Evaluating Quality of Chaotic Pseudo-Random Generators: Application to Information Hiding

Jacques M. Bahi, Xiaole Fang, Christophe Guyeux, and Qianxue Wang
University of Franche-Comté
Computer Science Laboratory LIFC, Besançon, France
Email:{jacques.bahi, xiaole.fang, christophe.guyeux, qianxue.wang}@univ-fcomte.fr

***Abstract*—Guaranteeing the security of information transmitted through the Internet, against passive or active attacks, is a major concern. The discovery of new pseudo-random number generators with a strong level of security is a field of research in full expansion, due to the fact that numerous cryptosystems and data hiding schemes are directly dependent on the quality of these generators. At the conference Internet'09, we described a generator based on chaotic iterations which behaves chaotically as defined by Devaney. In this paper which is an extension of the work presented at the conference Internet'10, the proposal is to improve the speed, the security, and the evaluation of this generator, to make its use more relevant in the Internet security context. In order to do so, a comparative study between various generators is carried out and statistical results are improved. Finally, an application in the information hiding framework is presented with details, to give an illustrative example of the use of such a generator in the Internet security field.**

***Keywords*-Internet security; Pseudo-random number generator; Chaotic sequences; Statistical tests; Discrete chaotic iterations; Information hiding.**

I. Introduction

Due to the rapid development of the Internet in recent years, the need to find new tools to reinforce trust and security through the Internet has become a major concern. Its recent role in everyday life implies the need to protect data and privacy in digital world. This extremely rapid development of the Internet brings more and more attention to the information security techniques in all kinds of applications. For example, new security concerns have recently appeared because of the evolution of the Internet to support such activities as e-Voting, VoD (Video on demand), and the protection of intellectual property. In all these emerging techniques, pseudo-random number generators (PRNG) play an important role, because they are fundamental components of almost all cryptosystems and information hiding schemes [1], [2]. PRNGs are typically defined by a deterministic recurrent sequence in a finite state space, usually a finite field or ring, and an output function mapping each state to an input value. Following [3], this value is often either a real number in the interval $(0, 1)$ or an integer in some finite range. PRNGs based on linear congruential methods and feedback shift-registers are popular for historical reasons [4], but their security level often has been revealed to be inadequate by today's standards. However, to use a PRNG with a high level of security is a necessity to protect the information contents sent through the Internet. This level depends both on theoretical properties and on statistical tests.

Many PRNGs have already been proven to be secure following a probabilistic approach [5], [6], [7]. However, their performances must regularly be improved, among other things by using new mathematical tools. This is why the idea of using chaotic dynamical systems for this purpose has recently been explored [8], [9]. The random-like and unpredictable dynamics of chaotic systems, their inherent determinism and simplicity of realization suggest their potential for exploitation as PRNGs. Such generators can strongly improve the confidence put in any information hiding scheme and in cryptography in general: due to their properties of unpredictability, the possibilities offered to an attacker to achieve his goal are drastically reduced in that context. For example, in cryptography, keys are needed to be unpredictable enough, to make sure any search optimization based on the reduction of the key space to the most probable values is impossible to work on. But the number of generators claimed as chaotic, which actually have been proven to be unpredictable (as it is defined in the mathematical theory of chaos) is very small.

II. Outline of our Work

This paper extends the study initiated in [10], [11], [12], and tries to fill this gap. In [11], it is mathematically proven that chaotic iterations (CIs), a suitable tool for fast computing distributed algorithms, satisfies the topological chaotic property, following the definition given by Devaney [13]. In the paper [12] presented at Internet'09, the chaotic behavior of CIs is exploited in order to obtain an unpredictable PRNG that depends on two logistic maps. We have shown that, in addition to being chaotic, this generator can pass the NIST (National Institute of Standards and Technology of the U.S. Government) battery of tests [14], widely considered as a comprehensive and stringent battery of tests for cryptographic applications. In this paper, which is an extension of [10], we have improved the speed, security, and evaluation of the former generator and of its application in information hiding. Chaotic properties, statistical tests, and security analysis [15] allow us to consider that this generator has good characteristics and is capable to withstand attacks. After having presented the theoretical framework of the study and a security analysis, we will give a comparison based on statistical tests. Finally a concrete example of how to use these pseudo-random numbers for information hiding through the Internet is detailed.

The remainder of this paper is organized in the following way. In Section III, some basic definitions concerning chaotic iterations and PRNGs are recalled. Then, the generator based on discrete chaotic iterations is presented in Section IV. Section V is devoted to its security analysis. In Section VI, various tests are passed with a goal to achieve a statistical comparison between this new PRNG and other existing ones.

In Section VII, a potential use of this PRNG in some Internet security field is presented, namely in information hiding. The paper ends with a conclusion and intended future work.

III. Review of Basics

A. Notations

$[\![1;\mathsf{N}]\!]$ $\rightarrow \{1,2,\ldots,\mathsf{N}\}$
S^n $\rightarrow$ the n^{th} term of a sequence $S=(S^1,S^2,\ldots)$
v_i $\rightarrow$ the i^{th} component of a vector $v=(v_1,v_2,\ldots,v_n)$
f^k $\rightarrow$ k^{th} composition of a function f
strategy $\rightarrow$ a sequence which elements belong in $[\![1;\mathsf{N}]\!]$
$\mathbb{S}$ $\rightarrow$ the set of all strategies
$\mathbf{C}_n^k$ $\rightarrow$ the binomial coefficient $\binom{n}{k}=\frac{n!}{k!(n-k)!}$
$\oplus$ $\rightarrow$ bitwise exclusive or
$+$ $\rightarrow$ the integer addition
$\ll$ and $\gg$ $\rightarrow$ the usual shift operators
$(\mathcal{X},\mathrm{d})$ $\rightarrow$ a metric space
mod $\rightarrow$ a modulo or remainder operator
$\lfloor x \rfloor \rightarrow$ returns the highest integer smaller than x
$n! \rightarrow$ the factorial $n!=n\times(n-1)\times\cdots\times 1$
$\mathbb{N}^* \rightarrow$ the set of positive integers {1,2,3,...}

B. XORshift

XORshift is a category of very fast PRNGs designed by George Marsaglia [16]. It repeatedly uses the transform of exclusive or (XOR) on a number with a bit shifted version of it. The state of a XORshift generator is a vector of bits. At each step, the next state is obtained by applying a given number of XORshift operations to w-bit blocks in the current state, where $w=32$ or 64. A XORshift operation is defined as follows. Replace the w-bit block by a bitwise XOR of the original block, with a shifted copy of itself by a positions either to the right or to the left, where $0<a<w$. This Algorithm 1 has a period of $2^{32}-1=4.29\times10^9$.

Input: the internal state z (a 32-bit word)
Output: y (a 32-bit word)
$z \leftarrow z \oplus (z \ll 13)$;
$z \leftarrow z \oplus (z \gg 17)$;
$z \leftarrow z \oplus (z \ll 5)$;
$y \leftarrow z$;
return y;

Algorithm 1: An arbitrary round of XORshift algorithm

C. Continuous Chaos in Digital Computers

In the past two decades, the use of chaotic systems in the design of cryptosystems, pseudo-random number generators (PRNG), and hash functions, has become more and more frequent. Generally speaking, the chaos theory in the continuous field is used to analyze performances of related systems. However, when chaotic systems are realized in digital computers with finite computing precisions, it is doubtful whether or not they can still preserve the desired dynamics of the continuous chaotic systems. Because most dynamical properties of chaos are meaningful only when dynamical systems evolve in the continuous phase space, these properties may become meaningless or ambiguous when the phase space is highly quantized (i.e., latticed) with a finite computing precision (in other words, dynamical degradation of continuous chaotic systems realized in finite computing precision). When chaotic systems are realized in finite precision, their dynamical properties will be deeply different from the properties of continuous-value systems and some dynamical degradation will arise, such as short cycle length and decayed distribution. This phenomenon has been reported and analyzed in various situations [17], [18], [19], [20], [21].

Therefore, continuous chaos may collapse into the digital world and the ideal way to generate pseudo-random sequences is to use a discrete-time chaotic system.

D. Chaos for Discrete Dynamical Systems

Consider a metric space $(\mathcal{X},d)$ and a continuous function $f:\mathcal{X}\longrightarrow\mathcal{X}$, for one-dimensional dynamical systems of the form:

$$x^0\in\mathcal{X} \text{ and } \forall n\in\mathbb{N}^*, x^n=f(x^{n-1}), \tag{1}$$

the following definition of chaotic behavior, formulated by Devaney [13], is widely accepted:

Definition 1 A dynamical system of Form (1) is said to be chaotic if the following conditions hold.

- Topological transitivity:
$$\forall U,V \text{ open sets of } \mathcal{X}\setminus\varnothing,\ \exists k>0, f^k(U)\cap V\neq\varnothing \tag{2}$$
- Density of periodic points in $\mathcal{X}$:
Let $P=\{p\in\mathcal{X}|\exists n\in\mathbb{N}^*: f^n(p)=p\}$ the set of periodic points of f. Then P is dense in $\mathcal{X}$:
$$\overline{P}=\mathcal{X} \tag{3}$$
- Sensitive dependence on initial conditions: $\exists\varepsilon>0$, $\forall x\in\mathcal{X}$, $\forall\delta>0$, $\exists y\in\mathcal{X}$, $\exists n\in\mathbb{N}$, $d(x,y)<\delta$ and $d\left(f^n(x),f^n(y)\right)\geqslant\varepsilon$.

