A Pseudo-Random Bit Generator Using Three Chaotic Logistic Maps

Michael FRANÇOIS *and David DEFOUR†

Univ. Perpignan Via Domitia, DALI F-66860, Perpignan, France
Univ. Montpellier II, LIRMM, UMR 5506, F-34095, Montpellier, France
CNRS, LIRMM, UMR 5506, F-34095, Montpellier, France

Dated: February 6, 2013

Abstract

A novel pseudo-random bit generator (PRBG) using three chaotic logistic maps is proposed. The algorithm generates at each iteration sequences of 32 bit-blocks by starting from randomly chosen initial seeds. The impact of relying on IEEE 754-2008 floating-point representation format for the generator is also taken into account. The performance of the generator is evaluated through various statistical analyses. The results show that the produced sequences possess high randomness statistical properties and good security level which make it suitable for cryptographic applications.

1 Introduction.

The generation of pseudo-random bits (or numbers) plays a critical role in a large number of applications such as statistical mechanics, numerical simulations, gaming industry, communication or cryptography [Sun, 2009]. 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. One interesting way to design such generators is connected to chaos theory [Alvarez, 2006]. That theory

[*] Electronic address: `michael.francois@univ-perp.fr`;

[†] Electronic address: `david.defour@univ-perp.fr`; Corresponding author

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 behaviour and high complexity [Alvarez, 2006]. 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 pseudo-random number generators. Moreover, during this last decade several pseudo-random number generators have been successfully developed [Guyeux, 2010, Zheng, 2008, Pareek, 2010, Patidar, 2009a, Orúe, 2010]. 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 block of 32 random bits at each iteration. An efficient process based on permutations among values produced by the three logistic maps and xor operation anyhilate the collision problem that may appear while using several logistic map based on the same function. 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 of floating-point arithmetic and the used chaotic logistic map is given in Sec. 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 Sec. 4. The global security analysis of the PRBG is achieved in Sec. 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 representations 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 [IEEE, 2008] is the current version of the technical standard used by hardware manufacturer to implement floating-point arithmetic. Among them, binary32 and binary64 which correspond to single and double precision format 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 this 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 figure 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 [Goldberg, 1991]

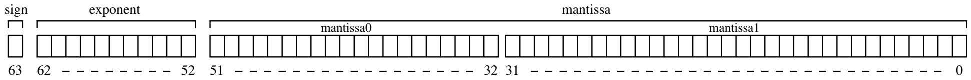

Figure 1: Floating-point representation in double precision format (64 bits).

2.2 The chaotic logistic map.

The PRBG uses the chaotic logistic map given by:

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

with λ between 3.57 and 4.0 [Bose, 1999]. This function have been widely studied [Weisstein, Eric W.] and several pseudo-random number generators have already used this logistic map [Bose, 1999, Baptista, 1998, Patidar, 2009a, Cecen, 2009, Xuan, 2011]. To avoid non-chaotic behavior (island of stability, oscillations, ...), the value of λ is fixed to 3.9999 which corresponds to a highly chaotic case [Pareek, 2006]. The chaotic logistic map is 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 the interval $]0, 1[$.

3 The proposed generator.

The main idea of the proposed PRBG is to combine several chaotic logistic maps and carefully arrange them in the same algorithm in order to increase the security level compared to others PRBG such as [Patidar, 2009a]. It generates a block of 32 random bits per iteration using the following three logistic maps :

$$
\begin{aligned}
X_{n+1} &= 3.9999X_n(1-X_n), \forall n \geq 0, && (3)\\
Y_{n+1} &= 3.9999Y_n(1-Y_n), \forall n \geq 0, && (4)\\
Z_{n+1} &= 3.9999Z_n(1-Z_n), \forall n \geq 0. && (5)
\end{aligned}
$$

For the three chaotic maps, the same value of λ is chosen to maintain its surjectivity in the same interval. For each computed value X_n, Y_n and Z_n, we used binary64 floating-point representation format as in Fig. 1.

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 starting seed values X_0, Y_0 and Z_0 are carefully chosen to avoid colision (see section 3.1).

2. Line 3-8; The results of the 30 first iterations of the three chaotic logistic maps are discared to have sufficient chaotic behavior among them (see section 3.2).

