Suitability of chaotic iterations schemes using XORshift for security applications

Jacques Bahi, Xiole Fang, Christophe Guyeux, Qianxue Wang

Suitability of chaotic iterations schemes using XORshift for security applications

Jacques M. Bahi, Xiaole Fang, Christophe Guyeux, and Qianxue Wang*
University of Franche-Comté
FEMTO-ST Institute, UMR 6174 CRNS
Besançon, France
Email:{jacques.bahi, xiaole.fang, christophe.guyeux, qianxue.wang}@univ-fcomte.fr

***Abstract*—The design and engineering of original cryptographic solutions is a major concern to provide secure information systems. In a previous study, we have described a generator based on chaotic iterations, which uses the well-known XORshift generator. By doing so, we have improved the statistical performances of XORshift and make it behave chaotically, as defined by Devaney. The speed and security of this former generator have been improved in a second study, to make its usage more relevant in the Internet security context. In this paper, these contributions are summarized and a new version of the generator is introduced. It is based on a new Lookup Table implying a large improvement of speed. A comparison and a security analysis between the XORshift and these three versions of our generator are proposed, and various new statistical results are given. Finally, an application in the information hiding framework is presented, to give an illustrative example of the use of such a generator in the Internet security field.**

***Keywords*-pseudorandom number generators; Chaotic sequences; Statistical tests; Discrete chaotic iterations; Information hiding.**

I. Introduction

To use a pseudorandom number generator (PRNG) with a large level of security is necessary to satisfy the Internet security requirements to support activities as e-Voting, information hiding, and the protection of intellectual property [1], [2], [3]. This level depends on the proof of theoretical properties and results of numerous statistical tests. Many PRNGs, based for instance on linear congruential methods and feedback shift-registers [4], [5], [6], have been proven to be secure, following a probabilistic approach. More recently, several researchers have explored the idea of using chaotic dynamical systems to reinforce the security of these important tools [7], [8], [9]. 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.

This paper extends a study initiated in [10], [11], [12], in which we tried to fill this gap. In [12], it is proven that chaotic iterations (CIs), a suitable tool for fast computing iterative algorithms, satisfies the topological chaotic property, as it is defined by Devaney [13]. In [10] the chaotic behavior of CIs is exploited in order to obtain an unpredictable PRNG, which depends on two logistic maps. Lastly, in [11], a new version of this generator using decimations has been proposed and XORshift has replaced the logistic map. We have shown that, in addition of 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, a new version of this chaotic PRNG is introduced. It is based on a Lookup Table (LUT) method. After having introduced it, we will give a comparison of the speed, of the statistical properties, and of the security for all of these generators based on XORshift generator [15]. These results added to its chaotic properties allow us to consider that this new generator has good pseudorandom characteristics and is able to withstand attacks. After having presented the theoretical framework of the study and a security analysis, we will give a comparison based on new statistical tests. Finally a concrete example of how to use these pseudorandom numbers for information hiding through the Internet is detailed.

The remainder of this paper is organized in the following way. In Section II, some basic definitions concerning chaotic iterations and PRNGs are recalled. Then, the generator based on LUT discrete chaotic iterations is presented in Section III. In Section IV, various tests are passed to make a statistical comparison between this new PRNG and other existing ones. In the next sections, a potential use of this PRNG in some Internet security field is presented, namely in information hiding. The paper ends with a conclusion section where the contribution is summarized and intended future work is presented.

II. Review of Basics

A. Notations

$[\![1;N]\!] \rightarrow \{1,2,\ldots,N\}$
$S^n \rightarrow$ 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 $[\![1;N]\!]$
$\mathbb{S} \rightarrow$ the set of all strategies
$\mathbf{C}_n^k \rightarrow$ the binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
$\wedge \rightarrow$ the bitwise exclusive or

* Authors in alphabetic order

$+$ $\rightarrow$ the integer addition
$\ll$ and $\gg$ $\rightarrow$ the usual shift operators
$(\mathcal{X}, \mathrm{d})$ $\rightarrow$ a metric space
$\lfloor x \rfloor$ $\rightarrow$ 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 $\{1,2,3,...\}$
$\&$ $\rightarrow$ the bitwise AND

B. Chaotic iterations

Definition 1 The set $\mathbb{B}$ denoting $\{0, 1\}$, let $f : \mathbb{B}^N \longrightarrow \mathbb{B}^N$ be an "iteration" function and $S \in \mathbb{S}$ be a chaotic strategy. Then, the so-called *chaotic iterations* are defined by [16]:

$$\begin{cases} x^0 \in \mathbb{B}^N, \\ \forall n \in \mathbb{N}^*, \forall i \in [\![1; N]\!], x_i^n = \begin{cases} x_i^{n-1} & \text{if } S^n \neq i \\ f(x^{n-1})_{S^n} & \text{if } S^n = i. \end{cases} \end{cases} \quad (1)$$

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., [16].

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

Algorithm 1 An arbitrary round of the old CI(XORshift1,XORshift2) generator

$a \leftarrow XORshift1()$
$m \leftarrow a \ mod \ 2 + c$
while $i = 0, \ldots, m$ **do**
  $b \leftarrow XORshift2()$
  $S \leftarrow b \ mod \ N$
  $x_S \leftarrow \overline{x_S}$
end while
$r \leftarrow x$
Return r

C. Old CI(XORshift, XORshift) algorithm

The basic design procedure of the old CI generator [10] is recalled in Algorithm 1. The internal state is x (N bits), the output state is r (N bits), a and b are computed by two XORshift generators. Finally, N and $c \geqslant 3N$ are constants defined by the user.