When f is chaotic, then the system $(\mathcal{X},f)$ is chaotic and quoting Devaney: "it is unpredictable because of the sensitive dependence on initial conditions. It cannot be broken down or decomposed into two subsystems which do not interact because of topological transitivity. And, in the midst of this random behavior, we nevertheless have an element of regularity." Fundamentally different behaviors are consequently possible and occur in an unpredictable way.

E. Discrete Chaotic Iterations

Definition 2 The set $\mathbb{B}$ denoting $\{0,1\}$, let $f:\mathbb{B}^{\mathsf{N}}\longrightarrow\mathbb{B}^{\mathsf{N}}$ be an "iteration" function and $S\in\mathbb{S}$ be a chaotic strategy. Then, the so-called *chaotic iterations* [22] are defined by $x^0\in\mathbb{B}^{\mathsf{N}}$ and

$$\forall n\in\mathbb{N}^*,\forall i\in[\![1;\mathsf{N}]\!], x_i^n=\begin{cases} x_i^{n-1} & \text{if } S^n\neq i \\ f(x^{n-1})_{S^n} & \text{if } S^n=i. \end{cases} \tag{4}$$

In other words, at the n^{th} iteration, only the S^n-th cell is "iterated". Note that in a more general formulation, S^n can be a subset of components and $f(x^{n-1})_{S^n}$ can be replaced by $f(x^k)_{S^n}$, where $k<n$, describing for example delays transmission. For the general definition of such chaotic iterations, see, e.g., [22].

Chaotic iterations generate a set of vectors (Boolean vector in this paper), they are defined by an initial state x^0, an iteration function f, and a chaotic strategy S.

The next section gives the outline proof that chaotic iterations satisfy Devaney's topological chaos property. Thus they can be used to define a chaotic pseudo-random bit generator.

IV. The Generation of CI Pseudo-Random Sequence

A. A Theoretical Proof for Devaney's Chaotic Dynamical Systems

The outline proofs, of the properties on which our pseudo-random number generator is based, are given in this section.

Denote by δ the *discrete Boolean metric*, $\delta(x,y)=0 \Leftrightarrow x=y$. Given a function f, define the function $F_f : [\![1;\mathsf{N}]\!]\times\mathbb{B}^{\mathsf{N}} \longrightarrow \mathbb{B}^{\mathsf{N}}$ such that

$$F_f(k,E)=\left(E_j.\delta(k,j)+f(E)_k.\overline{\delta(k,j)}\right)_{j\in[\![1;\mathsf{N}]\!]},$$

where + and . are the Boolean addition and product operations.

Consider the phase space: $\mathcal{X}=[\![1;\mathsf{N}]\!]^{\mathbb{N}}\times\mathbb{B}^{\mathsf{N}}$ and the map

$$G_f\left(S,E\right)=\left(\sigma(S),F_f(i(S),E)\right),$$

then the chaotic iterations defined in (III-E) can be described by the following iterations [11]

$$\left\{\begin{array}{l} X^0\in\mathcal{X}\\ X^{k+1}=G_f(X^k).\end{array}\right.$$

Let us define a new distance between two points $(S,E),(\check{S},\check{E})\in\mathcal{X}$ by

$$d((S,E);(\check{S},\check{E}))=d_e(E,\check{E})+d_s(S,\check{S}),$$

where

- $d_e(E,\check{E})=\sum_{k=1}^{\mathsf{N}}\delta(E_k,\check{E}_k)\in[\![0;\mathsf{N}]\!]$
- $d_s(S,\check{S})=\frac{9}{\mathsf{N}}\sum_{k=1}^{\infty}\frac{|S^k-\check{S}^k|}{10^k}\in[0;1]$.

It is then proven in [11] by using the sequential continuity that

Proposition 1 *G_f is a continuous function on $(\mathcal{X},d)$.*

Then, the vectorial negation $f_0(x_1,\ldots,x_{\mathsf{N}})=(\overline{x_1},\ldots,\overline{x_{\mathsf{N}}})$ satisfies the three conditions for Devaney's chaos, namely, regularity, transitivity, and sensitivity in the metric space $(\mathcal{X},d)$. This leads to the following result.

Proposition 2 *G_{f_0} is a chaotic map on $(\mathcal{X},d)$ in the sense of Devaney.*

B. Chaotic Iterations as Pseudo-Random Generator

1) Presentation: The CI generator (generator based on chaotic iterations) is designed by the following process. First of all, some chaotic iterations have to be done to generate a sequence $(x^n)_{n\in\mathbb{N}}\in\left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}}$ ($\mathsf{N}\in\mathbb{N}^*,\mathsf{N}\geqslant 2$, N is not necessarily equal to 32) of Boolean vectors, which are the successive states of the iterated system. Some of these vectors will be randomly extracted and our pseudo-random bit flow will be constituted by their components. Such chaotic iterations are realized as follows. Initial state $x^0\in\mathbb{B}^{\mathsf{N}}$ is a Boolean vector taken as a seed (see Section IV-B2) and chaotic strategy $(S^n)_{n\in\mathbb{N}}\in[\![1,\mathsf{N}]\!]^{\mathbb{N}}$ is an irregular decimation of a XORshift sequence (Section IV-B4). The iterate function f is the vectorial Boolean negation:

$$f_0:(x_1,...,x_{\mathsf{N}})\in\mathbb{B}^{\mathsf{N}}\longmapsto(\overline{x_1},...,\overline{x_{\mathsf{N}}})\in\mathbb{B}^{\mathsf{N}}.$$

At each iteration, only the S^i-th component of state $\underline{x^n}$ is updated, as follows: $x_i^n=x_i^{n-1}$ if $i\neq S^i$, else $x_i^n=\overline{x_i^{n-1}}$. Finally, some x^n are selected by a sequence m^n as the pseudo-random bit sequence of our generator. $(m^n)_{n\in\mathbb{N}}\in\mathcal{M}^{\mathbb{N}}$ is computed from a XORshift sequence $(y^n)_{n\in\mathbb{N}}\in[\![0,2^{32}-1]\!]$ (see Section IV-B3). So, the generator returns the following values:
Bits:

$$x_1^{m_0}x_2^{m_0}x_3^{m_0}\ldots x_{\mathsf{N}}^{m_0}x_1^{m_0+m_1}x_2^{m_0+m_1}\ldots x_{\mathsf{N}}^{m_0+m_1}x_1^{m_0+m_1+m_2}\ldots$$

or States:

$$x^{m_0}x^{m_0+m_1}x^{m_0+m_1+m_2}\ldots$$

2) The seed: The unpredictability of random sequences is established using a random seed that is obtained by a physical source like timings of keystrokes. Without the seed, the attacker must not be able to make any predictions about the output bits, even when all details of the generator are known [23].

The initial state of the system x^0 and the first term y^0 of the XORshift are seeded either by the current time in seconds since the Epoch, or by a number that the user inputs. Different ways are possible. For example, let us denote by t the decimal part of the current time. So x^0 can be t (mod 2^N) written in binary digits and $y^0=t$.

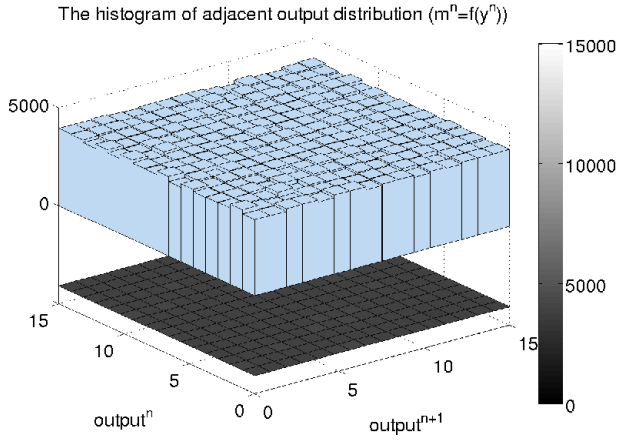
3) Sequence m of returned states: The output of the sequence (y^n) is uniform in $[\![0,2^{32}-1]\!]$, because it is produced by a XORshift generator. However, we do not want the output of (m^n) to be uniform in $[\![0,N]\!]$, because in this case, the returns of our generator will not be uniform in $[\![0,2^{\mathsf{N}}-1]\!]$, as it is illustrated in the following example. Let us suppose that $x^0=(0,0,0)$. Then $m^0\in[\![0,3]\!]$.

