A Fast Chaos-Based Pseudo-Random Bit Generator Using Binary64 Floating-Point Arithmetic

Michael François
INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022, Bourges, France
E-mail: michael.francois@insa-cvl.fr

David Defour and Christophe Negre
Univ. Perpignan Via Domitia, DALI F-66860, LIRMM UMR 5506 F-34095, Perpignan, France
E-mail: {david.defour,christophe.negre}@univ-perp.fr

Keywords: PRBG, pseudo-random, logistic map, IEEE-754, cryptography

Received: June 27, 2013

Chaos-based cryptography is widely investigated in recent years, especially in the field of random number generators. The paper describes a novel pseudo-random bit generator (PRBG) based on chaotic logistic maps. Three logistic maps are combined in the algorithmic process, and a block of 32 random bits is produced at each iteration. The binary64 double precision format is used according to the IEEE 754-2008 standard for floating-point arithmetic. This generator provides a considerable improvement of an existing generator in the literature. Rigorous statistical analyses are carefully conducted to evaluate the quality and the robustness of the PRBG. The obtained results showed the relevance of the proposed generator, which is suitable even for real-time applications.

Povzetek: V članku je opisan hitri psevdo-naključni generator za kriptiranje.

1 Introduction

The generation of pseudo-random bits (or numbers) plays a critical role in various applications such as: statistical mechanics, numerical simulations, gaming industry, communication systems, cryptographic protocols and many others [1]. In practice, the generation of such numbers with randomness properties is an open problem and continues to be investigated. There are two main classes of generators: software and physical generators.

For the software generators, the term "pseudo-random" is applied to indicate that, the generator is defined as an algorithm allowing to produce sequences of bits with randomness properties. From a single initial seed, these generators will always produce the same outputs. The assets of such generators are: a fast execution time, repeatability and reproducibility of the pseudo-random sequences. The second class of generators exploits physical random phenomena for the generation, but is not discussed here.

Some basic techniques are often used for generating pseudo-random numbers, such as: linear recurrence [2], non-linear congruence [3], linear feedback shift register (LFSR) [4], cellular automata [5], discrete logarithm problem [6], quadratic residuosity problem [7], etc. Generally, the security of a cryptographic generator is based on the difficulty to solve the related mathematical problem. Beyond the security, such kind of generator is sometimes too slow due to heavy computational instructions. For example, the Blum Blum Shub algorithm [7] has a security proof, assuming the computational difficulty of the quadratic residuosity problem. The algorithm is also proven to be secure, relatively to the difficulty of integer factorization problem. However, the generator is impractical unless extreme security is needed. The Blum-Micali algorithm [6] presents also an unconditional security proof based on the difficulty of the discrete logarithm problem, but is also ineffective.

One interesting way to design pseudo-random generators can be found in chaos theory [8, 9, 10]. Indeed, chaotic systems are characterized by their high sensitivity to initial parameters and some properties like ergodicity, mixing property and high complexity [8, 11]. A secret parameter should be sensitive enough to ensure the so-called avalanche property. A small deviation in the initial conditions should cause a large modification in the output, that makes chaotic systems very attractive for pseudo-random number generation. These generators commonly use chaotic logistic maps and produce large pseudo-random sequences. For a high security level, it is necessary to combine several logistics maps, in order to increase the complexity of the cryptosystem. But, this is not always sufficient, because a rigorous analysis is more appropriate to evaluate the randomness level and the global security of the generator.

In this paper, a new PRBG combining three chaotic logistic maps is presented. It provides a significant improvement on security and performance, of the generator proposed by Patidar et al. [12]. The proposed algorithm uses the binary64 floating-point arithmetic and produces at each

iteration a block of 32 random bits. The pseudo-random sequences passed successfully the various statistical tests related to the randomness and correlation. The assets of the PRBG are: high sensitivity to initial seeds, high level of randomness and fast execution time. The paper is structured as follows, in Sec. 2 the used chaotic logistic map and the description of the Patidar's algorithm are given. Section 3, presents a detailed description of our algorithm and a brief discussion about the floating-point representation. The statistical analysis is given in Sec. 4. The security aspect of the PRBG is discussed in Sec. 5, before concluding.

2 Background

2.1 The chaotic logistic map

Frequently used in chaos theory as well as in chaos-based cryptosystems, the form of the logistic map is given by:

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

with β between 3.57 and 4.0 [13]. Its chaotic behavior has been widely studied and several generators have already used such logistic map for generating pseudo-random numbers [14, 15, 16, 17]. To avoid non-chaotic behaviour (island of stability, oscillations, ...), the value of β should be near 4.0, which corresponds to a highly chaotic behaviour. The logistic map is used under the iterative form:

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

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

2.2 About the algorithm of Patidar

Patidar et al. [12] have proposed a PRBG based on two logistic maps. The algorithm starts from random independent initial seeds X_0, Y_0, belonging to $]0, 1[$. The chosen value of β is 4 and the two logistic maps are given by:

$$X_{n+1} = 4X_n(1 - X_n), \forall n \geq 0, \tag{3}$$
$$Y_{n+1} = 4Y_n(1 - Y_n), \forall n \geq 0. \tag{4}$$

The main idea of the algorithm is very simple and consists to compare the outputs of both the logistic maps in the following way:

$$g(X_{n+1}, Y_{n+1}) = \begin{cases} 1 & if\ X_{n+1} > Y_{n+1} \\ 0 & if\ X_{n+1} \leq Y_{n+1} \end{cases}$$

Even if the idea is interesting, the algorithm presents several weaknesses:

1. Only one bit is generated after each iteration, that corresponds to a very low throughput according to the relevance of the logistic maps.
2. The sequences produced with nearby seed values are extremely correlated.
3. The seed space has a much lower entropy than 128, due to the existing correlation between the pseudo-random sequences. Therefore, the generator presents weak or degenerate seeds.
4. At a given iteration n, in the case of eventual collision between X_{n+1} and Y_{n+1} (which is possible), the output bit will always be 0 until the end of the output sequence.