D. New CI(XORshift, XORshift) algorithm

Algorithm 2 summarizes [11] the basic design procedure of the new generator. The internal state is x (a Boolean vector of size N), the output state is r (N bits). a and b are those computed by the two XORshifts. The value $f(a)$ is an integer, defined as in Equation 2. Lastly, N is a constant defined by the user.

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

Algorithm 2 An arbitrary round of the new CI(XORshift1,XORshift2) generator

1: **while** $i = 0, \ldots, N$ **do**
2: $d_i \leftarrow 0$
3: **end while**
4: $a \leftarrow XORshift1()$
5: $m \leftarrow f(a)$
6: $k \leftarrow m$
7: **while** $i = 0, \ldots, k$ **do**
8: $b \leftarrow XORshift2() \ mod \ N$
9: $S \leftarrow b$
10: **if** $d_S = 0$ **then**
11: $x_S \leftarrow \overline{x_S}$
12: $d_S \leftarrow 1$
13: **else if** $d_S = 1$ **then**
14: $k \leftarrow k + 1$
15: **end if**
16: **end while**
17: $r \leftarrow x$
18: Return r

III. LUT CI(XORshift,XORshift) algorithms and example

A. Introduction

The LUT CI generator is an improved version of the new CI generator. The key-ideas are:

- To use a Lookup Table for a faster generation of strategies. These strategies satisfy the same property than the ones provided by the decimation process.
- And to use all the bits provided by the two inputted generators (to discard none of them).

These key-ideas are put together by the following way.

Let us firstly recall that in chaotic iterations, only the cells designed by S^n−th are "iterated" at the n^{th} iteration. S^n can be either a component (*i.e.*, only one cell is updated at each iteration, so $S^n \in [\![1; N]\!]$) or a subset of components (any number of cells can be updated at each iteration, that is, $S^n \subset [\![1; N]\!]$). The first kind of strategies are called "unary strategies" whereas the second one are denoted by "general strategies". In the last case, each term S^n of the strategy can be represented by an integer lower than 2^N, designed by $\mathcal{S}^n$, for a system having N bits: the k^{th} component of the system is updated at iteration number n if and only if the k^{th} digit of the binary decomposition of $\mathcal{S}^n$ is 1. For instance, let us consider that $\mathcal{S}^n = 5$, and that we iterate on a system having 6 bits ($N = 6$). As the integer 5 has a binary

decomposition equal to 000101, we thus conclude that the cells number 1 and 3 will be updated when the system changes its state from x^n to x^{n+1}. In other words, in that situation, $\mathcal{S}^n = 5 \in [\![0, 2^6 - 1]\!] \Leftrightarrow S^n = \{1, 3\} \subset [\![1, 6]\!]$. To sum up, to provide a general strategy of $[\![1; N]\!]$ is equivalent to give an unary strategy in $[\![0; 2^N - 1]\!]$. Let us now take into account this remark.

Until now the proposed generators have been presented in this document by using unary strategies (obtained by the first inputted PRNG S) that are finally grouped by "packages" (the size of these packages is given by the second generator m): after having used each terms in the current package $S^{m^n}, ..., S^{m^{n+1}-1}$, the current state of the system is published as an output. Obviously, when considering the new CI version, these packages of unary strategies defined by the couple $(S, m) \in [\![1; N]\!] \times [\![0; N]\!]$ correspond to subsets of $[\![1; N]\!]$ having the form $\left\{S^{m^n}, ..., S^{m^{n+1}-1}\right\}$, which are general strategies. As stated before, these lasts can be rewritten as unary strategies that can be described as sequences in $[\![0; 2^N - 1]\!]$.

The advantage of such an equivalency is to reduce the complexity of the proposed PRNG. Indeed the new CI(S,m) generator can be written as:

$$x^n = x^{n-1} \wedge \mathcal{S}^n. \tag{3}$$

where $\mathcal{S}$ is the unary strategy (in $[\![0; 2^N - 1]\!]$) associated to the couple $(S, m) \in [\![1; N]\!] \times [\![0, N]\!]$.

The speed improvement is obvious, the sole issue is to understand how to change (S, m) by $\mathcal{S}$. The problem to consider is that all the sequences of $[\![0; 2^n - 1]\!]$ are not convenient. Indeed, the properties required for the couple (S, m) (S must not be uniformly distributed, and a cell cannot be changed twice between two outputs) must be translated in requirements for $\mathcal{S}$ if we want to satisfy both speed and randomness. Such constrains are solved by working on the sequence m and by using some well-defined Lookup Tables presented in the following sections.

B. Sequence m

In order to improve the speed of the proposed generator, the first plan is to take the best usage of the bits generated by the inputted PRNGs. The problem is that the PRNG generating the integers of m^n does not necessary takes its values into $[\![0, N]\!]$, where N is the size of the system.

For instance, in the new CI generator presented previously, this sequence is obtained by a XORshift, which produces integers belonging into $[\![0, 2^{32} - 1]\!]$. However, the iterated system has 4 cells ($N = 4$) in the example proposed previously thus, to define the sequence m^n, we compute the remainder modulo 4 of each integer provided by the XORshift generator. In other words, only the last 4 bits of each 32 bits vector generated by the second XORshift are used. Obviously this stage can be easily optimized, by splitting this 32-bits vector into 8 subsequences of 4 bits. Thus, a call of XORshift() will now generate 8 terms of the sequence m, instead of only one term in the former generator.