- If $m^0=0$, then no bit will change between the first and the second output of our PRNG. Thus $x^1=(0,0,0)$.
- If $m^0=1$, then exactly one bit will change, which leads to three possible values for x^1, namely $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$.
- *etc.*

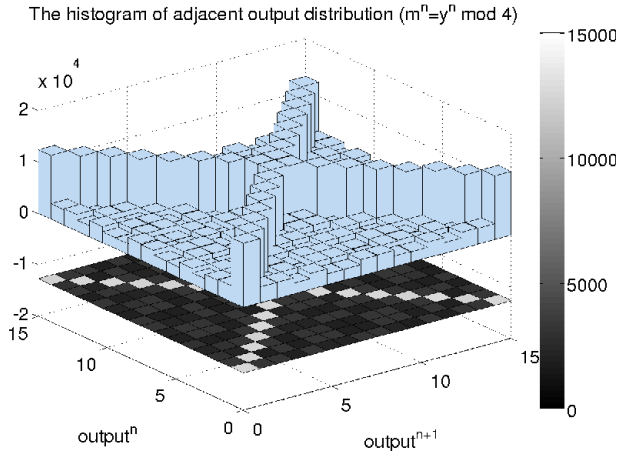
As each value in $[\![0,2^3-1]\!]$ must be returned with the same probability, then the values $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$ must occur for x^1 with the same probability. Finally we see that, in this example, $m^0=1$ must be three times more probable than $m^0=0$. This leads to the following general definition for m:

$$m^n=g_1(y^n)=\left\{\begin{array}{l} 0 \text{ if } 0\leqslant\frac{y^n}{2^{32}}<\frac{C_{\mathsf{N}}^0}{2^{\mathsf{N}}},\\ 1 \text{ if } \frac{C_{\mathsf{N}}^0}{2^{\mathsf{N}}}\leqslant\frac{y^n}{2^{32}}<\sum_{i=0}^{1}\frac{C_{\mathsf{N}}^i}{2^{\mathsf{N}}},\\ 2 \text{ if } \sum_{i=0}^{1}\frac{C_{\mathsf{N}}^i}{2^{\mathsf{N}}}\leqslant\frac{y^n}{2^{32}}<\sum_{i=0}^{2}\frac{C_{\mathsf{N}}^i}{2^{\mathsf{N}}},\\ \vdots\quad\quad\vdots\\ \mathsf{N} \text{ if } \sum_{i=0}^{\mathsf{N}-1}\frac{C_{\mathsf{N}}^i}{2^{\mathsf{N}}}\leqslant\frac{y^n}{2^{32}}<1.\end{array}\right. \quad (5)$$

In order to evaluate our proposed method and compare its statistical properties with various other methods, the density histogram and intensity map of adjacent outputs have been computed. The length of x is $\mathsf{N}=4$ bits, and the initial conditions and control parameters are the same. A large number of sampled values are simulated (10^6 samples). Figure 1(a) shows



(a) $m^n = f(y^n)$



(b) $m^n = y^n \ mod \ 4$

Fig. 1: Histogram and intensity maps

the intensity map for $m^n = g_1(y^n)$. In order to appear random, the histogram should be uniformly distributed in all areas. It can be observed that a uniform histogram and a flat color intensity map are obtained when using our scheme. Another illustration of this fact is given by Figure 1(b), whereas its uniformity is further justified by the tests presented in Section VI.

4) Chaotic strategy: The chaotic strategy $(S^k) \in [\![1, \mathsf{N}]\!]^{\mathbb{N}}$ is generated from a second XORshift sequence $(b^k) \in [\![1, N]\!]^{\mathbb{N}}$. The only difference between the sequences S and b is that some terms of b are discarded, in such a way that $\forall k \in \mathbb{N}, (S^{M^k}, S^{M^k+1}, \ldots, S^{M^{k+1}-1})$ does not contain any given integer twice, where $M^k = \sum_{i=0}^{k} m^i$. Therefore, no bit will change more than once between two successive outputs of our PRNG, increasing the speed of the former generator by doing so. S is said to be "an irregular decimation" of b. This decimation can be obtained by the following process.

Let $(d^1, d^2, \ldots, d^{\mathsf{N}}) \in \{0,1\}^{\mathsf{N}}$ be a mark sequence, such that whenever $\sum_{i=1}^{\mathsf{N}} d^i = m^k$, then $\forall i, d_i = 0$ ($\forall k$, the sequence is reset when d contains m^k times the number 1). This mark sequence will control the XORshift sequence b as follows:

- if $d^{b^j} \neq 1$, then $S^k = b^j$, $d^{b^j} = 1$, and $k = k + 1$,
- if $d^{b^j} = 1$, then b^j is discarded.

For example, if $b = 1422\underline{3}3414211\underline{22}34...$ and $m = 4341...$, then $S = 1423\ 341\ 4123\ 4...$ However, if we do not use the mark sequence, then one position may change more than once and the balance property will not be checked, due to the fact that $\bar{\bar{x}} = x$. As an example, for b and m as in the previous example, $S = 1422\ 334\ 1421\ 1...$ and $S = 14\ 4\ 42\ 1...$ lead to the same outputs (because switching the same bit twice leads to the same state).

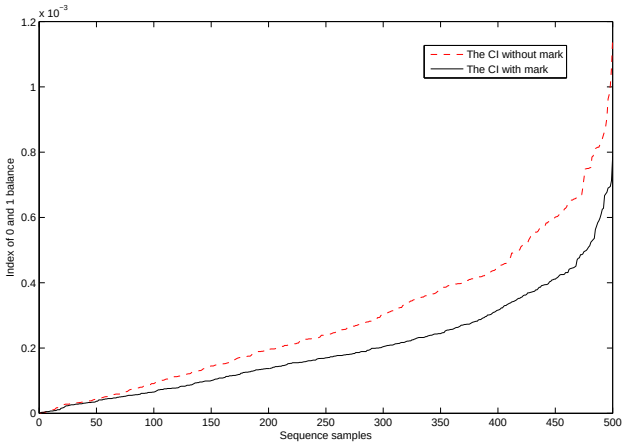


Fig. 2: Balance property

To check the balance property, a set of 500 sequences are generated with and without decimation, each sequence containing 10^6 bits. Figure 2 shows the percentages of differences between zeros and ones, and presents a better balance property for the sequences with decimation. This claim will be verified in the tests section (Section VI).

Another example is given in Table I, in which r means "reset" and the integers which are underlined in sequence b are discarded.

C. CI(XORshift, XORshift) Algorithm

The basic design procedure of the novel generator is summed up in Algorithm 2. The internal state is x, the output state is r. a and b are those computed by the two XORshift generators. The value $g_1(a)$ is an integer, defined as in Equation 5. Lastly, N is a constant defined by the user.

As a comparison, the basic design procedure of the old generator is recalled in Algorithm 3 (a and b are computed by logistic maps, N and $c \geqslant 3\mathsf{N}$ are constants defined by the user). See [12] for further information.

D. Illustrative Example

In this example, $\mathsf{N} = 4$ is chosen for easy understanding. As stated before, the initial state of the system x^0 can be seeded by the decimal part t of the current time. For example, if the current time in seconds since the Epoch is 1237632934.484088, so $t = 484088$, then $x^0 = t \ (mod\ 16)$ in binary digits, *i.e.*, $x^0 = (0, 1, 0, 0)$.

To compute m sequence, Equation 5 can be adapted to this example as follows:

$$m^n = g_1(y^n) = \begin{cases} 0 & \text{if} \quad 0 \leqslant \frac{y^n}{2^{32}} < \frac{1}{16}, \\ 1 & \text{if} \quad \frac{1}{16} \leqslant \frac{y^n}{2^{32}} < \frac{5}{16}, \\ 2 & \text{if} \quad \frac{5}{16} \leqslant \frac{y^n}{2^{32}} < \frac{11}{16}, \\ 3 & \text{if} \quad \frac{11}{16} \leqslant \frac{y^n}{2^{32}} < \frac{15}{16}, \\ 4 & \text{if} \quad \frac{15}{16} \leqslant \frac{y^n}{2^{32}} < 1, \end{cases} \tag{6}$$

Input: the internal state x (N bits)
Output: a state r of N bits
for $i = 0, \ldots, \mathsf{N}$ **do**
  $d_i \leftarrow 0$;
end
$a \leftarrow XORshift1()$;
$m \leftarrow g_1(a)$;
$k \leftarrow m$;
for $i = 0, \ldots, k$ **do**
  $b \leftarrow XORshift2() \ mod \ \mathsf{N}$;
  $S \leftarrow b$;
  if $d_S = 0$ **then**
    $x_S \leftarrow \overline{x_S}$;
    $d_S \leftarrow 1$;
  end
  else if $d_S = 1$ **then**
    $k \leftarrow k + 1$;
  end
end
$r \leftarrow x$;
return r;

Algorithm 2: An arbitrary round of the new CI(XORshift,XORshift) generator

Input: the internal state x (N bits)
Output: a state r of N bits
$a \leftarrow Logisticmap1()$;
if $a > 0.5$ **then**
  $d \leftarrow 1$
end
else
  $d \leftarrow 0$
end
$m \leftarrow d + c$;
for $i = 0, \ldots, m$ **do**
  $b \leftarrow Logisticmap2()$;
  $S \leftarrow 100000b \ mod \ \mathsf{N}$;
  $x_S \leftarrow \overline{x_S}$;
end
$r \leftarrow x$;
return r;

Algorithm 3: An arbitrary round of the old CI PRNG

where y is generated by XORshift seeded with the current time. We can see that the probabilities of occurrences of $m = 0$, $m = 1$, $m = 2$, $m = 3$, $m = 4$, are $\frac{1}{16}$, $\frac{4}{16}$, $\frac{6}{16}$, $\frac{4}{16}$, $\frac{1}{16}$, respectively. This m determines what will be the next output x. For instance,

- If $m = 0$, the following x will be $(0, 1, 0, 0)$.
- If $m = 1$, the following x can be $(1, 1, 0, 0)$, $(0, 0, 0, 0)$, $(0, 1, 1, 0)$, or $(0, 1, 0, 1)$.
- If $m = 2$, the following x can be $(1, 0, 0, 0)$, $(1, 1, 1, 0)$, $(1, 1, 0, 1)$, $(0, 0, 1, 0)$, $(0, 0, 0, 1)$, or $(0, 1, 1, 1)$.
- If $m = 3$, the following x can be $(0, 0, 1, 1)$, $(1, 1, 1, 1)$, $(1, 0, 0, 1)$, or $(1, 0, 1, 0)$.
- If $m = 4$, the following x will be $(1, 0, 1, 1)$.

