A Pseudo-Random Bit Generator Based on Three Chaotic Logistic Maps and IEEE 754-2008 Floating-Point Arithmetic

Michael François[1], David Defour[2], and Pascal Berthomé[1]

[1] INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022, Bourges, France
{michael.francois,pascal.berthome}@insa-cvl.fr
[2] Univ. Perpignan Via Domitia, DALI F-66860, LIRMM UMR 5506 F-34095, Perpignan, France
david.defour@univ-perp.fr

Abstract. A novel pseudo-random bit generator (PRBG), combining three chaotic logistic maps is proposed. The IEEE 754-2008 standard for floating-point arithmetic is adopted and the binary64 double precision format is used. A more efficient processing is applied to better extract the bits, from outputs of the logistic maps. The algorithm enables to generate at each iteration, a block of 32 random bits by starting from three chosen seed values. The performance of the generator is evaluated through various statistical analyzes. The results show that the output sequences possess high randomness statistical properties for a good security level. The proposed generator lets appear significant cryptographic qualities.

Keywords: PRBG, Pseudo-random, Logistic map, Chaotic map, IEEE 754-2008.

1 Introduction

The generation of pseudo-random bits (or numbers) plays a crucial role in a large number of applications such as statistical mechanics, numerical simulation, gaming industry, communication or cryptography [1]. The term "pseudo-random" is used to indicate that, the bits (or numbers) appear to be random and are generated from an algorithmic process so-called generator. From a single initial parameter (or seed), the generator will always produce the same pseudo-random sequence. The main advantages of such generators are the rapidity and the repeatability of the sequences and require less memory for algorithm storage. Some fundamental methods are typically used to implement pseudo-random number generators, such as: non-linear congruences [2], linear feedback shift registers (LFSR) [3], discrete logarithm problem [4], quadratic residuosity problem [5], cellular automata [6], etc. In general, the security of a cryptographic pseudo-random number generator (PRNG), is based on the difficulty to solve the related mathematical problem. That usually makes the algorithm much slower, due to heavy computational instructions. For example, the Blum Blum Shub

T V Gopal et al. (Eds.): TAMC 2014, LNCS 8402, pp. 229–247, 2014.

algorithm [5] has a security proof, assuming the intractability of the quadratic residuosity problem. The algorithm is also proven to be secure, under the assumption that the integer factorization problem is difficult. However, the algorithm is very inefficient and impractical unless extreme security is needed. The Blum-Micali algorithm [4] has also an unconditional security proof based on the difficulty of the discrete logarithm problem, but is also inefficient.
Another interesting way to design such generators is connected to chaos theory [7]. That theory focuses primarily on the description of these systems that are often very simple to define, but whose dynamics appears to be very confused. Indeed, chaotic systems are characterized by their high sensitivity to initial conditions and some properties like ergodicity, pseudo-random behavior and high complexity [7]. The extreme sensitivity to the initial conditions (i.e. a small deviation in the input can cause a large variation in the output) makes chaotic system very attractive for implementing pseudo-random number generators. Obviously, chaos-based generators do not enjoy universal mathematical proofs compared with cryptographic ones, but represent a serious alternative that needs to be exploited. Moreover, during the last decade, several pseudo-random number generators have been proposed [8–14]. However, a rigorous analysis is necessary to evaluate the randomness level and the global security of the generator.

In this paper, a new PRBG using a standard chaotic logistic map is presented. It combines three logistic maps involving binary64 floating-point arithmetic and generates a block of 32 random bits at each iteration. The novelty of the paper is mainly based on the extraction mechanism of bits from the outputs of chaotic logistic maps. The produced pseudo-random sequences have successfully passed the various statistical tests. The assets of the generator are: high sensitivity to initial seed values, high level of randomness and good throughput. The paper is structured as follows, a brief introduction on floating-point arithmetic and the used chaotic logistic map is given in Sect. 2. Section 3 presents a detailed description of the algorithm. The statistical analysis applied on two groups of generated pseudo-random sequences is given in Sect. 4. The global security analysis of the PRBG is achieved in Sect. 5, before concluding.

2 Background

2.1 IEEE 754-2008 Standard

Digital computers represent numbers in sets of binary digits. For real numbers, two formats of representation can be distinguished: fixed-point format and floating-point format. The fixed-point format is designed to represent and manipulate integers or real numbers with a fixed precision. In the case of real numbers with variable precision, the representation is made through the floating-point format. There exists a standard that defines the arithmetic formats, the rounding rules, the operations and the exception handling for floating-point arithmetic.

The IEEE 754-2008 [15] is the current version of the technical standard, used by hardware manufacturer to implement floating-point arithmetic. Among them, binary32 (single precision) and binary64 (double precision) are the two most

widely used and implemented formats. As the generator described herein relies exclusively on binary64, we will only consider this format in the rest of the article.

Binary64 comprises two infinities, two kinds of NaN (Not a Number) and the set of finite numbers. Each finite number is uniquely described by three integers: s a sign represented on 1 bit, e a biased exponent represented on 11 bits and m a mantissa represented on 52 bits, where the leading bit of the significand is implicitly encoded in the biased exponent (see Fig. 1). To make the encoding unique, the value of the significand m is maximized by decreasing e until either $e = emin$ or $m \geq 1$. After this process is done, if $e = emin$ and $0 < m < 1$, the floating-point number is subnormal. Subnormal numbers (and zero) are encoded with a reserved biased exponent value. Interested readers will find a good introduction to floating point arithmetic and issues that arise while using it in [16].

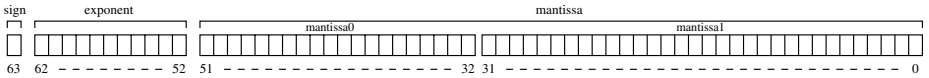

Fig. 1. Floating-point representation in double precision format (64 bits)

2.2 The Chaotic Logistic Map

The generator uses a chaotic logistic map given by:

$$F(X) = \lambda X(1 - X) \ , \tag{1}$$

with λ between 3.57 and 4.0 [17]. This function has been widely studied [18] and several pseudo-random number generators have already used such logistic map [12, 17, 19–22]. To avoid non-chaotic behavior (island of stability, oscillations, ...), the value of λ is fixed to 3.9999 that corresponds to a highly chaotic case [23]. The logistic map can be used under the iterative form:

$$X_{n+1} = 3.9999 X_n(1 - X_n), \forall n \geq 0 \ , \tag{2}$$

where the initial seed X_0 is a real number belonging to the interval $]0, 1[$. All the output elements X_n are also real numbers in $]0, 1[$.

