchmark for the filtering capabilities of our proposal, that appear to be excellent since even a simple counter sequence passes all randomness tests after proper filtering. A similar occurrence happens with the linear congruential generator (LCG in our results), which is a well-known and popular PRNG algorithm for its simplicity and efficiency and was therefore included as the default basic PRNG in many programming languages, but it is predictable and will be detected as nonrandom in any tests other than the most basic. Linear feedback shift registers generators (LFSR in our results), albeit still simple algorithms, introduce a bit more complexity and produce much higher quality output sequences in terms of statistical randomness. Nevertheless, LFSRs have a very low linear complexity and are easily predictable employing the Berlekamp–Massey algorithm (see [50]), failing in more advanced randomness tests.

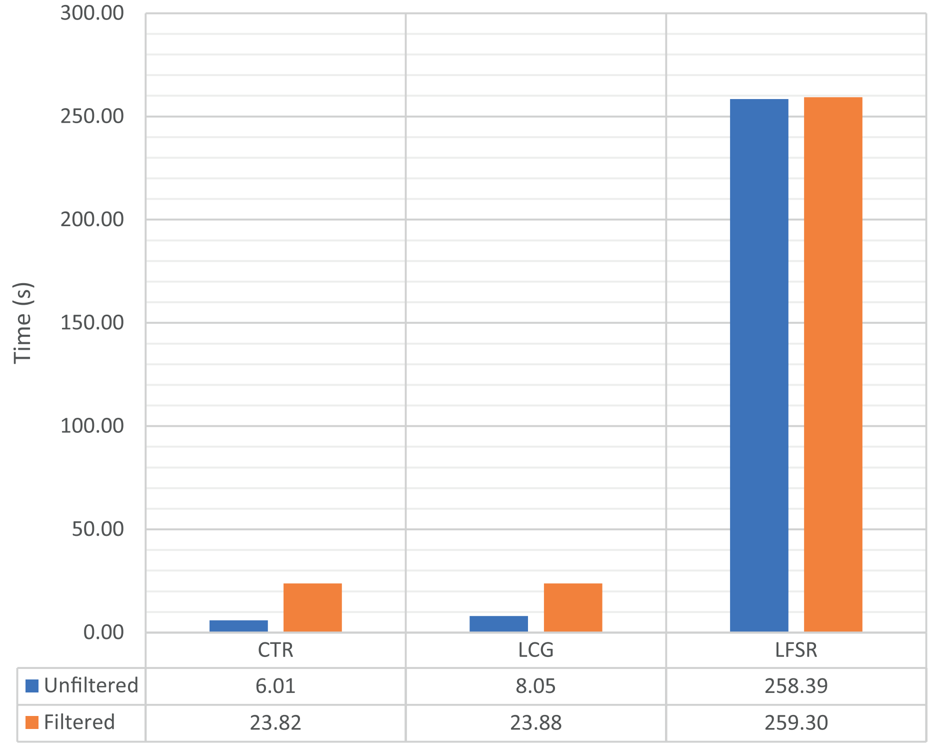

Figure 4. Time (in seconds) taken to generate a 10 GiB sequence.