The algorithm proposed in this paper also combines several chaotic logistic maps, but is designed to avoid all those weaknesses and then ensure a better security.

3 The proposed PRBG

3.1 Floating-point representation

As we know, digital computers use binary digits to represent numbers. In the case of real numbers, there are two representation formats: fixed-point and floating-point formats. To represent integers or real numbers with a fixed precision, it is more suitable to adopt the first format. The second format can support a much wider range of values. Nowadays, the floating point arithmetic is standardized by IEEE/ANSI [18]. Two different floating-point formats are defined: single precision (binary32) and double precision (binary64). In this paper, we only focus on binary64 floating-point format, which is generally used to achieve a higher simulation precision for the study of chaotic systems.
Binary64 has two infinities, two kinds of NaN (i.e. Not a Number) and the set of finite numbers. Each finite number is described by three fields: s a sign represented on one bit (1 indicating negative), e a biased exponent represented on 11 bits and m a mantissa represented on 52 bits (see Figure 1). The bits of the mantissa can be divided into two blocks of 20 bits and 32 bits and the treatment is applied on the block of 32 bits (mantissa1).

3.2 Description of the algorithm

As in the paper of Patidar et al., our algorithm uses the same type of chaotic logistic map given by Eq. 1. In our case, the value of β is fixed to 3.9999 that corresponds to a highly chaotic case [19, 20]. Indeed, the Lyapunov exponent [21, 22] measures the chaotic behavior of a function and the corresponding Lyapunov exponent of the logistic map for $\beta = 3.9999$ is 0.69 very close to its maximum which is 0.59. The chaotic logistic map is used under the iterative form:

$$X_{n+1} = 3.9999X_n(1 - X_n), \forall n \geq 0, \tag{5}$$

where the starting seed X_0 is a real number that belongs to $]0, 1[$. All the computed elements X_n are also real numbers

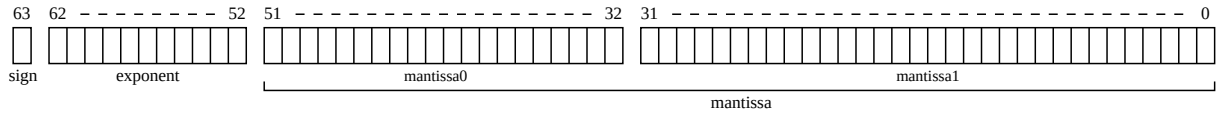

Figure 1: Floating-point representation in binary64 format.

in $]0, 1[$.

Our algorithm takes into account the various weaknesses of the algorithm proposed by Patidar et al. Thus, to have a large space of output sequences, three logistic maps are used during the generation process. The same value of β is used for each one and the corresponding equations are:

$$X_{n+1} = 3.9999X_n(1 - X_n), \forall n \geq 0, \quad (6)$$
$$Y_{n+1} = 3.9999Y_n(1 - Y_n), \forall n \geq 0, \quad (7)$$
$$Z_{n+1} = 3.9999Z_n(1 - Z_n), \forall n \geq 0. \quad (8)$$

For each computed value X_n, Y_n and Z_n, a binary64 floating-point representation is used as shown in Figure 1. The algorithmic principle is simple and consists at each iteration, to apply a xor operation on the 32 bits of mantissa1 of the three output elements X_n, Y_n and Z_n. Thus, the algorithm allows to produce 32 random bits per iteration and therefore increase the throughput of the generator. The operating principle of the algorithm is shown in Figure 2. As one can see, the seeds from which the generation process starts are X_k, Y_k and Z_k. Indeed, for nearby seed values, the elements X_n, Y_n and Z_n are almost identical in the first rounds. Thus, to completely decorrelate the beginning of the pseudo-random sequences, it is necessary to start the generation only at the kth iteration. The number of preliminary rounds k and the way to choose the initial seeds are presented in Sec. 3.3. The implementation of the algorithm in C language is simple: just include the file $ieee754.h$ and use the defined functions for extracting the bits of mantissa1 for each computed element X_n, Y_n and Z_n.

3.3 The choice of initial parameters

3.3.1 Initial seed selection

To improve the randomness quality of the generated sequences, the choice of the initial seed values should not be neglected. The coefficient values of the elements X_n, Y_n and Z_n, belong to $]0, 1[$. Due to symmetric structure of the logistic map, it is necessary to choose the starting seeds in one of the two half-intervals (here $]0, 2^{-1}[$) to avoid similar trajectories. In binary64 floating-point format, the computed term $(1 - X)$ is equal to 1.0 for any X in $]0, 2^{-53}[$, then for a seed value in $]0, 2^{-53}[$, the computed value of Eq. 2 is equivalent to βX_n. To avoid such problem, initial seed values must be chosen in $]2^{-53}, 2^{-1}[$.
The three initial seeds must be different, then the difference $\delta_{[2]}$ between the values should be representable in binary64. The value of $\delta_{[2]}$ is in the worst case 2^{-53}, which corresponds to $\log_{10}(2^{53})$ (≈ 15.955) decimal digits. To have a significant difference we choose $\delta_{[10]} = 10^{-15}$, which corresponds to $\delta_{[2]} = 2^{-49.8289}$. Thus, to avoid identical trajectories, the difference between each initial seed should be at least $\delta_{[2]} = 2^{-49.8289}$.