3 The Proposed Generator

The main idea of the PRBG is to combine several chaotic logistic maps and carefully arrange them in the same algorithm in order to increase the security level. A block of 32 random bits per iteration is produced using the following three logistic maps:

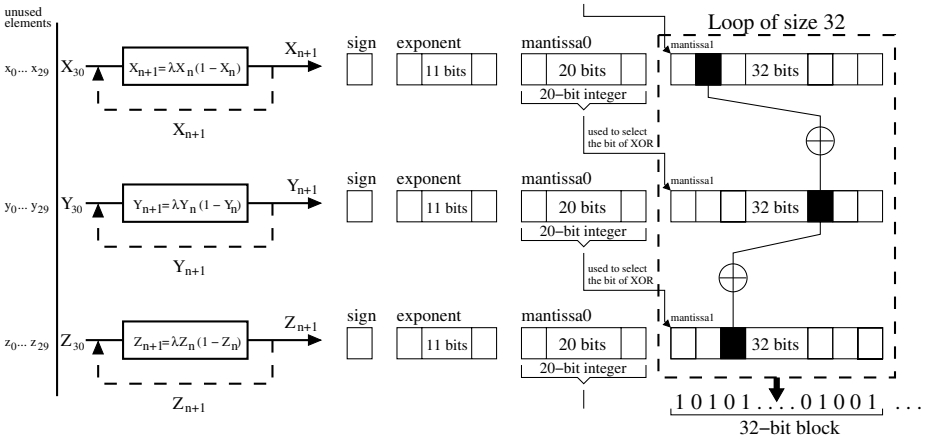

Fig. 2. Graphical description of the PRBG. In each mantissa1, the used bit (in xor) is moved at the end of chain and such process is not explicit on the scheme.

$$X_{n+1} = 3.9999X_n(1 - X_n), \forall n \geq 0 \ , \tag{3}$$

$$Y_{n+1} = 3.9999Y_n(1 - Y_n), \forall n \geq 0 \ , \tag{4}$$

$$Z_{n+1} = 3.9999Z_n(1 - Z_n), \forall n \geq 0 \ . \tag{5}$$

For the three chaotic maps, the same value of λ is chosen to maintain its surjectivity in the same interval. The graphical description of the generator is shown in Fig. 2. The technical details of the implementation in C, using definitions from the file *ieee754.h* are given in Algorithm 1. The algorithmic principle of the PRBG consists in three steps:

1. Line 2: three different seed values X_0, Y_0 and Z_0 are chosen to initiate the generation process (see Sect. 3.1).
2. Line 3–8: the results of the 30 first iterations are discarded to decorrelate the beginning of the output sequences (see Sect. 3.2).
3. Line 9–50: a loop of size N is started, with N being the length of the output sequence in block of 32 bits, then:
   (a) line 10–12: iterate the three logistic maps,
   (b) line 13–21: in each case, the bits of mantissa0 and mantissa1 are saved in two variables,
   (c) line 23–48: start another loop of size 32, and one bit is selected at a time from each mantissa1, according to the value of mantissa0. For more security, the value of the mantissa0 of Z_n is used to index the bits of the mantissa1 of X_n, the value of the mantissa0 of X_n to index the bits of mantissa1 of Y_n, and the value of mantissa0 of Y_n to index those of the mantissa1 in Z_n. Indeed, the bits of mantissa0 form a 20-bit integer, and by making a regressive modulo from 32, that allows to fix the position of the bit to be used. Thus, the three selected bits are combined by a

Algorithm 1. The PRBG algorithm

Require: $X_0; Y_0; Z_0; N$;
Ensure: A sequence of N blocks of 32 bits
Declaration: *union ieee754_double* $*F_1$, $*F_2$, $*F_3$;
Initialization: $i = 1; j = 1; X = X_0; Y = Y_0; Z = Z_0$;
while $i \leq 30$ **do**
  $X \leftarrow 3.9999 \times X \times (1 - X)$
  $Y \leftarrow 3.9999 \times Y \times (1 - Y)$
  $Z \leftarrow 3.9999 \times Z \times (1 - Z)$
  $i \leftarrow i + 1$
end while
while $j \leq N$ **do**
  $X \leftarrow 3.9999 \times X \times (1 - X)$
  $Y \leftarrow 3.9999 \times Y \times (1 - Y)$
  $Z \leftarrow 3.9999 \times Z \times (1 - Z)$
  $F_1 \leftarrow (union\ ieee754\_double\ *)\ \&\ X$
  $F_2 \leftarrow (union\ ieee754\_double\ *)\ \&\ Y$
  $F_3 \leftarrow (union\ ieee754\_double\ *)\ \&\ Z$
  $M_{0X} \leftarrow F_1 -> ieee.mantissa0$
  $M_{1X} \leftarrow F_1 -> ieee.mantissa1$
  $M_{0Y} \leftarrow F_2 -> ieee.mantissa0$
  $M_{1Y} \leftarrow F_2 -> ieee.mantissa1$
  $M_{0Z} \leftarrow F_3 -> ieee.mantissa0$
  $M_{1Z} \leftarrow F_3 -> ieee.mantissa1$
  $k \leftarrow 32$
  while $k > 0$ **do**
    $l \leftarrow k - 1$
    $P_X \leftarrow M_{0Z} \bmod k$
    $P_Y \leftarrow M_{0X} \bmod k$
    $P_Z \leftarrow M_{0Y} \bmod k$
    $B_x \leftarrow (M_{1X} >> (P_X)\ \&\ 1)$
    $B_y \leftarrow (M_{1Y} >> (P_Y)\ \&\ 1)$
    $B_z \leftarrow (M_{1Z} >> (P_Z)\ \&\ 1)$
    $B \leftarrow (B_x + B_y + B_z) \bmod 2$ {output bit}
    $b_x \leftarrow (M_{1X} >> (l)\ \&\ 1)$
    $b_y \leftarrow (M_{1Y} >> (l)\ \&\ 1)$
    $b_z \leftarrow (M_{1Z} >> (l)\ \&\ 1)$
    if $bx \neq B_x$ **then**
      $M_{1X} \leftarrow M_{1X}\ \hat{}\ (1 << (l))$
      $M_{1X} \leftarrow M_{1X}\ \hat{}\ (1 << (P_X))$
    end if
    if $by \neq B_y$ **then**
      $M_{1Y} \leftarrow M_{1Y}\ \hat{}\ (1 << (l))$
      $M_{1Y} \leftarrow M_{1Y}\ \hat{}\ (1 << (P_Y))$
    end if
    if $bz \neq B_z$ **then**
      $M_{1Z} \leftarrow M_{1Z}\ \hat{}\ (1 << (l))$
      $M_{1Z} \leftarrow M_{1Z}\ \hat{}\ (1 << (P_Z))$
    end if
    $k \leftarrow k - 1$
  end while
  $j \leftarrow j + 1$
end while

xor to give the output bit (line 31). From each mantissa1, the selected bit is then permuted with the bit at the end of chain to not be used again (line 35–46). At the end of this loop, a block of 32 random bits is produced. Such mechanism is definitely costly for the algorithm, but it allows to better decorrelate the outputs of the PRBG, especially in case of a possible collision.