3. Line 9-50; Iterate N times, N being the length of the output sequence in blocks of 32 bits to:

   (a) Line 10-12; Compute the three logistic maps.

   (b) Line 13-21; For each logistic map, save the bits of mantissa0 and mantissa1 in two different variables.

   (c) Line 23-48; Start another loop of length 32 on the bits of mantissa1 and in each case, select one bit at a time using the value of mantissa0. For more security, the value of the mantissa0 of the element 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. The three selected bits are combined by a xor operator to give the output bit (Line 35-46). The selected bit is then permuted with the bit at the end of chain to not be used again. At the end of the last loop, a 32-bit bloc is generated.

3.1 Seeds selection.

The choice of the starting seed values should not be neglected. We are using a logistic map where input and ouput value belong to the interval $]0, 1[$. The main idea of this article is to use three identical logistic maps in order to increased the robustness of the generator. To preserve such robustness, we have to avoid collision that may occur with incorrect initial values that will leads to identical series of numbers.

To understand the collision mecanism, we have to understand how a difference δ between two computed value X_n and Y_n at a given iteration n will propage to the next iteration. Without loss of generality, we can assume that:

$$Y_n = X_n(1+\delta).$$

Out of equation 2 we know that:

$$\begin{aligned} X_{n+1} &= \lambda X_n(1-X_n), && (6)\\ Y_{n+1} &= \lambda Y_n(1-Y_n), && (7) \end{aligned}$$

which is equivalent to

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

By evaluating the partial derivative in δ of the previous equation, we can deduce that the smallest difference between X_{n+1} and Y_{n+1} is reached when

$$\delta = \frac{1-2X_n}{X_n},$$

which means that when $\delta = \frac{1}{X_n} - 2$, then at the next iteration we will have $Y_{n+1} = X_{n+1}$ corresponding to a collision. To avoid this case the selection of the three seeds has to be done in the interval $]0, 2^{-1}[$.

The difference at the next iteration is minimal when X_n is close to 2^{-1}. In this case, the difference between X_{n+1} and Y_{n+1} is :

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

and its limit as X_n approaches 2^{-1} is

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

In order to avoid collision this difference has to be representable in binary64, which means:

$$\frac{\lambda\delta^2}{4} > 2^{-53}.$$

By hyphothesis, we set $\lambda = 3.9999$, then δ has to be such that $\delta > 2^{-26}$ in order to avoid collision.

Lack of chaos may arises when dealing with floating-point arithmetic due to the effect of rounding error. 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 equation 2 is equivalent to λX_n. To avoid this problem, initial seeds have to be taken in the interval $]2^{-53}, 2^{-1}[$.

To sumarize, collisions are avoided by choosing for the first seed X_0 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 a and b such that :

- a and b are different and not equal to 0.
- $|a| > 2^{-26}$ and $|b| > 2^{-26}$.
- $Y_0 = X_0(1 + a)$ and $Z_0 = X_0(1 + b)$ belong to the interval $]2^{-53}, 2^{-1}[$.

3.2 Initial chaotic behaviour

We have plotted in figure 2 the obtained trajectory of the logistic map with a small seed ($X_0 = 10^{-15}$). One can observe that for the first iterations of the process, the trajectory is not chaotic. This is because, the initial difference between two successive values, spreads slowly toward the leading bit of the mantissa. Thus, to decorrelate the beginning of the output sequences among the three logistic maps, it is necessary to discard the first iterations before starting the generation.

We have exhaustively tested for each input values X_0 in the interval $]2^{-53}, 2^{-1}[$ and $Y_0 = X_0(1+2^{-25})$, the minimum number of iterations k which is necessary to reach the condition $\log_2\left(\frac{|X_k - Y_k|}{X_k}\right) > -1$. This condition corresponds to loose all the initial leading bits of correlation between X_0 and Y_0. We measured that on average 25.4 iterations are required. Therefore, to decorrelate the beginning of the considered sequences and increase the security level of the PRBG, we choose that the generation will start from the 31st iteration.

4 Statistical analysis.

For any secure PRBG, the output sequences must have a high level of randomness and be completely decorrelated 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.