This common-sense action can be easily generalized to any size $N \leqslant 32$ of the system by the procedure described in Algorithm 3. The idea is simply to make a shift of the binary vector a produced by the XORshift generator, by 0, N, $2N$,... bits to the right, depending on the remainder c of n modulo $\lfloor N/32 \rfloor$ (that is, $a \gg (N \times c)$), and to take the bits between the positions $32 - N$ and 32 of this vector (corresponding to the right part "$\&(2^N - 1)$" of the formula). In that situation, all the bits provided by XORshift are used when N divide 32.

Algorithm 3 Generation of sequence b^n

```
c = n mod ⌊32/N⌋
if c = 0 then
    a ← XORshift()
end if
b^n ← (a ≫ (N × c))&(2^N − 1)
Return b^n
```

This Algorithm 3 produces a sequence $(b^n)_{n\in\mathbb{N}}$ of integers belonging into $[\![0, 2^N - 1]\!]$. It is now possible to define the sequence m by adapting the Equation 2 as follows.

$$m^n = f(b^n) = \begin{cases} 0 \text{ if } 0 \leqslant b^n < C_N^0, \\ 1 \text{ if } C_N^0 \leqslant b^n < \sum_{i=0}^{1} C_N^i, \\ 2 \text{ if } \sum_{i=0}^{1} C_N^i \leqslant b^n < \sum_{i=0}^{2} C_N^i, \\ \vdots \qquad \vdots \\ N \text{ if } \sum_{i=0}^{N-1} C_N^i \leqslant b^n < 2^N. \end{cases} \tag{4}$$

This common-sense measure can be improved another time if N is not very large by using the first Lookup Table of this document, which is called LUT-1. This improvement will be firstly explained through an example.

Let us consider that $N = 4$, so the sequence $(b^n)_{n\in\mathbb{N}}$ belongs into $[\![0, 15]\!]$. The function f of Equation 4 must translate each b^n into an integer $m^n \in [\![0, 4]\!]$, in such a way that the non-uniformity exposed previously is respected. Instead of defining the function f analytically, a table can be given containing all the images of the integers into $[\![0, 15]\!]$ (see Table I for instance). As stated before, the frequencies of occurrence of the images 0,1,2, 3, and 4 must be respectively equal to $\frac{C_4^0}{2^4}$, $\frac{C_4^1}{2^4}$, $\frac{C_4^2}{2^4}$, $\frac{C_4^3}{2^4}$, and $\frac{C_4^4}{2^4}$. This requirement is equivalent to demand C_N^i times the number i, which can be translated in terms of permutations. For instance, when $N = 4$, any permutation of the list [0,1,1,1,1,2,2,2,2,2,2,3,3,3,3,4] is convenient to define the image of [0,1,2,...,14,15] by f.

This improvement is implemented in Algorithm 4, which return a table $lut1$ such that $m^n = lut1[b^n]$.

Algorithm 4 The LUT-1 table generation

```
for j = 0...N do
    i = 0
    while i < C_N^j do
        lut1[i] = j
        i = i + 1
    end while
end for
Return lut1
```

TABLE I
A LUT-1 TABLE FOR $N = 4$

b^n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
m^n	0	1	1	1	1	2	2	2	2	2	2	3	3	3	3	4

C. Defining the chaotic strategy $\mathcal{S}$ with a LUT

The definition of the sequence m allows to determine the number of cells that have to change between two outputs of the LUT CI generator. There are C_N^m possibilities to change m bits in a vector of size N. As we have to choose between these C_N^m possibilities, we thus introduce the following sequence:

$$w^n = XORshift2() \ mod \ C_N^m \tag{5}$$

With this material it is now possible to define the LUT that provides convenient strategies to the LUT CI generator. If the size of the system is N, then this table has $N + 1$ columns, numbered from 0 to N. The column number m contains C_N^m values. All of these values have in common to present exactly m times the digit 1 and $N - m$ times the digit 0 in their binary decomposition. The order of appearance of these values in the column m has no importance, the sole requirement is that no column contains a same integer twice. Let us remark that this procedure leads to several possible LUTs.

Algorithm 5 $LUT21$ procedure

```
1: Procedure LUT21(M, N, b, v, c)
2: count ← c
3: value ← v
4: if count == M then
5:    lut2[M][num] = value
6:    num = num + 1
7: else
8:    for i = b....N do
9:       value = value + 2^i
10:      count = count + 1
11:      Call recurse LUT21(M, N, i + 1, value, count)
12:      value = v
13:      count = c
14:   end for
15: end if
16: End Procedure
```

An example of such a LUT is shown in Table III, when Algorithm 6 gives a concrete procedure to obtain such tables. This procedure makes recursive calls to the function $LUT21$ defined in Algorithm 5. The $LUT21$ uses the following variables. b is used to avoid overlapping computations between two recursive calls, v is to save the sum value between these calls, and c counts the number of cells that have already been processed. These parameters should be initialized as 0. For instance, the LUT presented in Table III is the $lut2$ obtained in Algorithm 5 with $N = 4$.