In this simulation, $m = 0, 4, 2, 2, 3, 4, 1, 1, 2, 3, 0, 1, 4, ...$ Additionally, b is computed with a XORshift generator too, but with another seed. We have found $b = 1, 4, 2, 2, 3, 3, 4, 1, 1, 4, 3, 2, 1, ...$

Chaotic iterations are made with initial state x^0, vectorial logical negation f_0, and strategy S. The result is presented in Table I. Let us recall that sequence m gives the states x^n to return, which are here $x^0, x^{0+4}, x^{0+4+2}, \ldots$ So, in this example, the output of the generator is: 10100111101111110011... or 4,4,11,8,1...

V. Security Analysis

PRNG should be sensitive with respect to the secret key and its size. Here, chaotic properties are also in close relation with the security.

A. Key Space

The PRNG proposed in this paper is based on discrete chaotic iterations. It has an initial value $x^0 \in \mathbb{B}^{\mathsf{N}}$. Considering this set of initial values alone, the key space size is equal to 2^{N}. In addition, this new generator combines digits of two other PRNGs. We used two different XORshifts here. Let k be the key space of XORshift, so the total key space size is close to $2^{\mathsf{N}} \cdot k^2$. Lastly, the impact of Equation 5, in which is defined the (m^n) sequence with a selector function g_1, must be taken into account. This leads to conclude that the key space size is large enough to withstand attacks.

Let us notice, to conclude this subsection, that our PRNG can use any reasonable function as selector. In this paper, $g_1()$ and $g_2()$ are adopted for demonstration purposes, where:

$$m^n = g_2(y^n) = \begin{cases} \mathsf{N} \text{ if } 0 \leqslant \frac{y^n}{2^{32}} < \frac{C_{\mathsf{N}}^0}{2^{\mathsf{N}}}, \\ \mathsf{N}-1 \text{ if } \frac{C_N^0}{2^{\mathsf{N}}} \leqslant \frac{y^n}{2^{32}} < \sum_{i=0}^{1} \frac{C_{\mathsf{N}}^i}{2^{\mathsf{N}}}, \\ \mathsf{N}-2 \text{ if } \sum_{i=0}^{1} \frac{C_{\mathsf{N}}^i}{2^{\mathsf{N}}} \leqslant \frac{y^n}{2^{32}} < \sum_{i=0}^{2} \frac{C_{\mathsf{N}}^i}{2^{\mathsf{N}}}, \\ \vdots \quad \vdots \\ 0 \text{ if } \sum_{i=0}^{\mathsf{N}-1} \frac{C_{\mathsf{N}}^i}{2^{\mathsf{N}}} \leqslant \frac{y^n}{2^{32}} < 1. \end{cases} \tag{7}$$

We will show later that both of them can pass all of the performed tests.

B. Key Sensitivity

As a consequence of its chaotic property, this PRNG is highly sensitive to the initial conditions. To illustrate this fact, several initial values are put into the chaotic system. Let H be the number of differences between the sequences obtained in this way. Suppose n is the length of these sequences. Then the variance ratio P, defined by $P = H/n$, is computed. The results are shown in Figure 3 (x axis is sequence lengths, y axis is variance ratio P). For the two PRNGs, variance ratios approach 0.50, which indicates that the system is extremely sensitive to the initial conditions.

C. Linear Complexity

The linear complexity (LC) of a sequence is the size in bits of the shortest linear feedback shift register (LFSR) which can produce this sequence. This value measures the difficulty of generating – and perhaps analyzing – a particular sequence. Indeed, the randomness of a given sequence can be linked to the size of the smallest program that can produce it. LC is the size required by a LFSR to be able to produce the given sequence. The Berlekamp-Massey algorithm can measure this

m	0	4					2		2		
k	0	4			$+1$		2		2	$+1$	
b		1	4	2	$\underline{2}$	3	3	4	1	$\underline{1}$	4
d	r	r $\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$	$\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$	$\begin{pmatrix}1\\1\\0\\1\end{pmatrix}$		$\begin{pmatrix}1\\1\\1\\1\end{pmatrix}$	r $\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\\1\end{pmatrix}$	r $\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$		$\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$
S		1	4	2		3	3	4	1		4
x^0	x^0					x^4		x^6			x^8
0	0	$\xrightarrow{1}1$				1		1	$\xrightarrow{1}0$		0
1	1			$\xrightarrow{2}0$		0		0			0
0	0				$\xrightarrow{3}1$	1	$\xrightarrow{3}0$	0			0
0	0		$\xrightarrow{4}1$			1		$\xrightarrow{4}0$ 0		$\xrightarrow{4}1$	1

Binary Output: $x_1^0x_2^0x_3^0x_4^0x_1^4x_2^4x_3^4x_4^4x_1^6x_2^6... = 0100101110000001...$
Integer Output: $x^0, x^4, x^6, x^8... = 4, 11, 8, 1...$

TABLE I: Example of New CI(XORshift,XORshift) generation

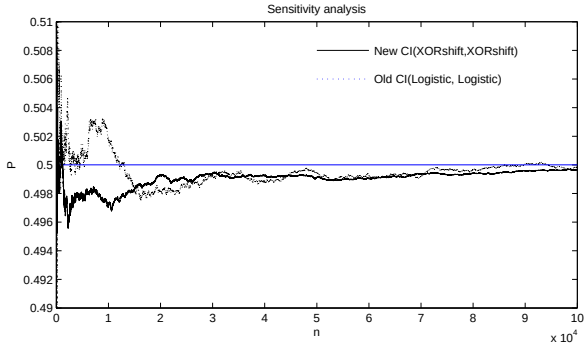


Fig. 3: Sensitivity analysis

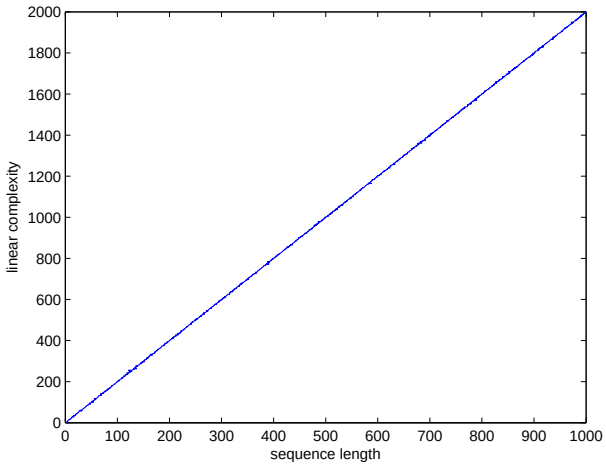


Fig. 4: Linear complexity

LC, which can be used to evaluate the security of a pseudo-random sequence. It can be seen in Figure 4 that the LC curve of a sample sequence of 2000 bits is close to the ideal line $C_i = i/2$, which implies that the generator has high linear complexity.

D. Devaney's Chaos Property

Generally speaking, the quality of a PRNG depends, to a large extent, on the following criteria: randomness, uniformity, independence, storage efficiency, and reproducibility. A chaotic sequence may satisfy these requirements and also other chaotic properties, as ergodicity, entropy, and expansivity. A chaotic sequence is extremely sensitive to the initial conditions. That is, even a minute difference in the initial state of the system can lead to enormous differences in the final state, even over fairly small timescales. Therefore, chaotic sequence fits the requirements of pseudo-random sequence well. Contrary to XORshift, our generator possesses these chaotic properties [11],[12]. However, despite a large number of papers published in the field of chaos-based pseudo-random generators, the impact of this research is rather marginal. This is due to the following reasons: almost all PRNG algorithms using chaos are based on dynamical systems defined on continuous sets (*e.g.*, the set of real numbers). So these generators are usually slow, requiring considerably more storage space, and lose their chaotic properties during computations as mentioned earlier in this paper. These major problems restrict their use as generators [24].

In this paper, we do not simply integrate chaotic maps hoping that the implemented algorithm remains chaotic. Indeed, the PRNG we conceive is just discrete chaotic iterations and we have proven in [11] that these iterations produce a topological chaos as defined by Devaney: they are regular, transitive, and sensitive to initial conditions. This famous definition of a chaotic behavior for a dynamical system implies unpredictability, mixture, sensitivity, and uniform repartition. Moreover, as only integers are manipulated in discrete chaotic iterations, the chaotic behavior of the system is preserved during computations, and these computations are fast.

Let us now explore the topological properties of our generator and their consequences concerning the quality of the generated pseudo-random sequences.