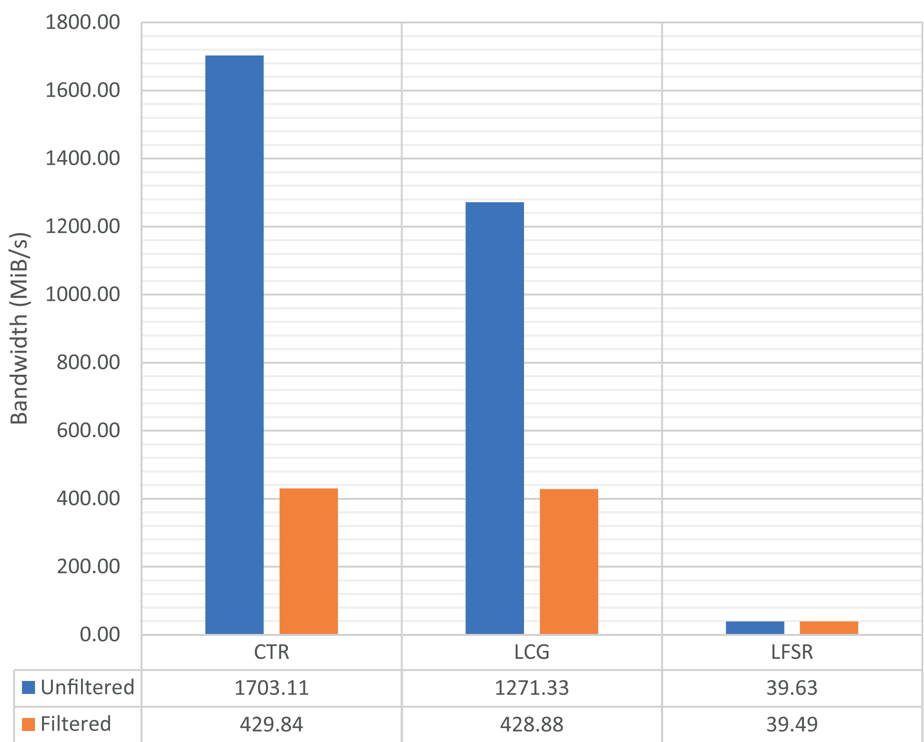

Figure 5. Bandwidth (in MiB/s) for each PRNG with and without filtering.

The perspective of time complexity regarding each algorithm is also important. The CTR algorithm is just a simple increment instruction over a CPU register, so it is extremely fast and mostly limited by memory bandwidth, procedure call overhead, etc. The LCG algorithm is a bit slower since it requires and addition, a multiplication and a modulo operation per iteration, with most implementations choosing values such that performance is maximized while maintaining a good enough period; such is the case of the LCG tested here. Traditionally, LFSR generators were chosen for their performance and simplicity of implementation in hardware but are less ideally suited for software implementation in modern CPUs since they are oriented to single bit manipulation, although this can be optimized employing alternative implementations, like the Galois form (see [51]).

In terms of the possible applications of these algorithms, an unfiltered CTR is not suitable by itself as a PRNG in any meaningful way for any application where randomness is required. An unfiltered LCG is only barely adequate for applications where performance is more important than statistical randomness, but it is a very poor option in any case where the quality of the randomness matters. An unfiltered LFSR is a better option when higher quality statistical randomness is required. It should be remarked that all these algorithms are predictable, and therefore not valid in isolation for applications such as cryptography or security protocols.

The filtering performance of our proposal enables the usage of these simple generators even in stringent application such as information security or cryptographic protocols because it improves randomness and the sequences pass all tests, but also introduces the required nonlinearity so that the filtered generators are no longer predictable.

We can see in Figure 5 that proposed filter presents excellent performance, achieving almost 430 MiB/s of bandwidth in software when filtering a simple counter. Considering that it entails just a simple increment instruction, this can be taken as the raw real-time performance of the proposed filter. To put this in context, a 1 gigabit network link has a bandwidth of 125 MiB/s. Furthermore, the proposed filter overhead is negligible when applied to proper generators (around 0.35% in the case of the LFSR), while being capable of efficiently adding the required complexity to generate properly pseudo random sequences.

These performance results are promising, enabling the application of the proposed filter in different computational scenarios.

Since the filtering operation is constant for each iteration, the computational complexity of the proposed algorithm is linear, $O(n)$, with the length of the input sequence.

5. Conclusions

We presented a new random number sequence filter design with interesting applications in many scientific and technical fields, including stochastic simulation. It is also applicable to information security and cryptography, considering that its design was motivated and inspired by many of the techniques employed in symmetric ciphers.

We also analyzed our proposal in terms of randomness and performance, obtaining excellent results. Our design was used to filter the output of different pseudo random generators, and the resulting sequences were then processed with three different statistical suites, including the well regarded TestU01 and PractRand packages. It manages to filter any statistical randomness problems in the input sequence and can even completely correct the many defects found in a nonrandom sequence, like those generated by a simple counter, transforming it into a perfectly usable random number sequence in practice.