3.1 Seed Selection

The input and output values of the logistic map belong to $]0, 1[$. To increase the robustness of the generator, three identical logistic maps are then combined. To preserve such robustness, one must avoid constructing identical chaotic trajectories, that may occur when using inappropriate initial seeds. To understand such mechanism, it is important to know how a difference δ between two computed values X_n and Y_n at a given iteration n will propagate to the next iteration. Without loss of generality, we can assume that $Y_n = X_n(1 + \delta)$. From (2) we know that:

$$X_{n+1} = \lambda X_n(1 - X_n) \quad and \quad Y_{n+1} = \lambda Y_n(1 - Y_n) \ ,$$

which is equivalent to:

$$Y_{n+1} = X_{n+1}\left(1 + \frac{\delta - 2\delta X_n - \delta^2 X_n}{(1 - X_n)}\right) \ .$$

Therefore, the difference between Y_{n+1} and X_{n+1} is:

$$Y_{n+1} - X_{n+1} = \lambda\delta X_n(1 - 2X_n - \delta X_n) \ .$$

We can deduce that, the smallest difference between Y_{n+1} and X_{n+1} is reached when $\delta = (1 - 2X_n)/X_n$. Finally, as X_n approaches 2^{-1} we obtain:

$$\lim_{X_n \to 2^{-1}} (Y_{n+1} - X_{n+1}) = -\frac{\lambda\delta^2}{4} \ .$$

To avoid identical representations, this difference must be representable in binary64, that means $\lambda\delta^2/4$ must be greater than 2^{-53}. By hypothesis, we set $\lambda = 3.9999$, then $\delta > 2^{-26.5}$. In this case, such value of δ allows to start with different chaotic trajectories, but it does not prevent a possible collision of elements at a certain rank n, which is a rare phenomenon but not impossible. In binary64 floating-point arithmetic, the computed value $(1 - x)$ is equal to 1.0 for any $x \in]0, 2^{-53}[$. This means that, for an initial seed selected in the interval $]0, 2^{-53}[$ the computed value of (2) is equivalent to λX_n. To avoid such problem, initial seeds have to be chosen in the interval $]2^{-53}, 2^{-1}[$.

Overall, the first seed X_0 is a random floating-point number representable in binary64 in the interval $]2^{-53}, 2^{-1}[$. The two other seeds Y_0 and Z_0 are constructed by randomly choosing two binary64 floating-point numbers. However, the minimum gap between each pair of seeds must be greater than 2^{-53} to avoid identical representations.

3.2 Initial Chaotic Behavior

The trajectory of the logistic map, from a small starting value (here $10^{-15} \approx 2^{-49.82}$) is plotted in Fig. 3. This value can also be considered as the minimum gap between two initial seeds. The aim is to analyze the evolution of this gap, through the iterative process. One can remark that, for the first iterations the trajectory is not chaotic. Indeed, a small initial difference between two seeds, spreads slowly toward the leading bits of mantissa. This problem does not occur, when the initial seeds are very different. However, to decorrelate the beginning of the output sequences in both cases, it is necessary to discard the first iterations before starting the generation. Thus, to decorrelate the outputs and increase the security level of the PRBG, we choose that the generation will start from the 31st iteration.

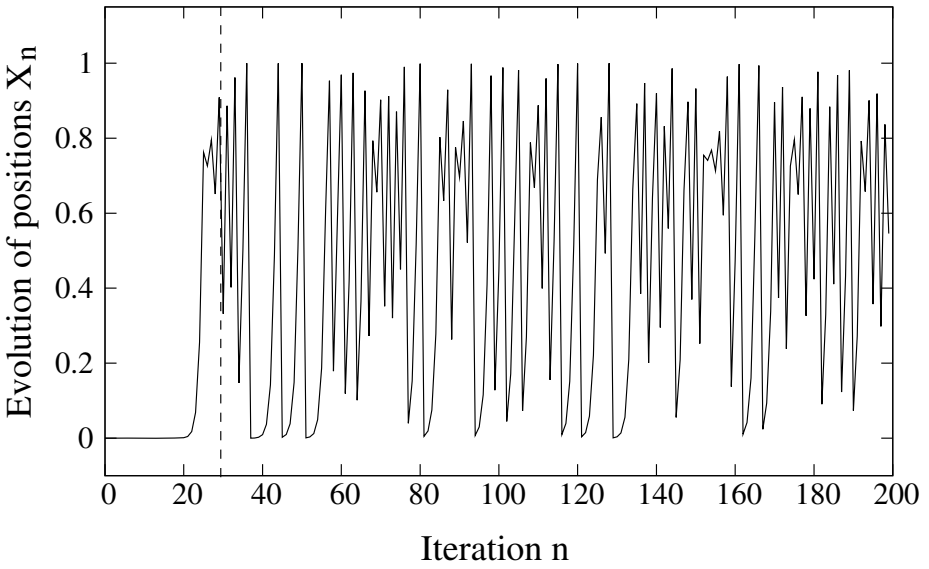

Fig. 3. Trajectory of the chaotic logistic map given in (2), for $X_0 = 10^{-15}$ and $n = 200$

4 Statistical Analysis

The output sequences of a PRBG must have a high level of randomness and be completely uncorrelated from each other. Therefore, a statistical analysis based on the randomness level and correlation should be carefully conducted to prove the quality of the sequences.

4.1 Randomness Evaluation

The analysis consists in evaluating the randomness quality of the sequences produced by the generator. Therefore, the sequences are evaluated through statistical tests suite NIST (National Institute of Standards and Technology of the U.S. Government). Such suite consists in a statistical package of fifteen tests developed to quantify and to evaluate the randomness of binary sequences produced by cryptographic random or pseudo-random number generators [24]. For each statistical test, a set of p_{value} is produced and compared to a fixed significance level $\alpha = 0.01$. A p_{value} of zero indicates that, the tested sequence appears to be not random. A p_{value} larger than α means that, the tested sequence is considered to be random with a confidence level of 99%. Therefore, a sequence passes a statistical test for $p_{value} \geq \alpha$ and fails otherwise. If at the same time more than one sequence is tested, each statistical test defines a proportion η as the ratio of sequences passing successfully the test relatively to the total number of tested sequences T (i.e. $\eta = n[p_{value} \geq 0.01]/T$). The proportion η is compared to an acceptable proportion η_{accept} which corresponds to the ratio of sequences that should pass the test. The range of acceptable proportions, excepted for the tests *Random Excursion-(Variant)* is determined by using the confidence interval defined as $(1 - 0.01) \pm 3\sqrt{0.01(1 - 0.01)/T}$ [24]. To analyze various aspects of the sequences, the NIST tests are applied on: individual sequences, the concatenated sequence and resulting sequences.