E. Topological Consequences

We have proven in [25] that chaotic iterations are expansive and topologically mixing. These topological properties are inherited by the generators we presented here. In particular, any error on the seed are magnified until being equal to the constant of expansivity. We will now investigate the consequences of being chaotic, as defined by Devaney.

First of all, the transitivity property implies the indecomposability of the system:

Definition 3 A dynamical system $(\mathcal{X}, f)$ is indecomposable if it is not the union of two closed sets $A, B \subset \mathcal{X}$ such that $f(A) \subset A, f(B) \subset B$.

Thus it is impossible to reduce the set of the outputs generated by our PRNG, in order to reduce its complexity. Moreover, it is possible to show that Old and New CI generators are strongly transitive:

Definition 4 A dynamical system $(\mathcal{X}, f)$ is strongly transitive if $\forall x, y \in \mathcal{X}$, $\forall r > 0$, $\exists z \in \mathcal{X}$, $d(z, x) \leqslant r \Rightarrow \exists n \in \mathbb{N}^*$, $f^n(z) = y$.

In other words, for all $x, y \in \mathcal{X}$, it is possible to find a point z in the neighborhood of x such that an iterate $f^n(z)$ is y. Indeed, this result has been established during the proof of the transitivity presented in [11]. Among other things, the strong transitivity property leads to the fact that without the knowledge of the seed, all of the outputs are possible. Additionally, no point of the output space can be discarded when studying our PRNG: it is intrinsically complicated and it cannot be simplified.

Finally, these generators possess the instability property:

Definition 5 A dynamical system $(\mathcal{X}, f)$ is unstable if for all $x \in \mathcal{X}$, the orbit $\gamma_x : n \in \mathbb{N} \longmapsto f^n(x)$ is unstable, that is: $\exists \varepsilon > 0$, $\forall \delta > 0$, $\exists y \in \mathcal{X}$, $\exists n \in \mathbb{N}$, $d(x, y) < \delta$ and $d\left(\gamma_x(n), \gamma_y(n)\right) \geqslant \varepsilon$.

This property, which is implied by the sensitive dependence to the initial condition, leads to the fact that in all of the neighborhoods of any x, there are points that are separate from x under iterations of f. We thus can claim that the behavior of our generators is unstable.

VI. Statistical Analysis

A. Basic Common Tests

1) Comparative test parameters: In this section, five well-known statistical tests [26] are used as comparison tools. They encompass frequency and autocorrelation tests. In what follows, $s = s^0, s^1, s^2, \ldots, s^{n-1}$ denotes a binary sequence of length n. The question is to determine whether this sequence possesses some specific characteristics that a truly random sequence would be likely to exhibit. The tests are introduced in this subsection and results are given in the next one.

a) Frequency test (monobit test): The purpose of this test is to check if the numbers of 0's and 1's are approximately equal in s, as it would be expected for a random sequence. Let n_0, n_1 denote these numbers. The statistic used here is:

$$X_1 = \frac{(n_0 - n_1)^2}{n},$$

which approximately follows a χ^2 distribution with one degree of freedom when $n \geqslant 10^7$.

b) Serial test (2-bit test): The purpose of this test is to determine if the number of occurrences of 00, 01, 10, and 11 as subsequences of s are approximately the same. Let n_{00}, n_{01}, n_{10}, and n_{11} denote the number of occurrences of $00, 01, 10$, and 11 respectively. Note that $n_{00} + n_{01} + n_{10} + n_{11} = n - 1$ since the subsequences are allowed to overlap. The statistic used here is:

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1,$$

which approximately follows a χ^2 distribution with 2 degrees of freedom if $n \geqslant 21$.

c) Poker test: The poker test studies if each pattern of length m (without overlapping) appears the same number of times in s. Let $\lfloor \frac{n}{m} \rfloor \geqslant 5 \times 2^m$ and $k = \lfloor \frac{n}{m} \rfloor$. Divide the sequence s into k non-overlapping parts, each of length m. Let n_i be the number of occurrences of the i^{th} type of sequence of length m, where $1 \leqslant i \leqslant 2^m$. The statistic used is

$$X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right) - k,$$

which approximately follows a χ^2 distribution with $2^m - 1$ degrees of freedom. Note that the poker test is a generalization of the frequency test: setting $m = 1$ in the poker test yields the frequency test.

d) Runs test: The purpose of the runs test is to figure out whether the number of runs of various lengths in the sequence s is as expected for a random sequence. A run is defined as a pattern of all zeros or all ones, a block is a run of ones, and a gap is a run of zeros. The expected number of gaps (or blocks) of length i in a random sequence of length n is $e_i = \frac{n-i+3}{2^{i+2}}$. Let k be equal to the largest integer i such that $e_i \geqslant 5$. Let B_i, G_i be the number of blocks and gaps of length i in s, for each $i \in [\![1, k]\!]$. The statistic used here will then be:

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i},$$

which approximately follows a χ^2 distribution with $2k - 2$ degrees of freedom.

e) Autocorrelation test: The purpose of this test is to check for coincidences between the sequence s and (non-cyclic) shifted versions of it. Let d be a fixed integer, $1 \leqslant d \leqslant \lfloor n/2 \rfloor$. The value $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$ is the amount of bits not equal between the sequence and itself displaced by d bits. The statistic used here is:

$$X_5 = |2(A(d) - \frac{n-d}{2})/\sqrt{n-d}|,$$

which approximately follows a normal distribution $\mathcal{N}(0, 1)$ if $n - d \geqslant 10$. Since small values of $A(d)$ are as unexpected as large values, a two-sided test should be used.

2) Comparison: We show in Table II a comparison among our new generator CI(XORshift, XORshift), its old version denoted Old CI(Logistic, Logistic), a basic PRNG based on logistic map, and a simple XORshift. In this table, time (in seconds) is related to the duration needed by each algorithm to generate a 2×10^5 bits long sequence. The test has been conducted using the same computer and compiler with the same optimization settings for both algorithms, in order to make the test as fair as possible. The results confirm that the proposed generator is a lot faster than the old one, while the statistical results are better for most of the parameters, leading to the conclusion that the new PRNG is more secure than the old one. Although the logistic map also has good results, it is too slow to be implemented in Internet applications, and this map is known to present various bias leading to severe security issues.

As a comparison of the overall stability of these PRNGs, similar tests have been computed for different sequence lengths (see Figures 5 - 9). For the monobit test comparison (Figure 5),

TABLE II: Comparison with Old CI(Logistic, Logistic) for a 2×10^5 bits sequence

Method	Monobit (X_1)	Serial (X_2)	Poker (X_3)	Runs (X_4)	Autocorrelation (X_5)	Time
Logistic map	0.1280	0.1302	240.2893	26.5667	0.0373	0.965s
XORshift	1.7053	2.1466	248.9318	18.0087	0.5009	0.096s
Old CI(Logistic, Logistic)	1.0765	1.0796	258.1069	20.9272	1.6994	0.389s
New CI(XORshift,XORshift)	0.3328	0.7441	262.8173	16.7877	0.0805	0.197s

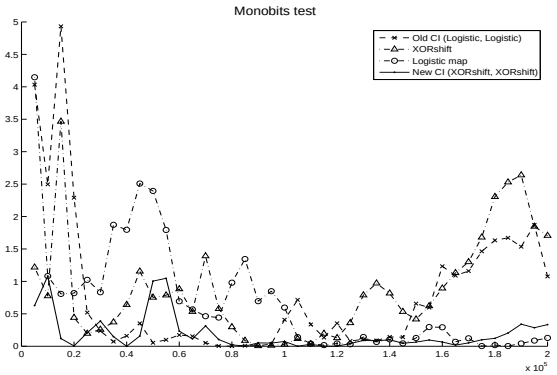


Fig. 5: Comparison of monobits tests

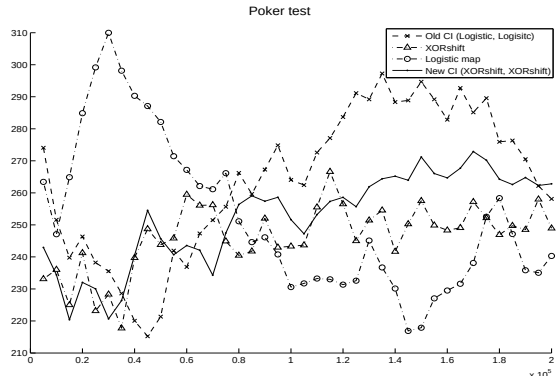


Fig. 7: Comparison of poker tests

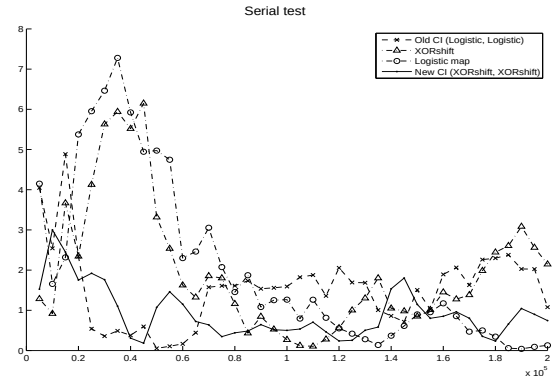


Fig. 6: Comparison of serial tests

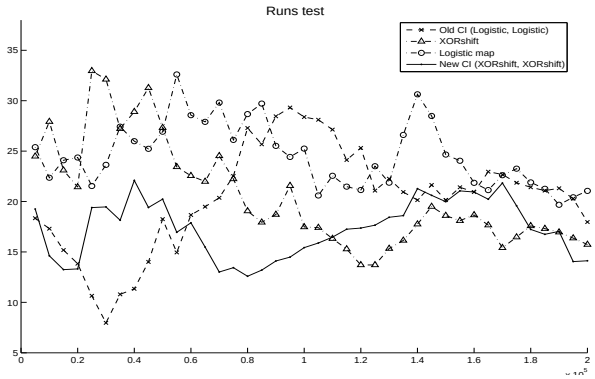


Fig. 8: Comparison of runs tests

almost all of the PRNGs present the same issue: the beginning values are a little high. However, for our new generator, the values are stable in a low level which never exceeds 1.2. Indeed, the new generator distributes very randomly the zeros and ones, whatever the length of the desired sequence. It can also be remarked that the old generator presents the second best performance, due to its use of chaotic iterations.