Regarding performance, when filtering a simple counter generator, the proposed filter achieves approximately 430 MiB/s of bandwidth. This value can be taken as the nominal performance of the filter since the impact of the counter is minimal. Conversely, when filtering the output from a more complex generator, like the linear feedback shift register instance that was tested, the difference between filtered and unfiltered performance is negligible (0.35% in this case), so the performance impact of using the proposed filter on an existing generator is extremely low.

Since it presents great performance and modest resource consumption, its implementation could be useful in embedded systems or other low power computational scenarios such as transport or digital identity systems, etc., enabling high quality pseudo random sequences and improved security in many of these applications, considering that pseudo random sequences are the basis of many cryptographic primitives such as keystream sequences in Vernam style stream ciphers for encryption or random values in challenge/response schemes for authentication, etc.

Author Contributions: Conceptualization, R.Á., F.M. and A.Z.; methodology, R.Á., F.M. and A.Z.; software, R.Á., F.M. and A.Z.; validation, R.Á., F.M. and A.Z.; formal analysis, R.Á., F.M. and A.Z.; investigation, R.Á., F.M. and A.Z.; resources, R.Á., F.M. and A.Z.; data curation, R.Á., F.M. and A.Z.; writing—original draft preparation, R.Á., F.M. and A.Z.; writing—review and editing, R.Á., F.M. and A.Z.; visualization, R.Á., F.M. and A.Z.; supervision, R.Á. and A.Z.; funding acquisition, R.Á. and A.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Spanish Ministry of Science, Innovation and Universities (MCIU), the State Research Agency (AEI), and the European Regional Development Fund (ERDF) under project RTI2018-097263-B-I00 (ACTIS).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FSR	Feedback Shift Register
LFSR	Linear Feedback Shift Register
LCG	Linear Congruential Generator
CTR	Counter
PRNG	Pseudo Random Number Generator
IEC	International Electrotechnical Commission