1. Individual sequences: all the produced sequences are individually tested and the results are given as ratio of success relatively to the threshold η_{accept}. Such test indicates the global randomness level of generated sequences.
2. Concatenated sequence: a new sequence of binary size $32 \times N \times T$ is constructed by concatenating all the individual sequences. The randomness level of the constructed sequence is also analyzed with the NIST tests. In the case of truly uncorrelated random sequences, the concatenated sequence should also be random.
3. Resulting sequences: are the sequences obtained from the columns, if the produced sequences are superimposed on each other. Thus, N resulting sequences of binary size $32 \times T$ are constructed, by collecting for each position $1 \leq j \leq N$, the 32-bit bloc of each sequence. The NIST tests are used to analyze such resulting sequences. This approach is interesting especially for sequences generated with successive seed values and can show whether there is some hidden linear structures between the original sequences.

4.2 Correlation Evaluation

The correlation evaluation is achieved in two different ways. Firstly, the correlation between the generated sequences is analyzed globally by computing the Pearson's correlation coefficient of each pair of sequences [25]. Consider a pair of sequences given by: $S_1 = [x_0, \ldots, x_{N-1}]$ and $S_2 = [y_0, \ldots, y_{N-1}]$. Therefore, the corresponding correlation coefficient is:

$$C_{S_1,S_2} = \frac{\sum_{i=0}^{N-1} (x_i - \overline{x}) \cdot (y_i - \overline{y})}{\left[\sum_{i=0}^{N-1} (x_i - \overline{x})^2\right]^{1/2} \cdot \left[\sum_{i=0}^{N-1} (y_i - \overline{y})^2\right]^{1/2}} \ , \tag{6}$$

where x_i and y_i are 32-bit integers, $\overline{x} = \sum_{i=0}^{N-1} x_i/N$ and $\overline{y} = \sum_{i=0}^{N-1} y_i/N$ the mean values of S_1 and S_2, respectively. For two uncorrelated sequences, $C_{S_1,S_2} = 0$. A strong correlation occurs for $C_{S_1,S_2} \simeq \pm 1$. The coefficients C_{S_1,S_2} are computed for each pair of produced sequences and the distribution of the values is presented by a histogram.

In the second approach, a correlation based directly on the bits of sequences is analyzed. The Hamming distance between two binary sequences (of the same length M) is the number of places where they differ, i.e., the number of positions where one has a 0 and the other a 1. Thus, for two binary sequences S_1^b and S_2^b, the corresponding Hamming distance is:

$$d(S_1^b, S_2^b) = \sum_{j=0}^{M-1} (x_j \oplus y_j) \ , \tag{7}$$

where x_j (resp. y_j) are the elements of S_1^b (resp. S_2^b). In the case of truly random binary sequences, such distance is typically around $M/2$, which gives a proportion (i.e. $d(S_1^b, S_2^b)/M$) of about 0.50. For each pair of produced sequences, this proportion is determined and all values are represented through a histogram. The interest of both approaches is to check the correlation for generated sequences mainly from nearby or successive seed values.

4.3 Analysis of Pseudo-Random Sequences

In the case of very distant seed values, the chaotic trajectories are very different, which usually allows to obtain good pseudo-random sequences. That is why the analysis is achieved on sequences produced from nearby or successive seed values. Here, two groups of pseudo-random sequences are considered. The binary length of each sequence is $32 \times N$ with $N = 1024$ and the total number of sequences per group is $T = 15000$. The first group (GRP1) is generated from the seed values $X_0 = 1 \times 10^{-15}, Y_0 = 2 \times 10^{-15}$ and $Z_0 = 3 \times 10^{-15}$ where each new sequence is obtained with the same values of X_0, Y_0 and by incrementing of 10^{-15} the last seed value Z_0. For the second group (GRP2), the same strategy is applied to the starting seeds $X_0' = 0.325873724698325, Y_0' = 0.325873724698326$ and $Z_0' = 0.325873724698327$. A simple loop on the latest seed values Z_0 and Z_0' allows to generate the two groups of sequences GRP1 and GRP2. The aim is to show whatever the structure of the initial seeds, the PRBG produces sequences of high quality.

Results of Randomness Evaluation. The results of NIST tests obtained on the two groups of 15000 sequences are presented in Table 1 and Table 2, respectively. For individual sequences (resp. resulting sequences), the acceptable proportion should lie above $\eta_{accept} = 98.75\%$ (resp. $\eta'_{accept} = 98.04\%$). For the tests *Non-Overlapping* and *Random Excursions-(Variant)*, only the smallest percentage of all under tests is presented. In the case of individual sequences, the *Universal* test is not applicable due to the size of sequences. Table 1 and Table 2 show that, all the tested sequences pass successfully the NIST tests.

Table 1. Results of the NIST tests on the 15000 generated sequences of GRP1. The ratio η (resp. η') of p_{value} passing the tests are given for individual (resp. resulting) sequences and the p_{value} is given for the concatenated sequence.

Test Name	Indiv. Seq.		Concat. Seq.		Result. Seq.	
	η	Result	p_{value}	Result	η'	Result
Frequency	99.06	Success	0.338497	Success	99.31	Success
Block-Frequency	99.11	Success	0.673515	Success	98.92	Success
Cumulative Sums (1)	99.08	Success	0.589087	Success	99.21	Success
Cumulative Sums (2)	99.00	Success	0.408891	Success	99.21	Success
Runs	98.93	Success	0.343876	Success	99.02	Success
Longest Run	98.99	Success	0.417880	Success	99.02	Success
Rank	98.86	Success	0.788352	Success	98.63	Success
FFT	98.90	Success	0.609162	Success	98.24	Success
Non-Overlapping	99.26	Success	0.012083	Success	98.04	Success
Overlapping	99.00	Success	0.175000	Success	98.92	Success
Universal	-	-	0.366163	Success	98.43	Success
Approximate Entropy	98.93	Success	0.138980	Success	98.14	Success
Random Excursions	98.75	Success	0.100729	Success	98.12	Success
Random Ex-Variant	98.75	Success	0.043821	Success	97.71	Success
Serial (1)	98.91	Success	0.158943	Success	99.12	Success
Serial (2)	99.06	Success	0.367717	Success	98.92	Success
Linear Complexity	98.98	Success	0.975515	Success	98.73	Success

Results of Correlation Evaluation. Concerning the correlation analysis, the Pearson's correlation coefficient between each pair of the 15000 produced sequences is computed. For each group, the corresponding histogram is presented in Fig. 4. One can see that, the two histograms have the same shape and show that the computed coefficients are very close to 0. For the group GRP1 (resp. GRP2), around 99.02% (resp. 99.00%) of the coefficients have an absolute value smaller than 0.08. The histograms show that, the correlation between the produced sequences is very small. About the correlation analysis using the Hamming distance, the histograms are presented in Fig. 5. The distributions show that, all the proportions are around 50%. For the group GRP1 (resp. GRP2), around 98.45% (resp. 99.98%) of the coefficients belong to $]0.488, 0.512[$. The values for GRP2 are better, due to the entropy of seed values.