Figure 6 shows the serial test comparison. The new generator outperforms this test, but the score of the old generator is not bad either: their occurrences of 00, 01, 10, and 11 are very close to each other.

The poker test comparison with $m = 8$ is shown in Figure 7. XORshift is the most stable generator in all of these tests, and the logistic map also becomes good when producing sequences of length greater than 1×10^5. Our old and new generators present a similar trend, with a maximum in the neighborhood of 1.7×10^5. These scores are not so good, even though the new generator has a better behavior than the old one. Indeed, the value of m and the length of the sequences should be enlarged to be certain that the chaotic iterations express totally their complex behavior. In that situation, the performances of our generators in the poker test can be improved.

The graph of the new generator is the most stable one during the runs test comparison (Figure 8). Moreover, this trend is reinforced when the lengths of the tested sequences are increased.

The comparison of autocorrelation tests is presented in Figure 9. The new generator clearly dominates these tests, whereas the score of the old generator is surprisingly bad. This difference between two generators based on chaotic iterations

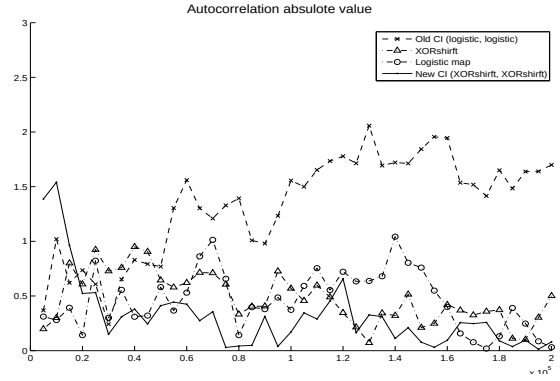


Fig. 9: Comparison of autocorrelation tests

can be explained by the fact that the improvements realized to define the new generator lead to a more randomly output.

To sum up we can claim that the new generator, which is faster than its former version, outperforms all of the other generators in these statistical tests, especially when producing long output sequences.

B. NIST Statistical Test Suite

1) Presentation: Among the numerous standard tests for pseudo-randomness, a convincing way to prove the quality of the produced sequences is to confront them with the NIST (National Institute of Standards and Technology) Statistical Test Suite SP 800-22, released by the Information Technology Laboratory in August 25, 2008.

The NIST test suite, SP 800-22, is a statistical package consisting of 15 tests. They were developed to measure the randomness of (arbitrarily long) binary sequences produced by either hardware or software based cryptographic pseudo-random number generators. These tests focus on a variety of different types of non-randomness that could occur in such sequences. These 15 tests include in the NIST test suite are described in the Appendix.

2) Interpretation of empirical results: $\mathbb{P}$ is the "tail probability" that the chosen test statistic will assume values that are equal to or worse than the observed test statistic value when considering the null hypothesis. For each statistical test, a set of $\mathbb{P}$s is produced from a set of sequences obtained by our generator (i.e., 100 sequences are generated and tested, hence 100 $\mathbb{P}$s are produced).

Empirical results can be interpreted in various ways. In this paper, we check whether the $\mathbb{P}$s are uniformly distributed, via an application of a χ^2 distribution and the determination of a $\mathbb{P}_T$ corresponding to the Goodness-of-Fit distributional test on the $\mathbb{P}$s obtained for an arbitrary statistical test.

If $\mathbb{P}_T \geq 0.0001$, then the sequences can be considered to be uniformly distributed. In our experiments, 100 sequences (s = 100) of 1,000,000 bits are generated and tested. If the value $\mathbb{P}_T$ of a least one test is smaller than 0.0001, the sequences are considered to be not good enough and the generator is unsuitable.

Table III shows $\mathbb{P}_T$ for the sequences based on discrete chaotic iterations using different schemes. If there are at least two statistical values in a test, this test is marked with an asterisk and the average is computed to characterize the statistical values.

We can conclude from Table III that the worst situations are obtained with the New CI ($m^n = y^n \ mod \ N$) and New CI (no mark) generators. Old CI, New CI ($m^n = g_1(y^n)$), and New CI ($m^n = g_2(y^n)$) have successfully passed the NIST statistical test suite. These results and the conclusion obtained from the aforementioned basic tests reinforce the confidence that can be put in the good behavior of chaotic CI PRNGs, thus making them suitable for security applications as information hiding and digital watermarking.

VII. Application Example in Information Hiding

A. Introduction

Information hiding is now an integral part of Internet technologies. In the field of social search engines, for example, contents like pictures or movies are tagged with descriptive labels by contributors, and search results are determined by these descriptions. These collaborative taggings, used for example in Flickr [27] and Delicious [28] websites, contribute to the development of a Semantic Web, in which any Web page contains machine-readable metadata that describe its content. Information hiding technologies can be used for embedding these metadata. The advantage of its use is the possibility to realize social search without websites and databases: descriptions are directly embedded into media, whatever their formats. Robustness is required in this situation, as descriptions should resist to modifications like resizing, compression, and format conversion.

The Internet security field is also concerned by watermarking technologies. Steganography and cryptography are supposed to be used by terrorists to communicate through the Internet. Furthermore, in the areas of defense or in industrial espionage, many information leaks using steganographic techniques have been reported. Lastly, watermarking is often cited as a possible solution to digital rights managements issues, to counteract piracy of digital work in an Internet based entertainment world [29].

B. Definition of a Chaos-Based Information Hiding Scheme

Let us now introduce our information hiding scheme based on CI generator.

1) Most and least significant coefficients: Let us define the notions of most and least significant coefficients of an image.

Definition 1 For a given image, most significant coefficients (in short MSCs), are coefficients that allow the description of the relevant part of the image, *i.e.*, its richest part (in terms of embedding information), through a sequence of bits.

For example, in a spatial description of a grayscale image, a definition of MSCs can be the sequence constituted by the first four bits of each pixel (see Figure 10). In a discrete cosine frequency domain description, each 8×8 block of the carrier image is mapped onto a list of 64 coefficients. The energy of the image is mostly contained in a determined part of themselves, which can constitute a possible sequence of MSCs.

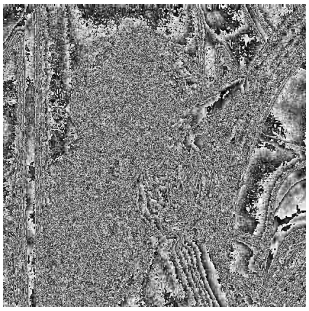
Definition 2 By least significant coefficients (LSCs), we mean a translation of some insignificant parts of a medium in a sequence of bits (insignificant can be understand as: "which can be altered without sensitive damages").

These LSCs can be, for example, the last three bits of the gray level of each pixel (see Figure 10). Discrete cosine, Fourier, and wavelet transforms can be used also to generate LSCs and MSCs. Moreover, these definitions can be extended to other types of media.

LSCs are used during the embedding stage. Indeed, some of the least significant coefficients of the carrier image will be chaotically chosen by using our PRNG. These bits will be either switched or replaced by the bits of the watermark. The MSCs are only useful in case of authentication; mixture and embedding stages depend on them. Hence, a coefficient should not be defined at the same time as a MSC and a LSC: the last can be altered while the first is needed to extract the watermark.

(a) Lena.

(b) MSCs of Lena.



(c) LSCs of Lena ($\times 17$).

Fig. 10: Example of most and least significant coefficients of Lena.

2) Stages of the scheme: Our CI generator-based information hiding scheme consists of two stages: (1) mixture of the watermark and (2) its embedding.

a) Watermark mixture: Firstly, for security reasons, the watermark can be mixed before its embedding into the image. A first way to achieve this stage is to apply the bitwise exclusive or (XOR) between the watermark and the New CI generator. In this paper, we introduce a new mixture scheme based on chaotic iterations. Its chaotic strategy, which depends on our PRNG, will be highly sensitive to the MSCs, in the case of an authenticated watermarking.

b) Watermark embedding: Some LSCs will be switched, or substituted by the bits of the possibly mixed watermark. To choose the sequence of LSCs to be altered, a number of integers, less than or equal to the number M of LSCs corresponding to a chaotic sequence U, is generated from the chaotic strategy used in the mixture stage. Thus, the U^k-th least significant coefficient of the carrier image is either switched, or substituted by the k^{th} bit of the possibly mixed watermark. In case of authentication, such a procedure leads to a choice of the LSCs that are highly dependent on the MSCs [30].