D. LUT CI(XORshift,XORshift) Algorithm

The LUT CI generator is defined by the following dynamical system:

$$x^n = x^{n-1} \wedge \mathcal{S}^n. \tag{6}$$

TABLE II
RESULTS OF DIEHARD BATTERY OF TESTS

No.	Test name	Generators			
		XORshift	old CI	new CI	LUT CI
1	Overlapping Sum	Pass	Pass	Pass	Pass
2	Runs Up 1	Pass	Pass	Pass	Pass
	Runs Down 1	Pass	Pass	Pass	Pass
	Runs Up 2	Pass	Pass	Pass	Pass
	Runs Down 2	Pass	Pass	Pass	Pass
3	3D Spheres	Pass	Pass	Pass	Pass
4	Parking Lot	Pass	Pass	Pass	Pass
5	Birthday Spacing	Pass	Pass	Pass	Pass
6	Count the ones 1	Pass	Pass	Pass	Pass
7	Binary Rank 6×8	Pass	Pass	Pass	Pass
8	Binary Rank 31×31	Pass	Pass	Pass	Pass
9	Binary Rank 32×32	Pass	Pass	Pass	Pass
10	Count the ones 2	Pass	Pass	Pass	Pass
11	Bit Stream	Pass	Pass	Pass	Pass
12	Craps Wins	Pass	Pass	Pass	Pass
	Throws	Pass	Pass	Pass	Pass
13	Minimum Distance	Pass	Pass	Pass	Pass
14	Overlapping Perm.	Pass	Pass	Pass	Pass
15	Squeeze	Pass	Pass	Pass	Pass
16	OPSO	Pass	Pass	Pass	Pass
17	OQSO	Pass	Pass	Pass	Pass
18	DNA	Pass	Pass	Pass	Pass
	Number of tests passed	18	18	18	18

Algorithm 6 LUT-2 generation

```
1: for i = 0....N do
2:    Call LUT21(i, N, 0, 0, 0)
3: end for
4: Return lut2
```

TABLE III
EXAMPLE OF A LUT FOR $N = 4$

w \ m	$m = 0$	$m = 1$	$m = 2$	$m = 3$	$m = 4$
$w = 0$	0	1	3	7	15
$w = 1$		2	5	11	
$w = 2$		4	6	13	
$w = 3$		8	9	14	
$w = 4$			10		
$w = 5$			12		

m	0	3	2	1
c	0	2	5	2
S	0	13	12	4
x^0	x^0	x^1	x^2	x^3
0	0	1	0	0
1	1	0	1	0
0	0	0	0	0
0	0	1	1	1

Binary Output: $x_1^0 x_2^0 x_3^0 x_4^0 x_1^1 x_2^1 x_3^1 x_4^1 x_1^2 x_2^2 ... = 0100100101010001...$
Integer Output: $x^0, x^1, x^2, x^3 ... = 4, 11, 8, 1...$

TABLE IV
EXAMPLE OF A LUT CI(XORSHIFT,XORSHIFT) GENERATION

where $x^O \in [\![0, 2^N - 1$ is a seed and $\mathcal{S}^n = lut2[w^n][m^n] = lut2[w^n][lut1[b^n]]$, in which b^n is provided by Algorithm 3 and $w^n = XORshift2()\ mod\ C_N^m$. An iteration of this generator is written in Algorithm 7. Let us finally remark that the two inputted XORshift can be replaced by any other operating PRNG.

Algorithm 7 LUT CI (XORshift,XORshift) algorithm

$c = n\ mod\ \lfloor 32/N \rfloor$
if $c = 0$ **then**
  $a \leftarrow XORshift1()$
end if
$b^n \leftarrow (a \gg (N \times c))\&(2^N - 1)$
$m^n = lut1[b^n]$
$d^n = XORshift2()$
$w^n = b^n\ mod\ C_N^m$
$S^n = lut2[m][w]$
$x = x \wedge S^n$
Return x

E. LUT CI(XORshift,XORshift) example of use

In this example, $N = 4$ is chosen another time for easy understanding. As before, the initial state of the system x^0 can be seeded by the decimal part t of the current time. With the same current time than in the examples exposed previously, we have $x^0 = (0, 1, 0, 0)$ (or $x^0 = 4$).

Algorithm 4 provides the LUT-1 depicted in Table I. The first XORshift generator has returned $y = 0, 11, 7, 2, 10, 4, 1, 0, 3, 9, ...$ By using this LUT, we obtain $m = 0, 3, 2, 1, 2, 1, 1, 0, 1, 2, ...$ Then the Algorithm 6 is computed, leading to the LUT-2 given by Table III.

So chaotic iterations of Algorithm 7 can be realized, to obtain in this example: 0100100101010001... or 4,9,5,1...

IV. STATISTICAL ANALYSIS

In order to make a fair comparison, we decided to choose the best parameters for each generator. According to the experiments, these values are $N = 4$ for the old CI, $N = 32$ for the new one, and finally $N = 8$ for the LUT CI generator (see Sections IV-A, IV-B, and IV-C respectively).

TABLE VI
COMPARISON BETWEEN THE PRESENTED PRNGS FOR A 2×10^8 BITS SEQUENCE

Methods	XORshift	Old CI	New CI	LUT CI
Monobit	0.6055	0.5689	0.0029	0.0471
Serial	0.7021	1.5765	0.3845	0.2232
Poker	7.957	6.3683	5.882	5.166
RunS	26.1022	28.4237	24.8094	21.9861
Autocorrelation	1.1628	0.3403	1.4220	0.4410
Time	9.33s	49.55s	28.82s	11.24s

A. NIST

In our experiments, 100 sequences (s = 100) of 1,000,000 bits are generated and tested. If the value $\mathbb{P}_T$ of any test is smaller than 0.0001, the sequences are considered to be not good enough and the generator is unsuitable. Table V shows $\mathbb{P}_T$ of 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 value is computed to characterize the statistics. We can see in Table V that old, new, and LUT CI(XORshift, XORshift) generators have successfully passed the NIST statistical test suite. In particular, the score of the XORshift generator is better when this last is embedded into any of the three proposed scheme (indeed, XORshift alone fails one of the NIST tests).