This analysis consists in evaluating the randomness quality of the sequences produced by the algorithm. 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

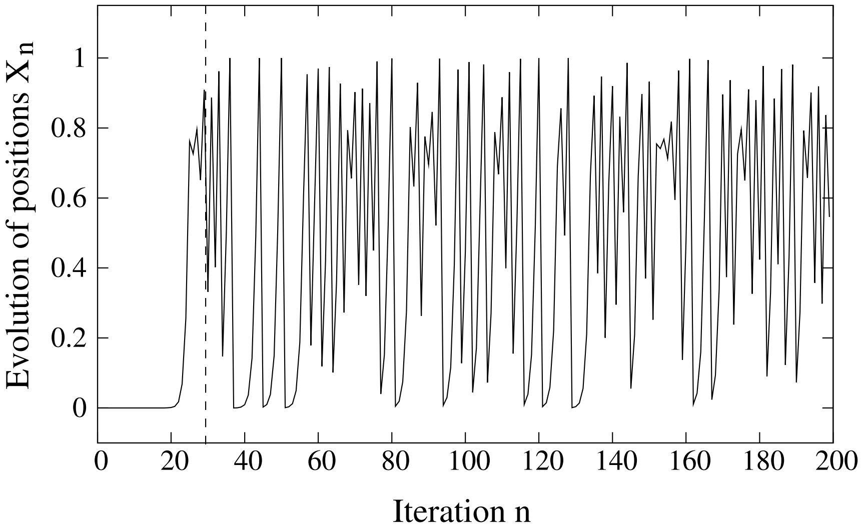

Figure 2: Trajectory of the chaotic logistic map given in Eq. 2 for $n = 200$ and $X_0 = 10^{-15}$.

generators [Rukhin, 2010]. For each statistical test, a set of p_{value} is produced and is 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 $(1 - 0.01) \pm 3\sqrt{0.01(1 - 0.01)/T}$ [Rukhin, 2010]. To analyse several aspects of the sequences, the NIST tests are applied on: individual sequences, 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 analysed with the NIST tests. In the case of truly decorrelated random sequences, the

concatenated sequence should also be random.

3. Resulting sequences: are the sequences obtained from the columns, if the tested 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 analyse such resulting sequences. If truly random sequences are superimposed on each other, the resulting sequences should also be random (with T as large as N). This approach is interesting especially for sequences generated with successive seed values and can show if there is some hidden linear structures between the original sequences.

4.2 Correlation evaluation.

The correlation evaluation is done in two different ways. Firstly, the correlation between generated sequences is analysed globally by computing the Pearson's correlation coefficient of each pair of sequences [Patidar, 2011]. Consider a pair of sequences given by: $S_1 = [x_0, \dots, x_{N-1}]$ and $S_2 = [y_0, \dots, y_{N-1}]$. Therefore, the corresponding correlation coefficient is:

$$C_{S_1,S_2} = \frac{\sum\limits_{i=0}^{N-1} (x_i - \overline{x}) \cdot (y_i - \overline{y})}{\left[\sum\limits_{i=0}^{N-1} (x_i - \overline{x})^2\right]^{1/2} \cdot \left[\sum\limits_{i=0}^{N-1} (y_i - \overline{y})^2\right]^{1/2}}, \tag{8}$$

where x_i and y_i are 32-bit integers, $\overline{x} = \sum\limits_{i=0}^{N-1} x_i/N$ and $\overline{y} = \sum\limits_{i=0}^{N-1} y_i/N$ 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 analysed. 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 Hamming distance is given by:

$$d(S_1^b, S_2^b) = \sum_{j=0}^{M-1} (x_j \oplus y_j), \tag{9}$$

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.

4.3.1 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. One can remark that for the two groups GRP1 and GRP2, all the tested sequences pass successfully the NIST tests. These results show clearly the quality of the produced sequences from successive seed values.