3.3.2 Number of preliminary rounds

In the case where the values of initial seeds (X_0, Y_0 and Z_0) are very close, the beginnings of chaotic trajectories are almost similar. To avoid such problem, it is necessary to apply some preliminary rounds before starting to produce the random bits. Thus, it is necessary to see at which number of iterations, the difference $\delta_{[2]}$ begins to be propagated. We consider that the initial seed is $X_0 = \delta_{[2]} = 2^{-49.8289}$, and the obtained trajectory with the Eq. 5 is shown in Figure 3.

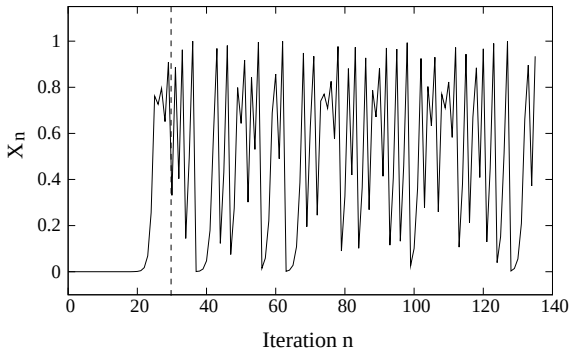

Figure 3: Trajectory of the chaotic logistic map given in Eq. 5, for $n = 135$ and $X_0 = 2^{-49.8289}$.

One can see that, the trajectory starts to oscillate almost from the 30th iteration. Thus, the generation of random bits will begin from the iteration 30. That allows to decorrelate the outputs of the PRBG, and then increase the sensitivity related to the initial seeds.

4 Statistical analysis

The quality of the output sequences produced by any PRBG is the crucial element. Indeed, the sequences should present individually a high level of randomness and be decorrelated with each other, whatever the initial seed values. Therefore, a statistical analysis should be carefully conducted to prove the quality of the pseudo-random sequences.

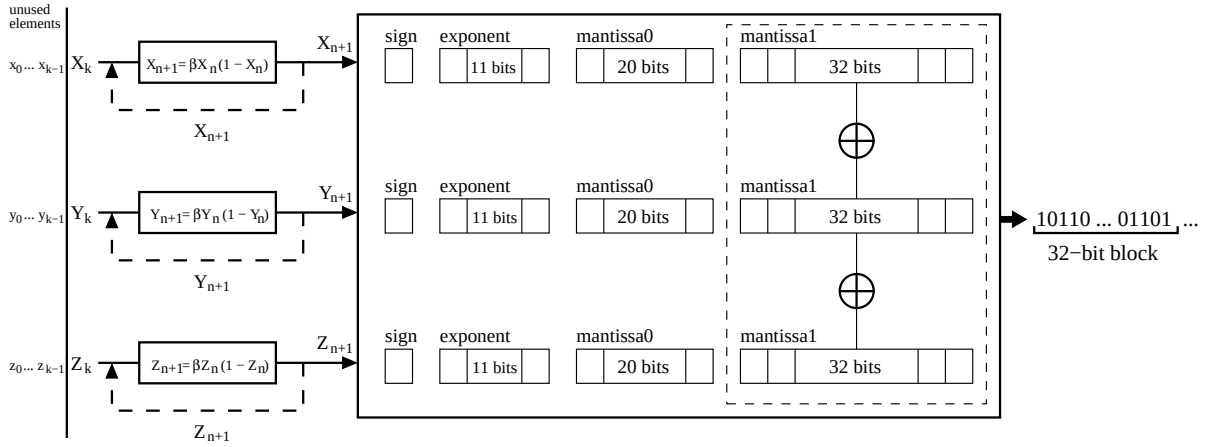

Figure 2: The operating principle of the proposed PRBG.

4.1 Randomness evaluation

The analysis consists in evaluating the randomness level of the sequences generated by the PRBG. In the literature, various statistical tests exist for analysing the randomness level of sequences. The NIST (National Institute of Standards and Technology of the U.S. Government) proposes a battery of tests that can be applied on the binary sequences [23]. One can also find other known libraries such as TestU01 [24] or the DieHARD suites [25]. Here, the sequences are evaluated through statistical tests suite NIST. Such suite consists in a statistical package of fifteen tests developed to quantify and to assess the randomness of binary sequences, produced by pseudo-random number generators. Here, we define three approaches for testing the randomness level of sequences. Let N be the total number of generated sequences and the binary size of each sequence is $M = 32 \times B$, with B the number of 32-bit blocs. The three approaches are:

1. **APP-1** (individual sequences): the produced sequences are individually tested and the results are given as ratio of success relatively to a threshold determined from the total number (N) of tested sequences. Such approach indicates the global randomness level of the tested sequences.

2. **APP-2** (concatenated sequence): all the individual sequences are concatenated to form a new single sequence. The randomness level of the constructed sequence is analysed through the NIST tests. The constructed sequence should pass the tests whether the original sequences are truly decorrelated and random.