B. Diehard

Table II gives the results derived from applying the DieHARD battery [17] of tests to the PRNGs considered in this work. As it can be observed, all the generator presented in this document can pass the DieHARD battery of tests.

C. Comparative test parameters

We show in Table VI a comparison in comparative test parameters [11] among the generators LUT CI(XORshift, XORshift), New CI (XORshift, XORshift), their old version: Old CI(XORshift, XORshift), and a PRNG based on a simple XORshift. Time (in seconds) is related to the duration needed by each algorithm to generate a $2{\times}10^8$ 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 it as fair as possible. The results confirm that the proposed LUT CI is the fastest CI PRNG, while the statistical results is better for most of the parameters, leading to the conclusion that this new PRNGs is more secure than the other ones.

In addition, a comparison of overall stability from 5×10^4 to 8×10^5 for these generators has been given in Figure 1. It can be seen that LUT CI and new CI are dominant in all, especially when the sequences are very long.

D. Varying the output size

The size of the outputs (N, in number of bits) produced by each of the proposed generators only depend on the size of the initial state x^0. Moreover, as the "CI process" is fundamentally

TABLE V
NIST SP 800-22 TEST RESULTS ($\mathbb{P}_T$)

Method	XORshift	old CI	new CI	LUT CI
Frequency (Monobit)	0.779188	0.145326	0.719747	0.657933
Frequency within a Block	0.779188	0.028817	0.071177	0.719747
Runs	0.514124	0.739918	0.911413	0.224821
Longest Run of Ones in a Block	0.883171	0.554420	0.779188	0.494392
Binary Matrix Rank	0.851383	0.236810	0.924076	0.023545
Discrete Fourier Transform (Spectral)	0.834308	0.514124	0.911413	0.514124
Non-overlapping Template Matching*	0.506389	0.512363	0.501621	0.437726
Overlapping Template Matching	0.534146	0.595549	0.275709	0.017912
Maurers Universal Statistical	0.366918	0.122325	0.419021	0.897763
Linear Complexity	0.275709	0.249284	0.779188	0.678686
Serial* (m=10)	0.328499	0.495847	0.933624	0.444265
Approximate Entropy (m=10)	0.000000	0.051942	0.262249	0.319084
Cumulative Sums (Cusum) *	0.720350	0.074404	0.368618	0.171384
Random Excursions *	0.396803	0.507812	0.518462	0.356105
Random Excursions Variant *	0.576643	0.289594	0.548078	0.587062
Success	14/15	15/15	15/15	15/15

a negation of bits, the size of the system does not really impact the speed of these PRNGs, at least for reasonable values of N. As various N values can be relevant, depending on the application, we thus investigate whether the statistical performances of the CI generators are impacted when N changes.

We can show in Table VII that, for the three CI generators, various N lead to success for both the NIST and DIEHARD tests. Concerning the whole TestU01 [18], various consequences can be dressed. Firstly, the LUT CI generator is unsuitable for $N = 32$, due to its too large consumption of memory resources when generating and using the LUTs. Secondly, this last generator is the only one capable to pass the whole TestU01 with only $N = 4$ cells. Finally, all the proposed generators have better scores than the XORshift they use.

E. Devaney's chaos property

Generally, 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 sequences fit the requirements of pseudorandom sequences well. Contrary to XORshift, our generator possesses these chaotic properties [12],[10]. However, despite a large number of papers published in the field of chaos-based pseudorandom 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. These major problems restrict their use as generators [19].

In this paper we don't 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 [12] 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.

V. APPLICATION EXAMPLE IN DIGITAL WATERMARKING

Information hiding has recently become a major information security technology, especially with the increasing importance and widespread distribution of digital media through the Internet [20]. It includes several techniques like digital watermarking. The aim of digital watermarking is to embed a piece of information into digital documents, such as pictures or movies. This is for a large panel of reasons, such as: copyright protection, control utilization, data description, content authentication, and data integrity. For these reasons, many different watermarking schemes have been proposed in recent years. Digital watermarking must have essential characteristics, including: security, imperceptibility, and robustness. Chaotic methods have been proposed to encrypt the watermark before embedding it in the carrier image for these security reasons. In this paper, a watermarking algorithm based on the chaotic PRNG presented above is given, as an illustration of use of this family of CI PRNG.

A. Most and least significant coefficients

Let us first introduce the definitions of most and least significant coefficients.