4.3.2 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. 3. 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, for each pair of the 15000 produced sequences, the proportion of places where the bits

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

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

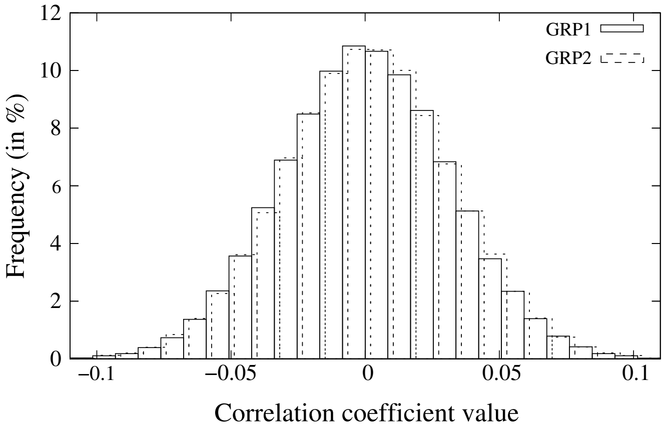

Figure 3: Histogram of Pearson's correlation coefficient values on interval $[-0.1, 0.1]$ for the group GRP1 (resp. GRP2).

differ is computed. The histograms are presented in Fig. 4. 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, because of the entropy of seed values. Such analysis provides another indication about the decorrelation between the generated sequences.

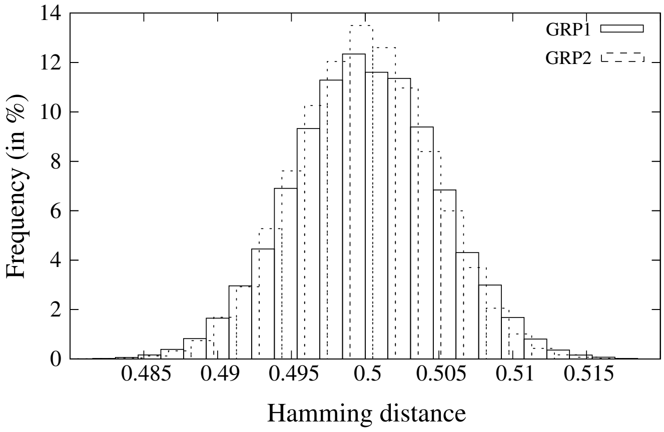

Figure 4: Histogram of Hamming distance on interval $[0.485, 0.515]$ for the group GRP1 (resp. GRP2).

5 Security analysis.

A global security analysis of the generator must be 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 the key space, the key sensitivity, the randomness quality of the outputs, weak or degenerate keys and speed performance. Even if all the existing attacks can not be tested, the PRBG must resist to some basic-known attacks. In the present case, three basic attacks are evaluated: brute-force attack, differential attack and guess-and-determine attack.

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 generator should have a large key space, to have a high diversity of choices in the generation of the pseudo-random sequence. The proposed PRBG combines three chaotic logistic maps.

We have set the conditions for seeds selection in section 3.1. The seed X_0 is a binary64 floating-point number selected from the interval $]2^{-53}, 2^{-1}[$. This 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 relative difference of 2^{-26} among each seeds. This means that Y_0 is selected in a space of $51 \times 2^{52} - 2^{26}$ possible seeds and Z_0 in a space of $51 \times 2^{52} - 2^{27}$ different numbers. The total space of seeds is approximately 2^{173}.

5.2 Key sensitivity.

The sensitivity related to the key (here the seed values) 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. In other words, even with a small difference δ on seed values, the output sequences should be completely uncorrelated. Actually in the test of correlation (Sec. 4.3.2), the seed sensitivity was already tested due to the successive seed values. To bring an additional analysis, a large pseudo-random sequences of size $N = 5000000$ (i.e. 160000000 in binary) 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 [Kendall, 1970] and the Hamming distance. The same analysis is conducted on the sets KS1 and KS2 by using the algorithm proposed by Patidar et al [Patidar, 2009a], 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.

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	Tests	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

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. Indeed, other treatments such as xor, permutations, etc., should be included in the algorithm to produce sequences of high cryptographic qualities.