3. **APP-3** (resulting sequences): all the sequences are superimposed on each other (forming a matrix), and new sequences are constructed from columns. Thus, B resulting sequences of binary size $32 \times N$ are constructed, by collecting for each position $1 \leq j \leq B$, the 32-bit bloc of each sequence. If the original sequences are really random, the resulting sequences should also be random (with B as large as N) and then pass the NIST tests. Such approach is very interesting, in the case of generating sequences by nearby seed values, and allows to detect some hidden linear structures between the original sequences.

These approaches are used to analyse a subset of generated sequences. In the case of very distant initial seed values, the corresponding chaotic trajectories are different, and allow to produce good pseudo-random sequences. The worst case occurs when closed seed values are used, because that can lead to highly correlated output sequences. That is why, the analysis is achieved on sequences generated from nearby initial seed values. Here, a subset of $N = 16000$ pseudo-random sequences is produced, where the binary size of each sequence is 32000 (i.e. $B = 1000$). We choose arbitrarily, one starting seed value $X_0 = 0.24834264038461704925$, and then $Y_0 = X_0 + \delta_{[2]}$ and $Z_0 = Y_0 + \delta_{[2]}$, with $\delta_{[2]} = 2^{-49.8289}$. These three seeds allow to generate one pseudo-random sequence. The other sequences, are generated from X_0, Y_0 and by incrementing of $\delta_{[2]}$ the last seed value Z_0 in a simple loop.
In the case of Patidar's algorithm, only two initial seeds are needed to produce a pseudo-random sequence. Here, the first two seeds values are given by: $X'_0 = Y_0$ and $Y'_0 = Z_0$. For generating the other sequences, the same strategy is applied and consists to make a loop by incrementing of $\delta_{[2]}$ the seed Y'_0. It should be noted that, for a better comparison, the same coefficient β (i.e. 3.9999) is used for the logistic map in the Patidar's algorithm.
The results of NIST tests obtained for the two algorithms are presented in Table 4.1 and Table 4.1. For approach **APP-1** (resp. **APP-3**), the acceptable proportion should lie above 98.76% (resp. 98.00%) and does not concern the tests *Random Excursions-(Variant)*. For **APP-2**, a sequence passes a statistical test for $p_{value} \geq 0.01$ and fails otherwise. For the tests *Non-Overlapping* and *Random Excursions-(Variant)*, only the smallest percentage of all sub-tests is given. For individual sequences, the *Universal*

test is not applicable due to the size of initial sequences. One can remark that, for the proposed PRBG, all the tested sequences pass successfully the NIST tests. These results show clearly the quality of the tested sequences. For Patidar's algorithm, individually, the sequences are not enough random, because there are many tests that are not successful, for example: *Runs, Overlapping* or even *Serial* tests. The results of approaches **APP-2** and **APP-3** show that, the tested sequences are extremely correlated. One should know that, in the article of Patidar et al., each sequence is produced from a randomly chosen initial seed belonging to $]0, 1[$, then the seeds were too different from each other. That is why this problem has not been detected.

4.2 Correlation evaluation

A part of the correlation evaluation has already been done by applying the NIST tests (**APP-2** and **APP-3**). Here, two additional methods are used to analyse the correlation between the pseudo-random sequences. Firstly, the correlation between sequences is evaluated globally by computing the Pearson's correlation coefficient [26] and secondly, by using the Hamming distance.

4.2.1 Pearson's correlation coefficient

The analyse consists to compute the Pearson's correlation coefficient between each pair of sequences, and to present the distribution of the values through a histogram. Consider a pair of sequences such as: $S_1 = [x_0, \ldots, x_{B-1}]$ and $S_2 = [y_0, \ldots, y_{B-1}]$. Therefore, the corresponding correlation coefficient is:

$$C_{S_1,S_2} = \frac{\sum_{i=0}^{B-1} (x_i - \overline{x}) \cdot (y_i - \overline{y})}{\left[\sum_{i=0}^{B-1} (x_i - \overline{x})^2\right]^{1/2} \cdot \left[\sum_{i=0}^{B-1} (y_i - \overline{y})^2\right]^{1/2}}, \quad (9)$$

where x_i and y_i are 32-bit integers, $\overline{x} = \sum_{i=0}^{B-1} x_i/B$ and $\overline{y} = \sum_{i=0}^{B-1} y_i/B$, the mean values of S_1 and S_2, respectively. Two uncorrelated sequences are characterized by $C_{S_1,S_2} = 0$. The closer the value of C_{S_1,S_2} is to ± 1, the stronger the correlation between the two sequences. In the case of two independent sequences, the value of C_{S_1,S_2} is equal to 0. Here we use the same subsets of 16000 sequences, and the coefficients C_{S_1,S_2} are computed. For the two algorithms, the histograms are shown in Figure 4. For the proposed PRBG, around 99.56% of the coefficients have an absolute value smaller than 0.09, then only a small correlation is detected. In the case of Patidar's PRBG, around 99.26% of the coefficients have an absolute value greater than 0.33, that means the sequences are highly correlated.

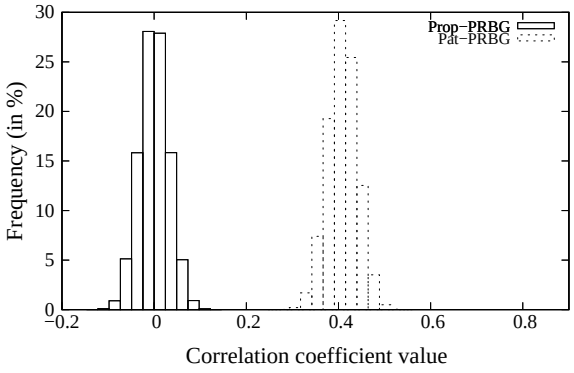

Figure 4: Histogram of Pearson's correlation coefficient values on interval $[-0.1, 0.1]$ (resp. $[-0.5, 0.5]$), for the proposed (resp. Patidar's) PRBG.