On the one hand, when the switch is chosen, the watermarked image is obtained from the original image whose LSBs $L = \mathbb{B}^{\mathsf{M}}$ are replaced by the result of some chaotic iterations. Here, the iterate function is the vectorial Boolean negation,

$$f_0 : (x_1, ..., x_{\mathsf{M}}) \in \mathbb{B}^{\mathsf{M}} \longmapsto (\overline{x_1}, ..., \overline{x_{\mathsf{M}}}) \in \mathbb{B}^{\mathsf{M}}, \qquad (8)$$

the initial state is L, and the strategy is equal to U. In this case, the whole embedding stage satisfies the topological chaos properties [30], but the original medium is required to extract the watermark. On the other hand, when the selected LSCs are substituted by the watermark, its extraction can be done without the original cover (blind watermarking). In this case, the selection of LSBs still remains chaotic because of the use of the New CI generator, but the whole process does not satisfy topological chaos [30]. The use of chaotic iterations is reduced to the mixture of the watermark. See the following sections for more detail.

c) Extraction: The chaotic strategy can be regenerated even in the case of an authenticated watermarking, because the MSCs have not changed during the embedding stage. Thus, the few altered LSCs can be found, the mixed watermark can be rebuilt, and the original watermark can be obtained. In case of a switch, the result of the previous chaotic iterations on the watermarked image should be the original cover. The probability of being watermarked decreases when the number of differences increase.

If the watermarked image is attacked, then the MSCs will change. Consequently, in case of authentication and due to the high sensitivity of our PRNG, the LSCs designed to receive the watermark will be completely different. Hence, the result of the recovery will have no similarity with the original watermark.

The chaos-based data hiding scheme is summed up in Figure 11.

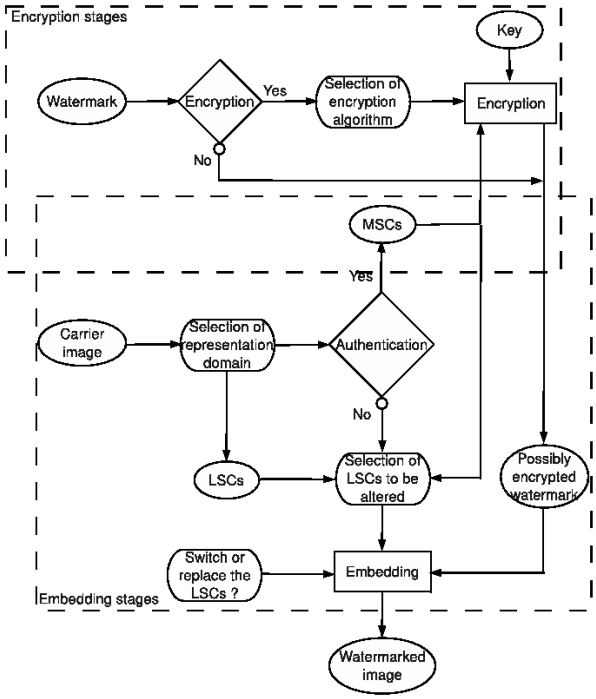


Fig. 11: The chaos-based data hiding decision tree.

C. Application Example

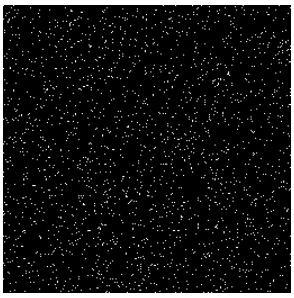
1) Experimental protocol: In this subsection, a concrete example is given: a watermark is encrypted and embedded into a cover image using the scheme presented in the previous section and CI(XORshift, XORshift). The carrier image is the well-known Lena, which is a 256 grayscale image, and the watermark is the 64×64 pixels binary image depicted in Figure 12.

The watermark is encrypted by using chaotic iterations: the initial state x^0 is the watermark, considered as a Boolean vector, the iteration function is the vectorial logical negation, and the chaotic strategy $(S^k)_{k\in\mathbb{N}}$ is defined with CI(XORshift, XORshift), where initial parameters constitute the secret key and $N = 64$. Thus, the encrypted watermark is the last

(a) The original image

(b) The watermark

Fig. 12: Original images



(a) Differences with the original

(b) Encrypted watermark

Fig. 13: Encrypted watermark and differences

Boolean vector generated by these chaotic iterations. An example of such an encryption is given in Figure 13.

Let L be the 256^3 Booleans vector constituted by the three last bits of each pixel of Lena and U^k defined by the sequence:

$$\begin{cases} U^0 &= S^0 \\ U^{n+1} &= S^{n+1} + 2 \times U^n + n \ [mod\ 256^3]. \end{cases} \quad (9)$$

The watermarked Lena I_w is obtained from the original Lena, whose three last bits are replaced by the result of 64^2 chaotic iterations with initial state L and strategy U (see Figure 13).

The extraction of the watermark can be obtained in the same way. Remark that the map $\theta \mapsto 2\theta$ of the torus, which is the famous dyadic transformation (a well-known example of topological chaos [13]), has been chosen to make $(U^k)_{k\leqslant 64^2}$ highly sensitive to the strategy. As a consequence, $(U^k)_{k\leqslant 64^2}$ is highly sensitive to the alteration of the image: any significant modification of the watermarked image will lead to a completely different extracted watermark, thus giving a way to authenticate media through the Internet.

Let us now evaluate the robustness of the proposed method.

2) Robustness evaluation: In what follows, the embedding domain is the spatial domain, CI(XORshift,XORshift) has been used to encrypt the watermark, MSCs are the four first bits of each pixel (useful only in case of authentication), and LSCs are the three next bits.

To prove the efficiency and the robustness of the proposed algorithm, some attacks are applied to our chaotic watermarked image. For each attack, a similarity percentage with the watermark is computed, this percentage is the number of equal bits between the original and the extracted watermark, shown as a percentage. Let us notice that a result less than or equal to 50% implies that the image has probably not been watermarked.

a) Zeroing attack: In this kind of attack, a watermarked image is zeroed, such as in Figure 14(a). In this case, the results in Table 1 have been obtained.

(a) Cropping attack

(b) Rotation attack

Fig. 14: Watermarked Lena after attacks.

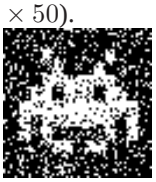
UNAUTHENTICATION		AUTHENTICATION	
Size (pixels)	Similarity	Size (pixels)	Similarity
10	99.08%	10	91.77%
50	97.31%	50	55.43%
100	92.43%	100	51.52%
200	70.75%	200	50.60%

Table. 1. Cropping attacks

In Figure 15, the decrypted watermarks are shown after a crop of 50 pixels and after a crop of 10 pixels, in the authentication case.

(a) Unauthentication (50×50).

(b) Authentication (50×50).



(c) Authentication (10×10).

Fig. 15: Extracted watermark after a cropping attack.

By analyzing the similarity percentage between the original and the extracted watermark, we can conclude that in case of unauthentication, the watermark still remains after a zeroing attack: the desired robustness is reached. It can be noticed that zeroing sizes and percentages are rather proportional.

In case of authentication, even a small change of the carrier image (a crop by 10×10 pixels) leads to a really different extracted watermark. In this case, any attempt to alter the carrier image will be signaled, the image is well authenticated.

b) Rotation attack: Let r_θ be the rotation of angle θ around the center $(128, 128)$ of the carrier image. So, the transformation $r_{-\theta} \circ r_\theta$ is applied to the watermarked image, which is altered as in Figure 14. The results in Table 2 have been obtained.

UNAUTHENTICATION		AUTHENTICATION	
Angle (degree)	Similarity	Angle (degree)	Similarity
2	96.44%	2	73.40%
5	93.32%	5	60.56%
10	90.68%	10	52.11%
25	78.13%	25	51.97%

Table. 2. Rotation attacks

TABLE III: SP 800-22 test results ($\mathbb{P}_T$)

Method	New CI ($m^n = y^n \ mod \ N$)	New CI (no mark)	Old CI	New CI ($g_1()$)	New CI ($g_2()$)
Frequency (Monobit) Test	0.0004	0.0855	0.595549	0.474986	0.419
Frequency Test within a Block	0	0	0.554420	0.897763	0.6786
Runs Test	0.2896	0.5544	0.455937	0.816537	0.3345
Longest Run of Ones in a Block Test	0.0109	0.4372	0.016717	0.798139	0.8831
Binary Matrix Rank Test	0	0.6579	0.616305	0.262249	0.7597
Discrete Fourier Transform (Spectral) Test	0	0	0.000190	0.007160	0.0008
Non-overlapping Template Matching Test*	0.020071	0.37333	0.532252	0.449916	0.51879
Overlapping Template Matching Test	0	0	0.334538	0.514124	0.2492
Maurer's "Universal Statistical" Test	0.6993	0.9642	0.032923	0.678686	0.1296
Linear Complexity Test	0.3669	0.924	0.401199	0.657933	0.3504
Serial Test* (m=10)	0	0.28185	0.013396	0.425346	0.2549
Approximate Entropy Test (m=10)	0	0.3838	0.137282	0.637119	0.7597
Cumulative Sums (Cusum) Test*	0	0	0.046464	0.279680	0.34245
Random Excursions Test*	0.46769	0.34788	0.503622	0.287409	0.18977
Random Excursions Variant Test*	0.28779	0.46505	0.347772	0.486686	0.26563
Success	8/15	11/15	15/15	15/15	15/15

The same conclusion as above can be declaimed: this watermarking method satisfies the desired properties.

c) JPEG compression: A JPEG compression is applied to the watermarked image, depending on a compression level. Let us notice that this attack leads to a change of the representation domain (from spatial to DCT domain). In this case, the results in Table 3 have been obtained.