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

5.3 Quality of pseudo-random sequences.

The strength of any generator is based on undeniable quality of its outputs. Indeed, whichever way the generator is designed, the produced sequences must be strong (i.e. random, uncorrelated and sensitive). In the literature, various statistical tests are available to analyse the randomness of sequences. In fact, the NIST proposes a battery of tests that must be applied on the binary sequences [Rukhin, 2010]. One can also find other tests such as TestU01 [Ecuyer, 2007] or the DieHARD suites [Marsaglia, 1996]. Here, the NIST tests are used to evaluate the randomness level of the PRBG outputs. Two sets of pseudo-random sequences were produced with different structures of seeds. For a complete analysis, the tests were applied to different aspects of the sequences (individual sequences, concatenated sequence and resulting sequences). All the produced sequences pass successfully the statistical tests. The correlation between the outputs is evaluated and the results showed that only a very small (or negligible) correlation exists between sequences. The sensitivity related to the seed values is also analysed and the results have shown that the proposed PRBG is very sensitive to these starting parameters.

5.4 Weak or degenerate keys.

A crucial element for any PRBG based chaos, is to ensure that the sequences are always generated from strong keys. Therefore, a careful study of the chaotic regions in the space of initial seeds is necessary in order to avoid weak or degenerate keys. The first task is to choose a logistic map, with parameter λ that contributes to have an excellent chaotic behaviour of the function. To avoid redundancy on the chaotic trajectories, initial seed values have to be taken in the interval $]2^{-53}, 2^{-1}[$ with a difference $\delta > 2^{-26}$ between them. To prove the quality of outputs, two groups consisting of 15000 sequences are produced using successive seed values. The seeds close to the interval bound expected to be weak are also tested. The various statistical tests clearly show the quality of tested sequences. Thus, these regions are considered as homogeneously chaotic, allowing to choose independently the seed values in the interval $]2^{-53}, 2^{-1}[$. Therefore, the proposed generator does not present weak or degenerate keys.

5.5 Speed analysis.

Beyond having a PRBG with a high level of randomness, it is also necessary to have a fast generator. Because, in real-time applications, the temporal constraint in the execution of a process is as important as the result of this process. Thus, for a fast generator, the domain of its applications can be extended. The speed performance analysis is achieved on a personal computer with Intel(R) Core(TM) 2 Duo CPU P7350 @ 2.00 GHz × 2. The algorithm is implemented using GCC on Fedora release 16 (Verne). The Table 5 shows the performance in terms of speed of the proposed algorithm compared with others. The algorithm has a good throughput which can be an advantage for applications requiring a high security level and a fast execution time.

Table 5: Comparison of speed between the proposed algorithm and some other algorithms.

Generator	Speed (Mbit/s)
Proposed	44.1120
Ref. [Pareschi, 2006]	40.0000
Ref. [Yang, 2012]	0.4844

5.6 Attacks.

Any new pseudo-random number generator must be analysed against attacks in order to check if the generator can not be broken. Here, the resistance of the generator against three basic attacks as the brute-force attack, differential attack and guess-and-determine attack is discussed.

5.6.1 Brute-force attack.

A brute-force attack [Alvarez, 2006] 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 need 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.

5.6.2 Differential attack.

Such technique of cryptanalysis was introduced by Biham and Shamir [Biham, 1993]. As a chosen-plaintext attack, its principle is to analyse 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, the difference can be computed between the two pseudo-random sequences relatively to the bits or blocks of bits.

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 initial seed values in $]2^{-53}, 2^{-1}[$ (with $\delta > 2^{-26}$) and by making 30 iterations before the beginning of the generation. Moreover, the results of the analyses showed that even with a small difference on the seeds, the produced outputs are almost independent from each other. Thus, the proposed PRBG should resist to the differential attack.

5.6.3 Guess-and-determine attack.

Guess-and-determine attack is proven to be effective against word-oriented stream ciphers [Ahmadi, 2009]. As it comes from the name, in guess-and-determine attacks, 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 produced pseudo-random sequence with the original pseudo-random sequence. If these two sequences are the same, then the guessed values are correct and the cryptosystem is broken otherwise the attack should be repeated with new guessed values. It seems that the attack discussed in reference [Ahmadi, 2009] 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 chosen 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 initial 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 until success. This process 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. For three given initial seeds, the generator produces a sequence formed of 32-bit blocks. The treatment combines permutations and xor operation on the 32 least significant bits of mantissa of the elements obtained by the logistic maps. Such a PRBG has shown its ability to produce a very large number of pseudo-random sequences which can be useful in several cryptographic applications. The advantages of the generator are: a high sensitivity related to the initial seed values, a high randomness level, the resistance against several attacks and the rapidity of the algorithm.