4.2.2 Hamming distance

Another type of correlation based on the bits of produced pseudo-random sequences is analysed. Given two binary sequences $S = [s_0, \ldots, s_{M-1}]$ and $S' = [s'_0, \ldots, s'_{M-1}]$ of same length (M), the Hamming distance is the number of positions where they differ. The distance is given as:

$$d(S, S') = \sum_{j=0}^{M-1} (s_j \oplus s'_j). \quad (10)$$

For truly random binary sequences, the value of $d(S, S')$ should be around $M/2$, that corresponds to the proportion 0.50. This distance is computed between each pair of generated sequences ($N = 16000$), and all values are represented through a histogram. For the two algorithms, the histograms are shown in Figure 5. One can see that for our

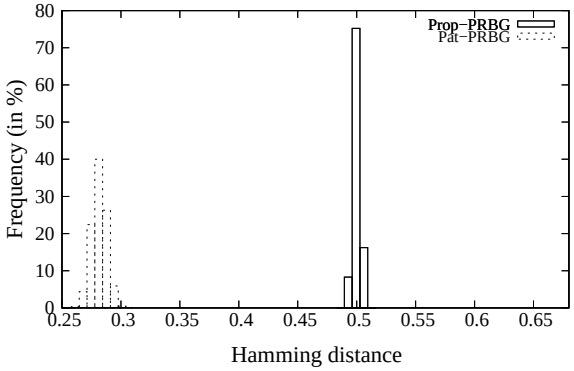

Figure 5: Histogram of Hamming distances on interval $[0.48, 0.52]$ (resp. $[0.25, 0.30]$), computed between each pair of sequences for the proposed (resp. Patidar's) PRBG.

algorithm, all the proportions of computed Hamming distances are around the mid-value 0.50 and almost 99.95% of the coefficients belong to $]0.49, 0.51[$. In the second case, the values are around 0.28, and near 99.83% of the coefficients belong to $]0.26, 0.30[$. The results show that, the

Test Name	Proposed PRBG					
	APP-1		**APP-2**		**APP-3**	
	r_1 (in %)	Result	p_{value}	Result	r_2 (in %)	Result
Frequency	99.16	Success	0.378240	Success	99.50	Success
Block-Frequency	99.10	Success	0.905858	Success	98.80	Success
Cumulative Sums (1)	99.17	Success	0.447272	Success	99.40	Success
Cumulative Sums (2)	99.12	Success	0.259837	Success	99.30	Success
Runs	98.90	Success	0.654035	Success	99.20	Success
Longest Run	98.90	Success	0.717020	Success	98.70	Success
Rank	98.86	Success	0.239335	Success	99.00	Success
FFT	98.76	Success	0.485387	Success	99.00	Success
Non-Overlapping	99.30	Success	0.012842	Success	98.20	Success
Overlapping	99.00	Success	0.935098	Success	98.80	Success
Universal	-	-	0.196700	Success	98.60	Success
Approximate Entropy	98.91	Success	0.199988	Success	99.00	Success
Random Excursions	97.56	Success	0.012412	Success	98.60	Success
Random Ex-Variant	97.56	Success	0.024851	Success	97.62	Success
Serial (1)	98.92	Success	0.379823	Success	99.30	Success
Serial (2)	99.05	Success	0.856303	Success	99.20	Success
Linear Complexity	98.84	Success	0.098641	Success	99.00	Success

Table 1: The results of NIST tests for the proposed PRBG on the 16000 sequences. The ratio r_1 (resp. r_2) of p_{value} passing the tests are given for **APP-1** (resp. **APP-3**). For the approach **APP-2** the corresponding p_{value} is given.

Test Name	Patidar's PRBG					
	APP-1		**APP-2**		**APP-3**	
	r_1 (in %)	Result	p_{value}	Result	r_2 (in %)	Result
Frequency	99.84	Success	0.000000	Failure	02.80	Failure
Block-Frequency	99.99	Success	0.989313	Success	16.00	Failure
Cumulative Sums (1)	99.80	Success	0.000000	Failure	03.10	Failure
Cumulative Sums (2)	99.74	Success	0.000000	Failure	03.00	Failure
Runs	29.25	Failure	-	-	00.60	Failure
Longest Run	00.00	Failure	0.000000	Failure	00.00	Failure
Rank	98.83	Success	0.442618	Success	04.00	Failure
FFT	98.70	Success	0.000000	Failure	00.00	Failure
Non-Overlapping	75.96	Failure	0.000000	Failure	71.80	Failure
Overlapping	00.00	Failure	0.000000	Failure	99.00	Success
Universal	-	-	0.000000	Failure	00.40	Failure
Approximate Entropy	00.00	Failure	0.000000	Failure	00.00	Failure
Random Excursions	98.55	Success	-	-	50.00	Failure
Random Ex-Variant	97.82	Success	-	-	93.75	Success
Serial (1)	95.10	Failure	0.000000	Failure	00.00	Failure
Serial (2)	98.99	Success	0.000000	Failure	00.00	Failure
Linear Complexity	98.81	Success	0.283356	Success	99.00	Success

Table 2: The results of NIST tests for the PRBG of Patidar et al., on the 16000 sequences. The ratio r_1 (resp. r_2) of p_{value} passing the tests are presented for the approach **APP-1** (resp. **APP-3**). For **APP-2** the corresponding p_{value} is given.

sequences are correlated for the Patidar's PRBG, then the algorithm is not enough sensitive to initial seed values. For more sensitivity, one must choose very different seed values, which reduces considerably the key space and then the security of the PRBG.