References

1. Firk, F.; Miller, S. Nuclei, primes and the random matrix connection. *Symmetry* **2009**, *1*, 64–105. [CrossRef]
2. Klueter, A.; Crandall, J.; Archer, F.; Teece, M.; Coffroth, M. Taxonomic and environmental variation of metabolite profiles in marine dinoflagellates of the genus Symbiodinium. *Metabolites* **2015**, *5*, 74–99. [CrossRef]
3. Morokoff, W.J.; Caflisch, R.E. Quasi-Monte Carlo simulation of random walks in finance. In *Monte Carlo and Quasi-Monte Carlo Methods 1996*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 340–352.
4. Martins, E.R.; Li, J.; Liu, Y.; Depauw, V.; Chen, Z.; Zhou, J.; Krauss, T.F. Deterministic quasi-random nanostructures for photon control. *Nat. Commun.* **2013**, *4*, 2665. [CrossRef]
5. Cao, H.; Zhao, Y.; Ho, S.T.; Seelig, E.; Wang, Q.; Chang, R.P. Random laser action in semiconductor powder. *Phys. Rev. Lett.* **1999**, *82*, 2278. [CrossRef]
6. Álvarez, R.; Andrade, A.; Zamora, A. Optimizing a Password Hashing Function with Hardware-Accelerated Symmetric Encryption. *Symmetry* **2018**, *10*, 705. [CrossRef]
7. Jäntschi, L. A Test Detecting the Outliers for Continuous Distributions Based on the Cumulative Distribution Function of the Data Being Tested. *Symmetry* **2019**, *11*, 835. [CrossRef]
8. Law, A. *Simulation Modeling and Analysis*, 5th ed.; Mcgraw-Hill: New York, NY, USA, 2014.
9. L'Ecuyer, P. *Handbook of Computational Statistics*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 35–71.
10. Niederreiter, H. *Random Number Generation and Quasi-Monte Carlo Methods*; SIAM: Philadelphia, PA, USA, 1992.
11. Eastlake, D.; Crocker, S.; Schiller, J. *Randomness Recommendations for Security*; RFC, 1750; Network Working Group: Wilmington, DE, USA, 1994.
12. Billinton, R.; Wang, P. Teaching distribution system reliability evaluation using Monte Carlo simulation. *IEEE Trans. Power Syst.* **1999**, *14*, 397–403. [CrossRef]
13. Kroese, D.P.; Taimre, T.; Botev, Z.I. *Handbook of Monte Carlo Methods*; John Wiley & Sons: Hoboken, NJ, USA, 2013; Volume 706.
14. L'Ecuyer, P.; Munger, D.; Oreshkin, B.; Simard, R. Random numbers for parallel computers: Requirements and methods, with emphasis on gpus. *Math. Comput. Simul.* **2017**, *135*, 3–17. [CrossRef]
15. Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2000.
16. Marsaglia, G. *DIEHARD: A Battery of Tests of Randomness*; Technical Report; Florida State University: Tallahassee, FL, USA, 1996.
17. L'Ecuyer, P.; Simard, R. TestU01: A C library for empirical testing of random number generators. *ACM Trans. Math. Softw. (TOMS)* **2007**, *33*, 22. [CrossRef]
18. Doty-Humphrey, C. Practically Random: C++ Library of Statistical Tests for RNGs. 2016. Available online: https://sourceforge.net/projects/pracrand/ (accessed on 31 October 2021).
19. Menezes, A.J.; Van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC Press: Boca Raton, FL, USA, 2018.
20. Fluhrer, S.; Mantin, I.; Shamir, A. Weaknesses in the key scheduling algorithm of RC4. In *International Workshop on Selected Areas in Cryptography*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 1–24.
21. Klein, A. Attacks on the RC4 stream cipher. *Des. Codes Cryptogr.* **2008**, *48*, 269–286. [CrossRef]
22. Gupta, S.S.; Maitra, S.; Paul, G.; Sarkar, S. (Non-) random sequences from (non-) random permutations—Analysis of RC4 stream cipher. *J. Cryptol.* **2014**, *27*, 67–108.
23. Özkaynak, F. On the effect of chaotic system in performance characteristics of chaos based s-box designs. *Phys. A Stat. Mech. Its Appl.* **2020**, *550*, 124072. [CrossRef]
24. Tanyildizi, E.; Özkaynak, F. A new chaotic S-box generation method using parameter optimization of one dimensional chaotic maps. *IEEE Access* **2019**, *7*, 117829–117838. [CrossRef]
25. Hussain, I.; Anees, A.; Al-Maadeed, T.A.; Mustafa, M.T. Construction of s-box based on chaotic map and algebraic structures. *Symmetry* **2019**, *11*, 351. [CrossRef]
26. Daemen, J.; Rijmen, V. The first 10 years of advanced encryption. *IEEE Secur. Priv.* **2010**, *8*, 72–74. [CrossRef]
27. Lu, Q.; Zhu, C.; Wang, G. A novel S-box design algorithm based on a new compound chaotic system. *Entropy* **2019**, *21*, 1004. [CrossRef]
28. Lu, Q.; Zhu, C.; Deng, X. An efficient image encryption scheme based on the LSS chaotic map and single S-box. *IEEE Access* **2020**, *8*, 25664–25678. [CrossRef]
29. Wang, J.; Zhu, Y.; Zhou, C.; Qi, Z. Construction method and performance analysis of chaotic S-box based on a memorable simulated annealing algorithm. *Symmetry* **2020**, *12*, 2115. [CrossRef]
30. Jiang, Z.; Ding, Q. Construction of an S-Box Based on Chaotic and Bent Functions. *Symmetry* **2021**, *13*, 671. [CrossRef]
31. Lambić, D. A new discrete-space chaotic map based on the multiplication of integer numbers and its application in S-box design. *Nonlinear Dyn.* **2020**, *100*, 699–711. [CrossRef]
32. Zhou, P.; Du, J.; Zhou, K.; Wei, S. 2D mixed pseudo-random coupling PS map lattice and its application in S-box generation. *Nonlinear Dyn.* **2021**, *103*, 1151–1166. [CrossRef]
33. Yang, C.; Wei, X.; Wang, C. S-Box Design Based on 2D Multiple Collapse Chaotic Map and Their Application in Image Encryption. *Entropy* **2021**, *23*, 1312. [CrossRef] [PubMed]
34. ul Haq, T.; Shah, T. 12×12 S-box design and its application to R GB image encryption. *Optik* **2020**, *217*, 164922. [CrossRef]
35. Zhang, Y. The unified image encryption algorithm based on chaos and cubic S-Box. *Inf. Sci.* **2018**, *450*, 361–377. [CrossRef]