References

[Sun, 2009] Sun, F., and S. Liu (2009). Cryptographic pseudo-random sequence from the spatial chaotic map. *Chaos Solitons Fractals*, **41**(5), 2216–2219.

[Weisstein, Eric W.] Weisstein, Eric W. Logistic Map. *From MathWorld–A Wolfram Web Resource.*, `http://mathworld.wolfram.com/LogisticMap.html`.

[Goldberg, 1991] Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic *ACM Computing Surveys*, **23**(1),5–48.

[Miyazaki, 2010] Miyazaki, T. and Araki, S. and Uehara, S. (2010). A study on differences in properties of the logistic maps over integers affected by rounding. *Information Theory and its Applications (ISITA), 2010 International Symposium on*, **10**830–835.

[Alvarez, 2006] Álvarez, G., and S. Li (2006). Some Basic Cryptographic Requirements for Chaos-Based Cryptosystems. *International Journal of Bifurcation and Chaos*, **16**(8), 2129–2151.

[Guyeux, 2010] Guyeux, C., QianxueWang, and J.M. Bahi (2010). A Pseudo Random Numbers Generator Based on Chaotic Iterations: Application to Watermarking. *Web Information Systems and Mining*, **6318**, 202–211.

[Zheng, 2008] Zheng, F., Tian, X., Song, J., and X. LI (2008). Pseudo-random sequence generator based on the generalized Henon map. *The Journal of China Universities of Posts and Telecommunications*, **15**(3), 64–68.

[Pareek, 2010] Pareek, N.K., Patidar, V., and K.K. Sud (2010). A Random Bit Generator Using Chaotic Maps. *International Journal of Network Security*, **10**(1), 32–38.

[Patidar, 2009a] Patidar, V., and K.K. Sud (2009a). A Pseudo Random Bit Generator Based on Chaotic Logistic Map and its Statistical Testing. *Informatica* **33**, 441–452.

[Orúe, 2010] Orúe, A.B., Álvarez, G., Guerra, A., Pastor, G., Romera, M., and F. Montoya (2010). Trident, a new pseudo random number generator based on coupled chaotic maps. *Computational Intelligence in Security for Information Systems, Advances in Intelligent and Soft Computing*, **85**, 183–190.

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

[Baptista, 1998] Baptista, M.S. (1998). Cryptography with chaos. *Physics Letters A*, **240**(1-2), 50–54.

[Cecen, 2009] Cecen, S., Demirer, R.M., and C. Bayrak (2009). A new hybrid nonlinear congruential number generator based on higher functional power of logistic maps. *Chaos, Solitons and Fractals*, **42**(2), 847–853.

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

[Pareek, 2006] Pareek, N.K., Patidar, V. and K.K. Sud (2006). Image encryption using chaotic logistic map. *Image and Vision Computing*, **24**(9), 926–934.

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

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

[Patidar, 2011] Patidar, V., Pareek, N.K., Purohit, G., and K.K. Sud (2011). A robust and secure chaotic standard map based pseudorandom permutation-substitution scheme for image encryption. *Optics Communications*, **284**(19), 4331–4339.

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

[Ecuyer, 2007] L'ecuyer, P., and R. Simard (2007). TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, **33** (4), 22-es.

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

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

[Yang, 2012] Yang, L., and T. Xiao-Jun (2012). A new pseudorandom number generator based on a complex number chaotic equation. *Chinese Physics B*, **21**(9), 1–7.

[Pareschi, 2006] Pareschi, F., Setti, G., and R. Rovatti, (2006). A Fast Chaos-based True Random Number Generator for Cryptographic Applications. *in: Proceedings of he 32nd European Solid-State Circuits Conference (ESSCIRC)*, 130–133.

[Ahmadi, 2009] Ahmadi, H., and T. Eghlidos (2009). Heuristic guess-and-determine attacks on stream ciphers. *Information Security, IET*, **3**(2), 66-73.

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
```