UNAUTHENTICATION		AUTHENTICATION	
Compression	Similarity	Compression	Similarity
2	85.76%	2	56.42%
5	67.62%	5	52.12%
10	62.43%	10	48.22%
20	54.74%	20	49.07%

Table. 3. JPEG compression attacks

A very good authentication through JPEG attack is obtained. As for the unauthentication case, the watermark still remains after a compression level equal to 10. This is a good result if we take into account the fact that we use spatial embedding.

d) Gaussian noise: Watermarked image can be also attacked by the addition of a Gaussian noise, depending on a standard deviation. In this case, the results in Table 4 have been obtained.

UNAUTHENTICATION		AUTHENTICATION	
Standard dev.	Similarity	Standard dev.	Similarity
1	81.14%	1	55.57%
2	75.01%	2	52.63%
3	67.64%	3	52.68%
5	57.48%	5	51.34%

Table. 4. Gaussian noise attacks

Once again we remark that good results are obtained, especially if we keep in mind that a spatial representation domain has been chosen.

VIII. Conclusion and Future Work

In this paper, the pseudo-random generator proposed in [12] has been improved. By using XORshift instead of logistic map and due to a rewrite of the way to generate strategies, the generator based on chaotic iterations works faster and is more secure. The speed and randomness of this new PRNG has been compared to its former version, to XORshift, and to a generator based on logistic map. This comparison shows that CI(XORshift, XORshift) offers a sufficient speed and level of security for a wide range of Internet usages as cryptography and information hiding.

In future work, we will continue to try to improve the speed and security of this PRNG, by exploring new strategies and iteration functions. Its chaotic behavior will be deepened by using the numerous tools provided by the mathematical theory of chaos. New statistical tests will be used to compare this PRNG to existing ones. Additionally a probabilistic study of its security will be done. Lastly, new applications in computer science will be proposed, especially in the Internet security field.

References

[1] X. Tong and M. Cui, "Image encryption scheme based on 3d baker with dynamical compound chaotic sequence cipher generator," *Signal Processing*, vol. 89, no. 4, pp. 480 – 491, 2009.

[2] E. Erclebi and A. SubasI, "Robust multi bit and high quality audio watermarking using pseudo-random sequences," *Computers Electrical Engineering*, vol. 31, no. 8, pp. 525 – 536, 2005.

[3] P. L'ecuyer, "Comparison of point sets and sequences for quasi-monte carlo and for random number generation," *SETA 2008*, vol. LNCS 5203, pp. 1–17, 2008.

[4] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Reading, Mass, and third edition, Eds. Addison-Wesley, 1998.

[5] A. Marchi, A. Liverani, and A. D. Giudice, "Polynomial pseudo-random number generator via cyclic phase," *Mathematics and Computers in Simulation*, vol. 79, no. 11, pp. 3328–3338, 2009.

[6] S. Sachez, R. Criado, and C. Vega, "A generator of pseudo-random numbers sequences with a very long period," *Mathematical and Computer Modelling*, vol. 42, pp. 809 – 816, 2005.

[7] C. J. K. Tan, "The plfg parallel pseudo-random number generator," *Future Generation Computer Systems*, vol. 18, no. 5, pp. 693 – 698, 2002.

[8] M. Falcioni, L. Palatella, S. Pigolotti, and A. Vulpiani, "Properties making a chaotic system a good pseudo random number generator," *arXiv*, vol. nlin/0503035, 2005.

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

[10] Q. Wang, J. M. Bahi, C. Guyeux, and X. Fang, "Randomness quality of CI chaotic generators. application to internet security," in *INTERNET'2010. The 2nd Int. Conf. on Evolving Internet*. Valencia, Spain: IEEE seccion ESPANIA, Sep. 2010, pp. 125–130.

[11] J. M. Bahi and C. Guyeux, "Topological chaos and chaotic iterations, application to hash functions," *WCCI'10: 2010 IEEE World Congress on Computational Intelligence*, vol. Accepted paper, 2010.
[12] Q. Wang, C. Guyeux, and J. M. Bahi, "A novel pseudo-random generator based on discrete chaotic iterations for cryptographic applications," *INTERNET '09*, pp. 71–76, 2009.
[13] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, 2nd ed. Redwood City: Addison-Wesley, 1989.
[14] N. S. Publication, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Aug. 2008.
[15] F. Zheng, X. Tian, J. Song, and X. Li, "Pseudo-random sequence generator based on the generalized henon map," *The Journal of China Universities of Posts and Telecommunications*, vol. 15(3), pp. 64–68, 2008.
[16] G. Marsaglia, "Xorshift rngs," *Journal of Statistical Software*, vol. 8(14), pp. 1–6, 2003.
[17] P. M. Binder and R. V. Jensen, "Simulating chaotic behavior with finite-state machines," *Physical Review A*, vol. 34, no. 5, pp. 4460–4463, 1986.
[18] D. D. Wheeler, "Problems with chaotic cryptosystems," *Cryptologia*, vol. XIII, no. 3, pp. 243–250, 1989.
[19] J. Palmore and C. Herring, "Computer arithmetic, chaos and fractals," *Physica D*, vol. 42, pp. 99–110, 1990.
[20] M. Blank, "Discreteness and continuity in problems of chaotic dynamics," *Translations of Mathematical Monographs*, vol. 161, 1997.
[21] S. Li, G. Chen, and X. Mou, "On the dynamical degradation of digital piecewise linear chaotic maps," *Bifurcation an Chaos*, vol. 15, no. 10, pp. 3119–3151, 2005.
[22] F. Robert, *Discrete Iterations. A Metric Study*. Springer Series in Computational Mathematics, 1986, vol. 6.
[23] M. S. Turan, A. Doganaksoy, and S. Boztas, "On independence and sensitivity of statistical randomness tests," *SETA 2008*, vol. LNCS 5203, pp. 18–29, 2008.
[24] L. Kocarev, "Chaos-based cryptography: a brief overview," *IEEE Circ Syst Mag*, vol. 7, pp. 6–21, 2001.
[25] C. Guyeux, N. Friot, and J. M. Bahi, "Chaotic iterations versus spread-spectrum: chaos and stego security," in *IIH-MSP'10, 6-th Int. Conf. on Intelligent Information Hiding and Multimedia Signal Processing*, Darmstadt, Germany, Oct. 2010, pp. 208–211, to appear.
[26] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of applied cryptography*, Bocarton, Ed. CRC Press, 1997.
[27] "The frick collection, http://www.frick.org/," Last visit the 7th of June, 2011.
[28] "Delicious social bookmarking, http://delicious.com/," Last visit the 7th of June, 2011.
[29] Y. Nakashima, R. Tachibana, and N. Babaguchi, "Watermarked movie soundtrack finds the position of the camcorder in a theater," *IEEE Transactions on Multimedia*, 2009, accepted for future publication Multimedia.
[30] J. M. Bahi and C. Guyeux, "Topological chaos and chaotic iterations, application to hash functions," in *WCCI'10, IEEE World Congress on Computational Intelligence*. Barcelona, Spain: IEEE, Jul. 2010, pp. 1–7.

APPENDIX

The NIST Statistical Test Suite

In what follows, the objectives of the fifteen tests contained in the NIST Statistical tests suite are recalled. A more detailed description for those tests can be found in [14].

Frequency (Monobit) Test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

Frequency Test within a Block is to determine whether the frequency of ones in an M-bits block is approximately M/2, as would be expected under an assumption of randomness (M is the length of each block).

Runs Test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such zeros and ones is too fast or too slow.

Test for the Longest Run of Ones in a Block is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence.

Binary Matrix Rank Test is to check for linear dependence among fixed length substrings of the original sequence.

Discrete Fourier Transform (Spectral) Test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness.

Non-overlapping Template Matching Test is to detect generators that produce too many occurrences of a given non-periodic (aperiodic) pattern.

Overlapping Template Matching Test is the number of occurrences of pre-specified target strings.

Maurer's "Universal Statistical" Test is to detect whether or not the sequence can be significantly compressed without loss of information.

Linear Complexity Test is to determine whether or not the sequence is complex enough to be considered random.

Serial Test is to determine whether the number of occurrences of the 2^m m-bit (m is the length in bits of each block) overlapping patterns is approximately the same as would be expected for a random sequence.

Approximate Entropy Test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence (m is the length of each block).

Cumulative Sums (Cusum) Test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences.

Random Excursions Test is to determine if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence.

Random Excursions Variant Test is to detect deviations from the expected number of visits to various states in the random walk.