TABLE VII
TESTU01 STATISTICAL TEST

PRNG	Battery	Parameters	Statistics	N=4	N=8	N=16	N=32
	Rabbit	32×10^9 bits	40	-	-	-	3
	Alphabit	32×10^9 bits	17	-	-	-	0
	Pseudo DieHARD	Standard	126	-	-	-	3
Single	FIPS_140_2	Standard	16	-	-	-	0
XORshift	Small Crush	Standard	15	-	-	-	1
	Crush	Standard	144	-	-	-	29
	Big Crush	Standard	160	-	-	-	44
	Number of failures		518	-	-	-	80
	Rabbit	32×10^9 bits	40	1	2	2	3
	Alphabit	32×10^9 bits	17	0	0	2	2
	Pseudo DieHARD	Standard	126	0	0	0	0
Old CI	FIPS_140_2	Standard	16	0	0	0	0
	Small Crush	Standard	15	0	0	1	0
	Crush	Standard	144	2	9	16	46
	Big Crush	Standard	160	3	18	30	78
	Number of failures		518	6	29	51	129
	Rabbit	32×10^9 bits	40	0	0	0	0
	Alphabit	32×10^9 bits	17	0	0	0	0
	Pseudo DieHARD	Standard	126	2	0	0	0
New CI	FIPS_140_2	Standard	16	0	0	0	0
	Small Crush	Standard	15	0	0	0	0
	Crush	Standard	144	0	0	0	0
	Big Crush	Standard	160	0	0	0	0
	Number of failures		518	2	0	0	0
	Rabbit	32×10^9 bits	40	0	0	0	-
	Alphabit	32×10^9 bits	17	0	0	0	-
	Pseudo DieHARD	Standard	126	0	0	0	-
LUT CI	FIPS_140_2	Standard	16	0	0	0	-
	Small Crush	Standard	15	0	0	0	-
	Crush	Standard	144	0	0	0	-
	Big Crush	Standard	160	0	0	0	-
	Number of failures		518	0	0	0	-

Definition 2 For a given image, the most significant coefficients (in short MSCs), are coefficients that allow the description of the relevant part of the image, *i.e.* its most rich part (in terms of embedding information), through a sequence of bits.

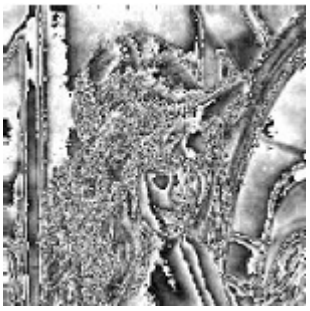
For example, in a spatial description of a grayscale image, a definition of MSCs can be the sequence constituted by the first three bits of each pixel as shown in Figure 2(b). In a discrete cosine frequency domain description, each 8×8 block of the carrier image is mapped to a list of 64 coefficients. The energy of the image is contained in the first of them. After binary conversion, the first fourth coefficients of all these blocks can constitute a possible sequence of MSCs.

Definition 3 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, in the case of a spatial domain watermarking of a grayscale image, as in Figure 2(c).

(a) Lena

(b) MSCs of Lena



(c) LSCs of Lena

Fig. 2. Spatial MSCs and LSCs of Lena

Discrete cosine, Fourier, and wavelet transform can be used to define LSCs and MSCs, in the case of frequency domain watermarking, among other possible choices. Moreover, these definitions are not limited to image media, but can easily be extended to the audio and video media as well.
LSCs are used during the embedding stage: some of the least significant coefficients of the carrier image will be chaotically chosen and replaced by the bits of the mixed watermark. With a large number of LSCs, the watermark can be inserted more than once and thus the embedding will be more secure and robust, but also more detectable.
The MSCs are only useful in the case of authentication: encryption 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.

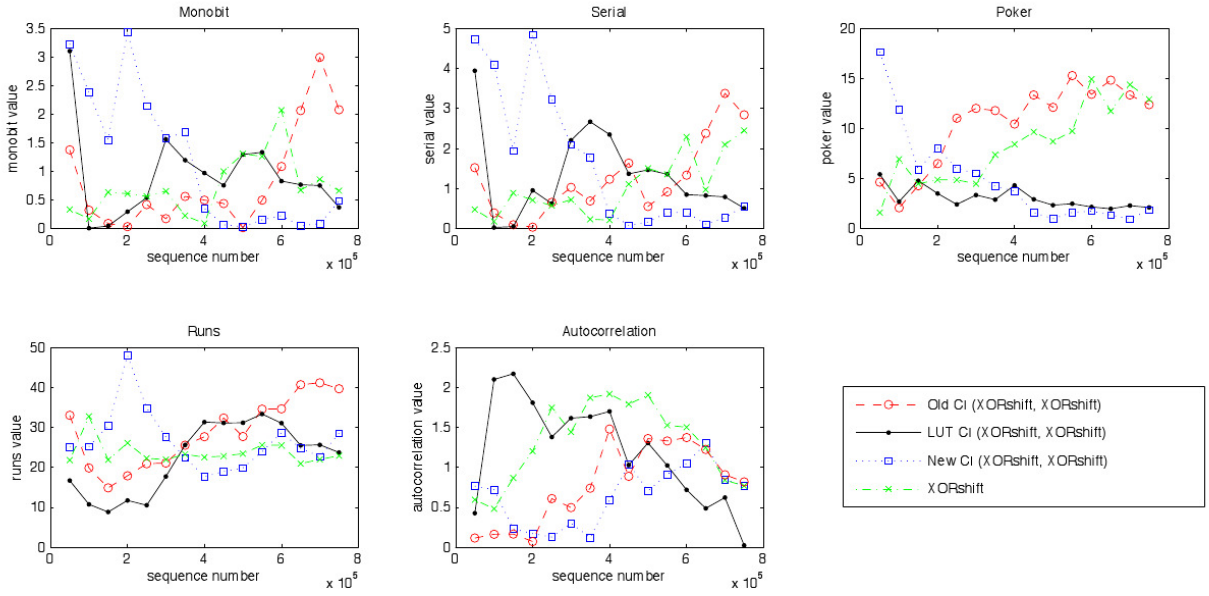


Fig. 1. Overall Sequence Stability Comparison

B. Stages of the algorithm

Our watermarking scheme consists of two stages: (1) mixture of the watermark and (2) its embedding.