4.3 Seed sensitivity

A small deviation from the initial seeds, should cause a large variation in the output sequences. Actually, in the NIST tests (**APP-2** and **APP-3**, Sec. 4.1) and the correlation evaluation (Sec. 4.2), the sensitivity related to the seeds was indirectly analysed. To make an additional analysis, a large size of pseudo-random sequences is considered. Here, the number of 32-bit blocs is $B = 10000000$, then the binary size $M = 320000000$. A pseudo-random sequence ($Seq1$) is produced using the seed values: $X_0 = 0.32164872553014364784, Y_0 = X_0 + \delta_{[2]}$ and $Z_0 = Y_0 + \delta_{[2]}$. Two others sequences ($Seq2$ and $Seq3$) are produced by adding the value of $\delta_{[2]}$ on the last seed value Z_0. Between each pair of sequences, the correlation analysis is done by computing the linear correlation coefficient of Pearson, the correlation coefficient of Kendall [27] and the Hamming distance. The same analysis is applied on the Patidar's algorithm with the starting seeds $X'_0 = Y_0$ and $Y'_0 = Z_0$. The results are given in Table 4.3 and show that: our algorithm is highly sensitive to initial seeds, whereas in the case of the Patidar's algorithm, the sensitivity is extremely weak.

4.4 Speed analysis

Another important aspect for any PRBG is the execution time of the algorithm. Indeed, in real-time applications, the temporal constraint about the performance of a process is as considerable as the final results of the process. The speed evaluation 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 results are presented in Table 4.4 and one can see that, with no optimization option (-O0), the proposed algorithm enables to produce around 2.62 Gbits per second. However, with the classical optimization option (-O1), the throughput is approximately 80 Gbits per second.

PRBG	Speed (Gbits/s)	
	-O0	-O1
Proposed	2.62	80.00
Patidar's	0.06	1.18

Table 4: Comparison of speed between the two algorithms by using the options "-O0" and "-O1".

The Table 4.4 presents the approximative throughput of some known pseudo-random number generators. One can remark that, the throughput of our generator is almost in the same order than CURAND.

Generator	Speed (Gbits/s)	
	GT120 GPU (4 cores)	GTX260 GPU (27 cores)
MTGP11213	41.88	340.42
CURAND	96.97	533.33

Table 5: The approximative throughput in Gbits/s for MTGP11213 (*Mersenne Twister for Graphic Processor*) and CURAND (*NVIDIA CUDA Random Number Generation library*).

5 Security analysis

Some points related to the security of the PRBG are discussed here, such as: the size of the seed space, the period length of the logistic map and some basic-known attacks (brute-force attack and differential attack).

5.1 Seed space

Given today's computational resources, a seed space of size smaller than 2^{128} is not secure enough. A robust PRBG should have a large key space, to allow a large choice for the pseudo-random number generation. In order to enlarge the key space, three chaotic logistic maps are used during the generation process. Each logistic map needs to be initialized with a seed corresponding to a binary64 floating-point number, selected from $]2^{-53}, 2^{-1}[$. Knowing that the difference between each seed value is $2^{-49.8289}$, this allows to have $2^{48.8289}$ possible choices of initial seeds. Thus, the total number of choices for the three initial seeds is:

$$2^{48.8289} \times [2^{48.8289} - 1] \times [2^{48.8289} - 2],$$

or about $2^{146.50}$. In the case of Patidar's algorithm, the entropy of the seed space is much smaller than 128. Indeed, the algorithm uses only two logistic maps and possesses a weak sensitivity, that requires to choose very distant seeds to produce secure outputs.

5.2 Period length of the logistic map

The period length is a fundamental indicator of any PRBG. A generator should have a reasonably long period before its output sequence repeats itself, for avoiding attacks. The length of the period will indicate the maximal secure size for the producible pseudo-random sequences. The idea is to determine the cycle formed by each chaotic trajectory according to the different starting seed values. In a period-p cycle, $X_k = F^p(X_k)$ for some X_k, where F^p is the pth iterate of F, *e.g.*, $F^3(X) = F(F(F(X)))$ for $p = 3$.
The GNU MPFR library [29] is used to vary the bits of the mantissa for analysing the cycles of the logistic map. The Figure 6 shows the length of cycles, when the bits of mantissa vary between 10 and 25. In this case, the logistic map has very small cycle lengths. The Figure 7 indicates at the the same time the corresponding occurrences.

PRBG	Tests	$Seq1/Seq2$	$Seq1/Seq3$	$Seq2/Seq3$
Proposed algorithm	Pearson Corr. Coef.	-0.000460	0.000210	-0.000536
	Kendall Corr. Coef.	-0.000127	-0.000377	-0.000201
	Hamming Distance	0.499985	0.499986	0.500010
Patidar's algorithm	Pearson Corr. Coef.	0.328897	0.328990	0.328856
	Kendall Corr. Coef.	0.245210	0.242712	0.242441
	Hamming Distance	0.333467	0.333379	0.333313

Table 3: Comparison using the Pearson's and Kendall's correlation coefficients, and also Hamming distance (in proportion) between each pair of sequences (Seq_1, Seq_2 and Seq_3), produced from slightly different seeds.