Table 2. Results of the NIST tests on the 15000 produced sequences of GRP2. The ratio η (resp. η') of p_{value} passing the tests are given for individual (resp. resulting) sequences and the p_{value} for the concatenated sequence is also given.

Test Name	Indiv. Seq.		Concat. Seq.		Result. Seq.	
	η	Result	p_{value}	Result	η'	Result
Frequency	99.09	Success	0.408718	Success	98.73	Success
Block-Frequency	99.14	Success	0.458897	Success	98.63	Success
Cumulative Sums (1)	99.16	Success	0.239274	Success	98.63	Success
Cumulative Sums (2)	99.00	Success	0.494016	Success	99.02	Success
Runs	98.99	Success	0.025894	Success	98.82	Success
Longest Run	98.92	Success	0.281249	Success	99.12	Success
Rank	99.01	Success	0.806842	Success	99.12	Success
FFT	98.75	Success	0.673608	Success	98.73	Success
Non-Overlapping	99.27	Success	0.012472	Success	98.04	Success
Overlapping	99.02	Success	0.711625	Success	99.12	Success
Universal	-	-	0.149652	Success	98.53	Success
Approximate Entropy	99.02	Success	0.532585	Success	98.33	Success
Random Excursions	96.29	Success	0.060350	Success	98.52	Success
Random Ex-Variant	97.53	Success	0.134550	Success	98.73	Success
Serial (1)	98.92	Success	0.291906	Success	99.60	Success
Serial (2)	99.04	Success	0.196383	Success	99.51	Success
Linear Complexity	98.99	Success	0.215418	Success	99.60	Success

5 Security Analysis

The global security analysis of the generator is carefully conducted. The analysis is based on all the critical points allowing to detect weaknesses in the generator. The investigated points are: the size of key space, key sensitivity, quality of outputs, weak or degenerate keys, speed performance and period length of the logistic map. Even if all the existing attacks can not be tested, the PRBG must resist to some basic-known attacks. In the present case, the resistance to three basic attacks (brute-force attack, differential and guess-and-determine attacks) is discussed.

5.1 Key Space

It is generally accepted that, today a key space of size smaller than 2^{128} is not secure enough. A good PRBG should have a large key space, to have a high diversity of choices for the generation. The proposed generator combines three chaotic logistic maps. A key is then a combination of three initial seeds, used to generate a pseudo-random bit sequence. We have set the conditions for seed selection in Sect. 3.1. The seed X_0 is a binary64 floating-point number selected from the interval $]2^{-53}, 2^{-1}[$. That corresponds to 2^{52} different combinations of mantissa times 51 different values for the exponent, which gives 51×2^{52} different seeds. The seeds Y_0 and Z_0 are selected such that there is a minimum

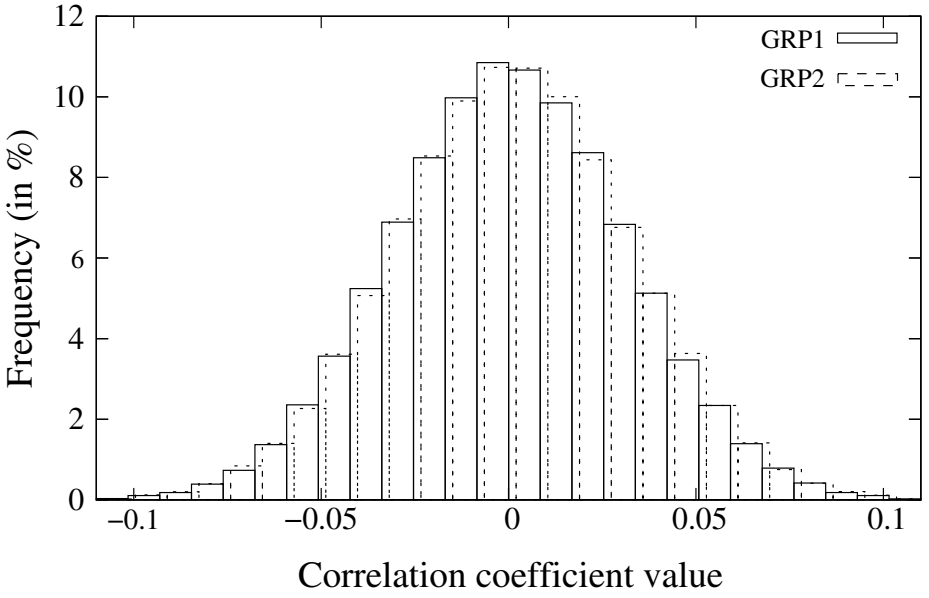

Fig. 4. Histogram of Pearson's correlation coefficient values on interval $[-0.1, 0.1]$ for the group GRP1 (resp. GRP2)

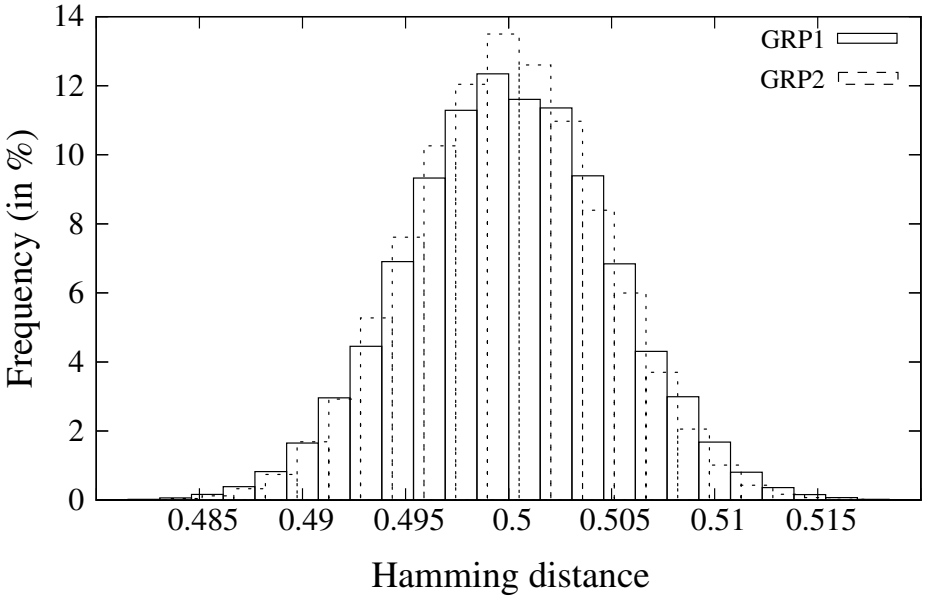

Fig. 5. Histogram of Hamming distance on interval $[0.485, 0.515]$ for the group GRP1 (resp. GRP2)

gap of $2^{-49.82}$ among each seeds. This means that Y_0 is selected in a space of $51 \times 2^{52} - 2^{49.82}$ possible seeds and Z_0 in a space of $51 \times 2^{52} - 2^{50.82}$ different numbers. The total space of seeds is approximately 2^{173}.