36. Wang, X.; Çavuşoğlu, Ü.; Kacar, S.; Akgul, A.; Pham, V.T.; Jafari, S.; Alsaadi, F.E.; Nguyen, X.Q. S-box based image encryption application using a chaotic system without equilibrium. *Appl. Sci.* **2019**, *9*, 781. [CrossRef]
37. Idris, M.F.; Teh, J.S.; Yan, J.L.S.; Yeoh, W.Z. A deep learning approach for active S-box prediction of lightweight generalized feistel block ciphers. *IEEE Access* **2021**, *9*, 104205–104216. [CrossRef]
38. Zhu, S.; Han, Y.; Wu, X. Time Sequence based AES S-box Implementation Cryptoanalysis using Deep Learning Approaches. In Proceedings of the 2020 39th Chinese Control Conference (CCC), Shenyang, China, 27–29 July 2020; IEEE: New York, NY, USA, 2020; pp. 7067–7072.
39. Kim, D.h.; Kim, S.; Hong, D.; Sung, J.; Hong, S. An Study on the Analysis of Design Criteria for S-Box Based on Deep Learning. *J. Korea Inst. Inf. Secur. Cryptol.* **2020**, *30*, 337–347.
40. Mishra, G.; Krishna Murthy, S.; Pal, S. Dependency of lightweight block ciphers over S-boxes: A deep learning based analysis. *J. Discret. Math. Sci. Cryptogr.* **2021**, *24*, 1–21. [CrossRef]
41. Bolufé-Röhler, A.; Tamayo-Vera, D. Machine learning based metaheuristic hybrids for S-box optimization. *J. Ambient. Intell. Humaniz. Comput.* **2020**, *11*, 5139–5152. [CrossRef]
42. Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*; Wiley: Hoboken, NJ, USA, 2015.
43. Smeets, B. A note on sequences generated by clock controlled shift registers. In *Workshop on the Theory and Application of of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1985; pp. 142–148.
44. Stallings, W. *Cryptography and Network Security: Principles and Practice*, 7th ed.; Pearson: London, UK, 2017.
45. Lehmer, D.H. Mathematical methods in large-scale computing units. *Annu. Comput. Lab. Harvard Univ.* **1951**, *26*, 141–146.
46. Álvarez, R.; McGuire, G. S-Boxes, APN functions and related codes. In *Enhancing Cryptographic Primitives with Techniques from Error Correcting Codes*; IOS Press: Amsterdam, The Netherlands, 2009; pp. 49–62.
47. Álvarez, R.; Zamora, A. Randomness analysis and generation of key-derived s-boxes. *Log. J. IGPL* **2015**, *24*, 68–79. [CrossRef]
48. International Electrotechnical Commission. *Letter Symbols to Be Used in Electrical Technology-Part 2: Telecommunications and Electronics*; Technical Report; International Electrotechnical Commission: Geneva, Switzerland, 2000.
49. Álvarez, R. RandTest. 2004. Available online: https://github.com/rias/randtest (accessed on 31 October 2021).
50. Ilani, I. Berlekamp-Massey Algorithm: Euclid in Disguise. In Proceedings of the 2018 IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE), Eilat, Israel, 2–14 December 2018; IEEE: New York, NY, USA, 2018; pp. 1–5.
51. Borodzhieva, A.N. Computer-Based Education for Teaching the Topic "Galois Linear Feedback Shift Registers". In Proceedings of the 2020 IEEE 26th International Symposium for Design and Technology in Electronic Packaging (SIITME), Pitesti, Romania, 21–24 October 2020; IEEE: New York, NY, USA, 2020; pp. 291–294.