In binary32 format, the obtained smallest (resp. longest) cycle length is equal to 1 (resp. 3055). Also, the logistic map has numerous pathological seeds (corresponding in minimum cycles of length 1) and globally the cycle lengths are too small. Therefore, the binary32 format is not appropriate and must be avoided, when implementing a PRBG with such logistic map. Besides, this result is consistent with that published by Persohn and Povinelli [30].
For binary64 format, the cycle lengths are much longer. Due to the large size of the binary format, it is difficult to analyse all the corresponding trajectories. Only a reasonable set of randomly chosen seeds is considered. The length of the smallest cycle is 2169558 ($\approx 2^{21.04}$), while for the longest cycle is 40037583 ($\approx 2^{25.25}$). Here, the computed cycle lengths are in the same order as those given in [28], and no pathological seed was found. This format was not studied by Persohn and Povinelli, and it is a format that benefits to the logistic map. Also, the used parameter λ (equal to 4) does not provide the best chaotic behavior. In our case, combining three chaotic logistic maps allows to increase the length of the global resulting cycle, which is given by the LCM of the three cycle lengths. Also, the value of β (here 3.9999) plays a crucial role, because it provides a better chaotic behavior of the logistic map. However, for a maximum security level, it might be better to limit the length of sequences to the smallest cycle length. The best way to avoid the problem of short period and use efficiently this PRBG, is to generate pseudo-random bit sequences of only small sizes. However in case of need, long sequences can be constructed by concatenating several ones.

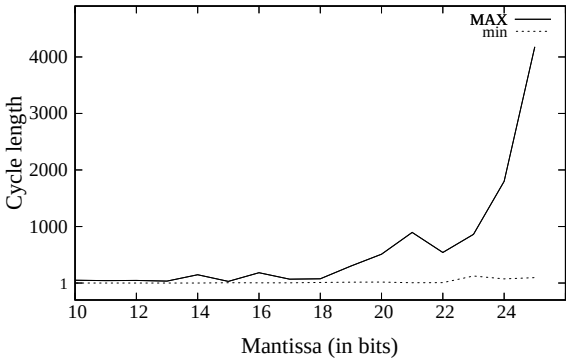

Figure 6: The curve "MAX" (resp. "min") shows the length of longest (resp. smallest) cycles, when varying mantissa bits between 10 and 25.

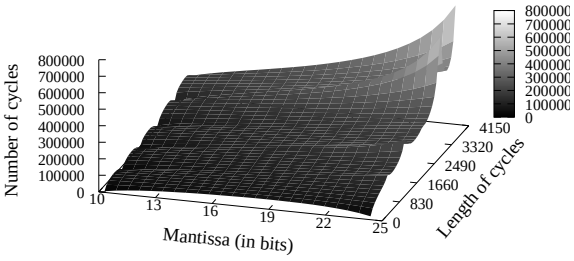

Figure 7: Representation of the length of cycles and their total numbers, when the bits of the mantissa vary between 10 and 25.

5.3 Brute-force attack

In theory, a brute-force attack [8] is an attack that can be used against any kind of PRBG. Such attack is usually utilized, when it is not easy (or possible) to detect any weakness in the algorithm, that would make the task easier. The strategy of the attack is simple and consists to check systematically all possible keys until the original key is found. On average, just half of the size of key space needs to be tested to find the initial seeds. A large key space allows to frustrate such kind of attack. Nowadays a key space of size larger than 2^{128} is computationally secure enough to resist to such attack. The proposed PRBG has a key space of size $2^{146.50}$ and we consider that, such attack can not succeed on the generator. In the case of Patidar's PRBG, the entropy of the key space is logically much smaller than 128. Thus, such attack can be envisaged for breaking the generator.

5.4 Differential attack

As a chosen-plaintext attack, the principle of such technique of cryptanalysis is to analyse the effect of a small difference in input pairs (i.e. seeds), on the difference of corresponding output pairs (i.e. sequences) [31]. This strategy allows to get the most probable key, that was used to generate the pseudo-random sequence. The initial differ-

ence may be in the form of a subtraction modulus or a xor difference and the diffusion aspect is measured by a differential probability. The proposed algorithm is designed to avoid such attack. Indeed, the initial seeds are chosen in the interval $]2^{-53}, 2^{-1}[$ and the bit-generation starts only at the 30th iteration. The results of the statistical analysis (Sec. 4) showed also that, even with a small difference on the seeds, the pseudo-random sequences are highly decorrelated from each other. Thus, we consider that the proposed PRBG should resist to the differential cryptanalysis. On the other side, the attack can be possible on the Patidar's PRBG, because the algorithm is not sensitive enough to the initial seed values.

6 Conclusion

A chaos-based PRBG, combining three chaotic logistic maps under binary64 floating-point arithmetic was presented. It provides significant improvements of an existing generator. The principle consists at each iteration, to apply a bitwise xor operator on the 32 least significant bits of mantissa, from the computed elements of logistic maps. The algorithm is fast and allows to produce pseudo-random sequences formed of 32-bit blocks. The assets of the PRBG are: the simplicity of implementation, a high randomness level for outputs, a high sensitivity related to the initial seeds and a fast execution time, allowing to use the algorithm even in real-time applications.

References

[1] F. Sun and S. Liu (2009). Cryptographic pseudo-random sequence from the spatial chaotic map. *Chaos Solitons & Fractals*, Vol. 41, No. 5, pp. 2216–2219.