5.2 Key Sensitivity

The sensitivity related to the key (i.e. the seeds) is an essential aspect for chaos-based PRBG. Indeed, a small deviation in the starting seeds should cause a large change in the pseudo-random sequences. Actually in the test of correlation (Sect. 4.3), the seed sensitivity was already tested due to the successive seed values. To bring an additional analysis, large pseudo-random sequences of size $N = 5000000$ (i.e. 160000000 random bits) are considered. A sequence S_1 is produced by using the seed values $X_0 = 1 \times 10^{-15}, Y_0 = 2 \times 10^{-15}$ and $Z_0 = 4 \times 10^{-15}$. Two others sequences S_2 and S_3 are produced with $X_0' = X_0, Y_0' = Y_0, Z_0' = 3 \times 10^{-15}$ and $X_0'' = X_0, Y_0'' = Y_0, Z_0'' = 5 \times 10^{-15}$, respectively. The set of the three produced sequences is denoted KS1. The same approach is achieved from another set of sequences denoted KS2. The first sequence is generated with $X_0 = 0.328964524728163$, $Y_0 = 0.423936234268352$ and $Z_0 = 0.267367904037358$. The two supplementary sequences are obtained by decrementing and incrementing of 10^{-15} the last seed. In both cases, the analysis is done using the linear correlation coefficient of Pearson, the correlation coefficient of Kendall [26] and the Hamming distance. The same analysis is conducted on the sets KS1 and KS2 by using the algorithm proposed by Patidar et al. [2009], with the parameter $\lambda = 3.9999$. As the algorithm uses only two chaotic logistic maps, for each set of sequences only the last two seed values are considered. The results are given in Table 3 and show that, for the proposed algorithm the correlation coefficient values are close to 0 and the proportion of elements that differ in sequences are around 50%. The results show also that, the sequences are highly correlated for the Patidar's algorithm.

Another test of correlation using the randomness of the sequences is achieved. The test is to concatenate the three generated sequences and evaluate the obtained sequence through the NIST tests. The results are presented in Table 4. In each case, all the p_{value} are larger than 0.01 for the current PRBG. Therefore, the concatenated sequence can be viewed as a random sequence, which prove that the sequences S_1, S_2 and S_3 are completely uncorrelated. The results of the Patidar et al. algorithm are added to show that, it is not enough just to combine multiple chaotic logistic maps to build a secure generator.

5.3 Quality of Pseudo-Random Sequences

The strength of any generator is undeniably related to the quality of its outputs. Indeed, whichever way the algorithm is designed, the produced sequences must be strong (i.e. random, uncorrelated and sensitive). In the literature, various statistical tests are available to analyze the randomness of sequences. In fact, the NIST proposes a battery of tests, that must be applied on the binary sequences [24]. One can also find other batteries of tests, such as TestU01 [27] or

Table 3. Pearson's and Kendall's correlation coefficients and Hamming distance (in term of proportion) between large output sequences S_1, S_2, S_3 produced from slightly different initial seeds

PRBG	Set	Test	S_1/S_2	S_1/S_3	S_2/S_3
Proposed algorithm	KS1	Pearson Corr.	−0.000422	−0.000201	0.000127
		Kendall Corr.	−0.000150	−0.000437	−0.000141
		Ham. Dist.	0.499985	0.500064	0.500033
	KS2	Pearson Corr.	−0.000423	0.000235	0.000583
		Kendall Corr.	−0.000116	−0.000025	0.000199
		Ham. Dist.	0.500002	0.500055	0.499931
Patidar et al. algorithm	KS1	Pearson Corr.	0.329043	0.329214	0.329024
		Kendall Corr.	0.233170	0.231704	0.231653
		Ham. Dist.	0.333416	0.333362	0.333366
	KS2	Pearson Corr.	0.329542	0.329417	0.330055
		Kendall Corr.	0.231709	0.232413	0.231693
		Ham. Dist.	0.333284	0.333324	0.333354

the DieHARD suites [28]. Here, the NIST tests are adopted and all the produced sequences passed successfully the tests. The correlation between the outputs is evaluated and the results showed that, only a very small (or negligible) correlation exists between sequences. The proposed PRBG is also very sensitive to starting seeds, even when using slightly different seed values. That shows the quality of the pseudo-random sequences produced by the proposed generator.

5.4 Weak or Degenerate Keys

A crucial element for any PRBG is to ensure that, the output sequences are always produced from strong keys. Here, a careful study of the chaotic regions from the seed space, is necessary for avoiding weak keys. However, the first task is to choose a parameter λ of the logistic map, that contributes to have an excellent chaotic behavior. To avoid similar chaotic trajectories, the seed values must be chosen in $]2^{-53}, 2^{-1}[$, with a representable difference in binary64. The various statistical tests clearly showed the quality of tested sequences, from successive seed values. Thus, these regions are considered as homogeneously chaotic, allowing to choose independently the seed values in $]2^{-53}, 2^{-1}[$. Therefore, the proposed PRBG should not present weak or degenerate keys.

5.5 Speed Analysis

Beyond the randomness aspect, it is also necessary to have a fast generator. Indeed, in real-time applications, the temporal constraint in the execution of a

Table 4. Results of the NIST tests on the concatenated sequence obtained from the sequences of the set KS1 (resp. KS2) for the proposed and Patidar et al. algorithm

Test Name	Proposed algo.		Patidar et al. algo.	
	KS1	KS2	KS1	KS2
	p_{value}	p_{value}	p_{value}	p_{value}
Frequency	0.888272	0.189097	0.218594	0.933359
Block-Frequency	0.013966	0.409511	1.000000	1.000000
Cumulative Sums (1)	0.851269	0.250461	0.343496	0.679001
Cumulative Sums (2)	0.723802	0.375858	0.284913	0.757413
Runs	0.239428	0.196619	0.000000	0.000000
Longest Run	0.341867	0.124501	0.000000	0.000000
Rank	0.690933	0.468857	0.611764	0.788756
FFT	0.704824	0.837336	0.000000	0.000000
Non-Overlapping	0.014372	0.017263	0.000000	0.000000
Overlapping	0.544746	0.513071	0.000000	0.000000
Universal	0.693543	0.467674	0.000000	0.000000
Approximate Entropy	0.534042	0.087565	0.000000	0.000000
Random Excursions	0.016321	0.014831	0.013588	0.000622
Random Ex-Variant	0.014383	0.013285	0.125754	0.038526
Serial (1)	0.532881	0.383964	0.000000	0.000000
Serial (2)	0.508815	0.828262	0.000000	0.000000
Linear Complexity	0.956706	0.189871	0.817657	0.400909

process is as important as the result of the process. Thus, for a fast generator, the domain of its applications can be extended. The speed performance analysis is achieved on a work computer with processor: Intel(R) Xeon(R) CPU E5410 @ 2.33 GHz $\times$ 4. The source code is compiled using GCC 4.6.3 on Ubuntu (64 bits). The proposed generator enables to produce around 7 Gbits/s. This is an advantage for applications requiring a good security level and a fast execution time.