1) Watermark mixture: Firstly, for safety reasons, the watermark can be mixed before its embedding into the image. A common way to achieve this stage is to use the bitwise exclusive or (XOR), for example, between the watermark and the above PRNG. In this paper, we will use another mixture scheme based on chaotic iterations. Its chaotic strategy, defined with our PRNG, will be highly sensitive to the MSCs, in the case of an authenticated watermark, as stated in [12].

2) Watermark embedding: Some LSCs will be substituted by all 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 N of LSCs corresponding to a chaotic sequence $\left(U^k\right)_k$, is generated from the chaotic strategy used in the mixture stage. Thus, the U^k-th least significant coefficient of the carrier image is substituted by the k^{th} bit of the possibly mixed watermark. In the case of authentication, such a procedure leads to a choice of the LSCs which are highly dependent on the MSCs. For the detail of this stage see Section VI-A2.

3) Extraction: The chaotic strategy can be regenerated, even in the case of an authenticated watermarking because the MSCs have not been changed during the stage of embedding the watermark. Thus, the few altered LSCs can be found, the mixed watermark can then be rebuilt, and the original watermark can be obtained. If the watermarked image is attacked, then the MSCs will change. Consequently, in the case of authentication and due to the high sensitivity of the embedding sequence, 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: authentication is reached.

VI. Evaluation of the Proposed Scheme

In this section, a complete application example of the above chaotic watermarking method is given and its robustness to some attacks is studied. This case study enables us to precise the details of the algorithm and evaluate it.

A. Stages and details

1) Images description: Carrier image is Lena, a 256 grayscale image of size 256×256 (see Figure 2(a)). The watermark is the 64×64 pixels binary image depicted in Figure 3(a). The embedding domain will be the spatial domain. The selected MSCs are the four most significant bits of each pixel and the LSCs are the three last bits (a given pixel will at most be modified of four levels of gray by an iteration). Before its embedment, the watermark is mixed with chaotic iterations. The system to iterate, chaotic strategy S^n and iterate function are defined below.

2) Embedding of the watermark: To embed the watermark, the sequence $(U^k)_{k\in\mathbb{N}}$ of altered bits taken from the M LSCs must be defined. To do so, the strategy $(S^k)_{k\in\mathbb{N}}$ of the encryption stage is used as follows:

$$\left\{\begin{array}{lcl} U^0 & = & S^0 \\ U^{n+1} & = & S^{n+1} + 2 \times U^n + n \ (mod\ \mathsf{M}) \end{array}\right. \tag{7}$$

to obtain the result depicted in Figure 4(b). The map $\theta \mapsto 2\theta$ of the torus, which is a famous example of topological Devaney's chaos [13], has been chosen to make $(U^k)_{k\in\mathbb{N}}$ highly sensitive to the chaotic strategy $(S^k)_{k\in\mathbb{N}}$. As a consequence, $(U^k)_{k\in\mathbb{N}}$ is

(a) Watermark

(b) Watermarked Lena

(c) Differences with original

Fig. 3. Watermarked Lena and differences

highly sensitive to the alteration of the MSCs. In case of authentication, any significant modification of the watermarked image will lead to a completely different extracted watermark.

B. Robustness results

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

1) Cropping attack: In this kind of attack, a watermarked image is cropped. In this case, the results in Table VIII have been obtained. In Figure 4, the decrypted watermarks are shown after a crop of 50 pixels and after a crop of 10 pixels, in the authentication case.

(a) Unauthentication (10×10)

(b) Authentication (10×10)

(c) Unauthentication (50×50)

Fig. 4. Extracted watermark after a cropping attack (zoom $\times 2$)

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

TABLE VIII
ROBUSTNESS AGAINS ATTACKS

Attacks	UNAUTHENTICATION		AUTHENTICATION	
Cropping	Size (pixels)	Similarity	Size (pixels)	Similarity
	10	99.48%	10	49.68%
	50	97.63%	50	54.54%
	100	91.31%	100	52.24%
	200	68.56%	200	51.87%
Rotation	Angle (degree)	Similarity	Angle (degree)	Similarity
	2	97.41%	2	70.01%
	5	94.67%	5	59.47%
	10	91.30%	10	54.51%
	25	80.85%	25	50.21%
JPEG compression	Compression	Similarity	Compression	Similarity
	2	82.95%	2	54.39%
	5	65.23%	5	53.46%
	10	60.22%	10	50.14%
	20	53.17%	20	48.80%
Gaussian noise	Standard dev.	Similarity	Standard dev.	Similarity
	1	74.26%	1	52.05%
	2	63.33%	2	50.95%
	3	57.44%	3	49.65%

2) 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. The results in Table VIII have been obtained. The same conclusion as above can be declaimed.

3) JPEG compression: A JPEG compression is applied to the watermarked image, depending on a compression level. This attack leads to a change of the representation domain (from spatial to DCT domain). In this case, the results in Table VIII have been obtained, illustrating a good authentication through JPEG attack. 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.

4) Gaussian noise: A 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 VIII are obtained.

C. Security study of the proposed information hiding scheme

For the sake of completeness, and to show the effectiveness of the method, we will now introduce two other strategies different from the one given in Eq. 7. The proposed scheme will be rewritten too, in order to give a more theoretical

evaluation of the security of the proposed information hiding algorithm.