[2] M. Matsumoto and T. Nishimura (1986). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Vol. 8, No. 1, pp. 3–30.

[3] J. Eichenauer and J. Lehn (1986). A non-linear congruential pseudo random number generator. *Statistische Hefte*, Vol. 27, No. 1, pp. 315–326.

[4] A.K. Varshney, S.K. Sharma and R. Singh (2012). A study on the effect of shifting on LFSR PRNG. *International Journal of Engineering*, Vol. 1, No. 5, pp. 1–7.

[5] M. Tomassini, M. Sipper, M. Zolla and M. Perrenoud (1999). Generating high-quality random numbers in parallel by cellular automata. *Future Generation Computer Systems*, Vol. 16, No. 2, pp. 291–305.

[6] M. Blum and S. Micali (1984). How to generate cryptographically strong sequences of pseudorandom bits. *SIAM journal on Computing*, Vol. 13, No. 4, pp. 850–864.

[7] L. Blum, M. Blum and M. Shub (1986). A simple unpredictable pseudo-random number generator. *SIAM journal on Computing*, Vol. 15, No. 2, pp. 364–383.

[8] G. Álvarez and S. Li (2006). Some Basic Cryptographic Requirements for Chaos-Based Cryptosystems. *International Journal of Bifurcation and Chaos*, Vol. 16, No. 8, pp. 2129–2151.

[9] H.P. Hu, L.F. Liu and N.D. Ding (2012). Pseudorandom sequence generator based on Chen chaotic system. *Computer Physics Communications*, Vol. 184, No. 3, pp. 765–768.

[10] M. François, T. Grosges, D. Barchiesi and R. Erra (2013). A new pseudo-random number generator based on two chaotic maps. *Informatica*, Vol. 24, No. 2, pp. 181–197.

[11] J.M. Bahi, C. Guyeux and Q. Wang. A pseudo random numbers generator based on chaotic iterations. application to watermarking. *In WISM 2010, International Conference on Web Information Systems and Mining*, Vol. 6318 of LNCS, pp. 202–211, Sanya, China, October 2010.

[12] V. Patidar, K.K. Sud and N.K. Pareek (2009). A Pseudo Random Bit Generator Based on Chaotic Logistic Map and its Statistical Testing. *Informatica*, Vol. 33, No. 4, pp. 441–452.

[13] R. Bose and A. Banerjee. Implementing Symmetric Cryptography Using Chaos Functions, *in: Proceedings of the 7th International Conference on Advanced Computing and Communications*, pp. 318–321, 1999.

[14] M.S. Baptista (1998). Cryptography with chaos. *Physics Letters A*, Vol. 240, No. 1, pp. 50–54.

[15] S. Cecen, R.M. Demirer and C. Bayrak (2009). A new hybrid nonlinear congruential number generator based on higher functional power of logistic maps. *Chaos Solitons & Fractals*, Vol. 42, No. 2, pp. 847–853.

[16] S. Xuan, G. Zhang and Y. Liao. Chaos-based true random number generator using image. *in: IEEE International Conference on Computer Science and Service System (CSSS), Nanjing, China*, pp. 2145–2147, 2011.

[17] M. François, T. Grosges, D. Barchiesi and R. Erra (2014). Pseudo-random number generator based on mixing of three chaotic maps. *Communications in Nonlinear Science and Numerical Simulation*, Vol. 19, No. 4, pp. 887–895.

[18] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pp. 1–58.

[19] N.K. Pareek, V. Patidar and K.K. Sud (2006). Image encryption using chaotic logistic map. *Image and Vision Computing*, Vol. 24, No. 9, pp. 926–934.

[20] M. François, T. Grosges, D. Barchiesi and R. Erra (2012). Image Encryption Algorithm Based on a Chaotic Iterative Process. *Applied Mathematics*, Vol. 3, No. 12, pp. 1910–1920.

[21] A. Wolf, J.B. Swift, H.L. Swinney, J.A. Vastano (1985). Determining Lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, Vol. 16, No. 3, pp. 285–317.

[22] E. Aurell, G. Boffetta, A. Crisanti, G. Paladin, A. Vulpiani (1997). Predictability in the large: an extension of the concept of Lyapunov exponent. *Journal of Physics A: Mathematical and General*, Vol. 30, No. 1, pp. 1–26.

[23] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray and S. Vo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. *NIST Special Publication Revision 1a 2010*.

[24] P. L'ecuyer and R. Simard (2007). TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, Vol. 33, No. 4, Article 22.

[25] G. Marsaglia (1996). Diehard: a battery of tests of randomness. http://stat.fsu.edu/geo/diehard.html

[26] J. Rodgers and W. Nicewander (1988). Thirteen ways to look at the correlation coefficient. *The American Statistician*, Vol. 42, No. 1, pp. 59–66.

[27] M.G. Kendall. Rank correlation methods. *Fourth ed., Griffin, London*, 1970.

[28] J. Keller and H. Hanno. Period lengths of chaotic pseudo-random number generators, *in: Proceedings of the Fourth IASTED International Conference on Communication, Network and Information Security*, pp. 7–11, 2007.

[29] The GNU MPFR library. http://www.mpfr.org

[30] K.J. Persohn and R.J. Povinelli (2012). Analyzing logistic map pseudorandom number generators for periodicity induced by finite precision floating-point representation. *Chaos Solitons & Fractals*, Vol. 45, No. 3, pp. 238–245.

[31] E. Biham and A. Shamir. Differential Cryptanalysis of the Data Encryption Standard. *Springer-Verlag*, 1993.