5.6 Period Length of the Logistic Map

Here, the period length of the logistic map is discussed. A PRBG should have a reasonably long period before its output sequence repeats itself. The idea is to build the trajectories formed by the different seed values and then compute the lengths of cycles. In a period-p cycle, $X_k = F^p(X_k)$ for some X_k, where F^p is the pth iterate of F. To analyze the evolution of cycles of the logistic map, the mantissa bits are modified by using the GNU MPFR library [29]. Figure 6 shows the curve representing the length of longest (resp. smallest) cycles, when the mantissa bits are varied between 10 and 25. One can see that, the logistic map has very small cycle lengths. For example under binary32 format, the computed longest (resp. smallest) cycle length is equal to 3055 (resp. 1). Such format is not appropriate for generating pseudo-random numbers. To obtain long periods,

it is necessary to consider more bits in the mantissa, in other words increase the precision. Thus, by using the binary64 format, the cycle lengths should be much longer. Indeed, the length of the longest cycle is 40037583 ($\approx 2^{25.25}$), while for the smallest cycle is 2169558 ($\approx 2^{21.04}$). In this case, only a given set of randomly chosen seeds is tested due to the large size of the binary format. Approximately we found the same cycle lengths than those given in [30]. For the proposed PRBG, three logistic maps are used during the generation process. In this way, the length of the global resulting cycle is given by the LCM of the three cycle lengths. As one can see, the best way to use this PRBG and then avoid the problem of short period, is to produce sequences of small sizes. However, if needed, long sequences can be obtained by concatenating several ones. In the case of maximum security, it might be better to limit the length of output sequences to the smallest cycle length.

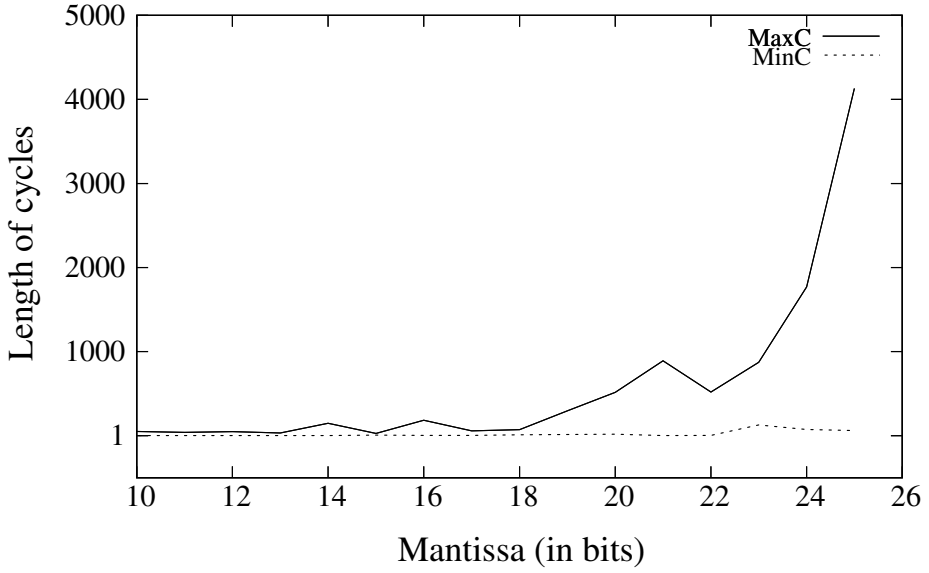

Fig. 6. The curve "MaxC" (resp. "MinC") representing the length of longest (resp. smallest) cycles, when the mantissa bits vary between 10 and 25

5.7 Basic Attacks

Here, the resistance of the generator against three basic attacks, such as brute-force attack, differential and guess-and-determine attacks is discussed.

Brute-force Attack. A brute-force attack [7] is a standard attack that can be used against any PRNG. The strategy consists in checking systematically all possible keys, until the correct key is found. In the worst case, all the combinations are tested, that necessitates to try all the key space. On average, just half

of the key space needs to be tested to find the original key. Such an attack might be utilized when it is not possible to detect any weakness in the algorithm, that would make the task easier. To resist this kind of attack, the size of the key space must be large. It is generally accepted that, a key space of size larger than 2^{128} is computationally secure against such attack. In this case, the size of the key space is around 2^{173}, which clearly allows to resist the brute force-attack.

Differential Attack. Such technique of cryptanalysis was introduced by Biham and Shamir [31]. As a chosen-plaintext attack, its principle is to analyze and exploit the effect of a small difference in input pairs on the difference of corresponding output pairs. This strategy allows to find the most probable key that was used to produce the pseudo-random sequence. Given two inputs I and I' (e.g. X_0, Y_0, Z_0 and X'_0, Y'_0, Z'_0) and the corresponding outputs O and O', the most commonly used differences are:

1. Subtraction modulus: the differences related to both inputs and outputs are defined by $\Delta_{in} = |I - I'|$ and $\Delta_{out} = |O - O'|$, respectively. Here, for inputs the difference can be computed between (X_0, X'_0), (Y_0, Y'_0) and (Z_0, Z'_0) and for outputs, between the bits of pseudo-random sequences.
2. Xor difference: defined by $\Delta_{in} = I \oplus I'$ and $\Delta_{out} = O \oplus O'$.

The diffusion aspect on the initial conditions is then measured by a differential probability. However, in the design of the algorithm, the decorrelation of outputs was taken into account by choosing the seed values in $]2^{-53}, 2^{-1}[$, and by making 30 iterations before starting the generation. Moreover, even with slightly different seeds, the produced sequences are almost independent from each other. Therefore, the proposed PRBG should resist to the differential attack.

Guess-and-determine Attack. Such kind of attack is proven to be effective against word-oriented stream ciphers [32]. As it comes from the name, in guess-and-determine attack, the strategy is to guess firstly the value of few unknown variables of the cipher and then, the remaining unknown variables are deduced by iterating the system a few times and by comparing the output sequence with the original sequence. If these two sequences are the same, then the guessed values are correct and the generator is broken, otherwise the attack should be repeated with new guessed values. It seems that the attack discussed in reference [32] can not be directly applied on the proposed algorithm, which is not of the same family of involved stream ciphers. Indeed, the internal structure of the cipher algorithm is completely different from a Linear Feedback Shift Register (LFSR). Here the algorithm starts with three seed values and generates a 32-bit bloc after each iteration. An alternative way to apply such attack would be to guess and fix the two seed values X_0 and Y_0, then iterate the algorithm to find the seed Z_0. Knowing that the algorithm is very sensitive to starting seeds, one should try in the worst case $2^{57.67}$ different values. Once all the comparisons made without success, the two initial seeds (X_0 and Y_0) are guessed again and the process is repeated in the same way until success. However, this approach has almost the same complexity than a classic brute-force attack.

6 Conclusions

A novel pseudo-random bit generator based on the combination of three chaotic logistic maps was presented. The generator uses the IEEE 754-2008 standard for floating-point arithmetic and especially the binary64 double precision format. For three given initial seeds, the algorithm produces a pseudo-random sequence formed of 32-bit blocks. The main strength of the generator is based on a special mechanism allowing to effectively extract the random bits. Such a generator has shown its ability to produce a very large number of pseudo-random sequences. The advantages of this PRBG are: a high sensitivity related to the initial seed values, a high randomness level of output sequences and the rapidity of the algorithm. The proposed scheme can be considered to be a serious alternative for generating pseudo-random bit sequences.

References

1. Sun, F., Liu, S.: Cryptographic pseudo-random sequence from the spatial chaotic map. Chaos Solit. Fract. 41(5), 2216–2219 (2009)
2. Eichenauer, J., Lehn, J.: A non-linear congruential pseudo random number generator. Statistische Hefte 27(1), 315–326 (1986)
3. Rose, G.: A stream cipher based on linear feedback over $GF(2^8)$. In: Boyd, C., Dawson, E. (eds.) ACISP 1998. LNCS, vol. 1438, pp. 135–146. Springer, Heidelberg (1998)
4. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudorandom bits. SIAM J. Comput. 13(4), 850–864 (1984)
5. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM J. Comput. 15(2), 364–383 (1986)
6. Tomassini, M., Sipper, M., Zolla, M., Perrenoud, M.: Generating high-quality random numbers in parallel by cellular automata. Future Gener. Comput. Syst. 16(2), 291–305 (1999)
7. Álvarez, G., Li, S.: Some basic cryptographic requirements for chaos-based cryptosystems. Int. J. Bifurcat. Chaos 16(8), 2129–2151 (2006)
8. Guyeux, C., Wang, Q., Bahi, J.M.: A pseudo random numbers generator based on chaotic iterations: Application to watermarking. In: Wang, F.L., Gong, Z., Luo, X., Lei, J. (eds.) Web Information Systems and Mining. LNCS, vol. 6318, pp. 202–211. Springer, Heidelberg (2010)
9. Zheng, F., Tian, X., Song, J., Li, X.: Pseudo-random sequence generator based on the generalized Henon map. J. China Univ. Posts Telecommun. 15(3), 64–68 (2008)
10. Pareschi, F., Setti, G., Rovatti, R.: A fast chaos-based true random number generator for cryptographic applications. In: Proceedings of the 32nd European Solid-State Circuits Conference, ESSCIRC 2006, pp. 130–133. IEEE (2006)
11. Pareek, N., Patidar, V., Sud, K.: A random bit generator using chaotic maps. Int. J. Netw. Secur. 10(1), 32–38 (2010)
12. Patidar, V., Sud, K., Pareek, N.: A pseudo random bit generator based on chaotic logistic map and its statistical testing. Informatica (Slovenia) 33(4), 441–452 (2009)

13. López, A.B.O., Marañon, G.Á., Estévez, A.G., Dégano, G.P., García, M.R., Vitini, F.M.: Trident, a new pseudo random number generator based on coupled chaotic maps. In: Herrero, Á., Corchado, E., Redondo, C., Alonso, Á. (eds.) Computational Intelligence in Security for Information Systems 2010. AISC, vol. 85, pp. 183–190. Springer, Heidelberg (2010)
14. François, M., Grosges, T., Barchiesi, D., Erra, R.: A new pseudo-random number generator based on two chaotic maps. Informatica 24(2), 181–197 (2013)
15. 754-2008 IEEE standard for floating-point arithmetic. IEEE Computer Society Std (August 2008)
16. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. 23(1), 5–48 (1991)
17. Bose, R., Banerjee, A.: Implementing symmetric cryptography using chaos functions. In: Proc. 7th Int. Conf. Advanced Computing and Communications, pp. 318–321. Citeseer (1999)
18. Weisstein, E.: Logistic map (2013), http://mathworld.wolfram.com/LogisticMap.html
19. Baptista, M.: Cryptography with chaos. Phys. Lett. A 240(1), 50–54 (1998)
20. Cecen, S., Demirer, R., Bayrak, C.: A new hybrid nonlinear congruential number generator based on higher functional power of logistic maps. Chaos Solit. Fract. 42(2), 847–853 (2009)
21. Xuan, L., Zhang, G., Liao, Y.: Chaos-based true random number generator using image. In: 2011 International Conference on Computer Science and Service System (CSSS), pp. 2145–2147. IEEE (2011)
22. François, M., Grosges, T., Barchiesi, D., Erra, R.: Pseudo-random number generator based on mixing of three chaotic maps. Commun. Nonlinear Sci. Numer. Simul. 19(4), 887–895 (2014)
23. Pareek, N., Patidar, V., Sud, K.: Image encryption using chaotic logistic map. Image Vis. Comput. 24(9), 926–934 (2006)
24. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, NIST Special Publication Revision 1a (2010)
25. Patidar, V., Pareek, N., Purohit, G., Sud, K.: A robust and secure chaotic standard map based pseudorandom permutation-substitution scheme for image encryption. Opt. Commun. 284(19), 4331–4339 (2011)
26. Kendall, M.: Rank correlation methods, 4th edn. Griffin, London (1970)
27. L'ecuyer, P., Simard, R.: Testu01: A C library for empirical testing of random number generators. ACM Trans. Math. Softw. 33(4), 22–es (2007)
28. Marsaglia, G.: Diehard: a battery of tests of randomness (1996), http://stat.fsu.edu/geo/diehard.html
29. The GNU MPFR library, http://www.mpfr.org
30. Keller, J., Wiese, H.: Period lengths of chaotic pseudo-random number generators. In: Proceedings of the Fourth IASTED International Conference on Communication, Network and Information Security, CNIS 2007, pp. 7–11 (2007)
31. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer, New York (1993)
32. Ahmadi, H., Eghlidos, T.: Heuristic guess-and-determine attacks on stream ciphers. Inf. Secur. IET 3(2), 66–73 (2009)