1) Reformulation of the scheme: Let us consider the phase space $\mathcal{X} = [\![1;\mathsf{N}]\!]^{\mathbb{N}} \times \mathbb{B}^{\mathsf{N}}$ and the map $G_f(S,E) = \Big(\sigma(S), F_f(i(S),E)\Big)$, where σ is defined by $\sigma : (S^n)_{n\in\mathbb{N}} \in \mathbb{S} \to (S^{n+1})_{n\in\mathbb{N}} \in \mathbb{S}$, and i is the map $i : (S^n)_{n\in\mathbb{N}} \in \mathbb{S} \to S^0 \in [\![1;\mathsf{N}]\!]$. Using this rewriting of the chaotic iterations presented previously, let:

- $(K,N) \in [0;1] \times \mathbb{N}$ be an embedding key,
- $X \in \mathbb{B}^{\mathsf{N}}$ be the N least significant coefficients (LSCs) of a given cover media C,
- $(S^n)_{n\in\mathbb{N}} \in [\![1,\mathsf{N}]\!]^{\mathbb{N}}$ be a strategy, which depends on the message to hide $M \in [0;1]$ and K,
- $f_0 : \mathbb{B}^{\mathsf{N}} \to \mathbb{B}^{\mathsf{N}}$ be the vectorial logical negation.

So the watermarked media is C whose LSCs are replaced by $Y_K = X^N$, where [21]:

$$\begin{cases} X^0 = X \\ \forall n < N, X^{n+1} = G_{f_0}(X^n). \end{cases}$$

2) New examples of strategies:

a) CIIS strategy: Let us first introduce the Piecewise Linear Chaotic Map (PLCM, see [22]), defined by:

Definition 4 (PLCM)

$$F(x,p) = \begin{cases} x/p & \text{if} \quad x \in [0;p] \\ (x-p)/(\frac{1}{2}-p) & \text{if} \quad x \in \left[p;\frac{1}{2}\right] \\ F(1-x,p) & \text{else.} \end{cases}$$

where $p \in \left]0;\frac{1}{2}\right[$ is a "control parameter". Then, we can define the general term of the strategy $(S^n)_n$ in Chaotic Iterations with Independent Strategy (CIIS) setup by the following expression: $S^n = \lfloor \mathsf{N} \times K^n \rfloor + 1$, where:

$$\begin{cases} p \in \left[0;\frac{1}{2}\right] \\ K^0 = M \otimes K \\ K^{n+1} = F(K^n,p), \forall n \leq N_0 \end{cases}$$

in which $\otimes$ denotes the bitwise exclusive or (XOR) between two floating part numbers (*i.e.*, between their binary digits representation). Lastly, to be certain to enter into the chaotic regime of PLCM [22], the strategy can be preferably defined by: $S^n = \left\lfloor \mathsf{N} \times K^{n+D} \right\rfloor + 1$, where $D \in \mathbb{N}$ large enough: we thus iterate the PLCM a certain number of times before taking terms of the strategy.

b) CIDS strategy: The same notations as above are used. We define Chaotic Iterations with Dependent Strategy (CIDS) strategy as follows: $\forall k \leqslant N$,

- if $k \leqslant \mathsf{N}$ and $X^k = 1$, then $S^k = k$,
- else $S^k = 1$.

In this situation, if $N \geqslant \mathsf{N}$, then only two watermarked contents are possible with the scheme proposed previously, namely: $Y_K = (0,0,\cdots,0)$ and $Y_K = (1,0,\cdots,0)$. Indeed, in CIIS, the strategy is independent from the cover media X, whereas in CIDS the strategy will be dependent on X.

3) Evaluation of the stego-security: Let $\mathbb{K}$ be the set of embedding keys, $p(X)$ the probabilistic model of N_0 initial host contents, and $p(Y|K_1)$ the probabilistic model of N_0 watermarked contents. We suppose that each host content has been watermarked with the same key K_1 and the same embedding function e.

Definition 5 The embedding function e is stego-secure if and only if [23]: $\forall \mathbf{K_1} \in \mathbb{K}, \mathbf{p}(\mathbf{Y}|\mathbf{K_1}) = \mathbf{p}(\mathbf{X})$.

Let us now study the stego-security of the scheme. We will prove that,

Proposition 1 *The information hiding scheme using the CIIS strategy is stego-secure, whereas CIDS is not stego-secure.*

Proof: Let us suppose that $X \sim \mathbf{U}\left(\mathbb{B}^N\right)$ in a CIIS setup. We will prove by a mathematical induction that $\forall n \in \mathbb{N}, X^n \sim \mathbf{U}\left(\mathbb{B}^N\right)$. The base case is immediate, as $X^0 = X \sim \mathbf{U}\left(\mathbb{B}^N\right)$. Let us now suppose that the statement $X^n \sim \mathbf{U}\left(\mathbb{B}^N\right)$ holds for some n. Let $e \in \mathbb{B}^N$ and $\mathbf{B}_k = (0,\cdots,0,1,0,\cdots,0) \in \mathbb{B}^N$ (the digit 1 is in position k). So $P\left(X^{n+1} = e\right) = \sum_{k=1}^N P(X^n = e + \mathbf{B}_k, S^n = k)$. These two events are independent in CIIS setup, thus: $P\left(X^{n+1} = e\right) = \sum_{k=1}^N P(X^n = e + \mathbf{B}_k) \times P(S^n = k)$. According to the inductive hypothesis: $P\left(X^{n+1} = e\right) = \frac{1}{2^N} \sum_{k=1}^N P(S^n = k)$. The set of events $\{S^n = k\}$ for $k \in [\![1;N]\!]$ is a partition of the universe of possible, so $\sum_{k=1}^N P(S^n = k) = 1$.

Finally, $P\left(X^{n+1} = e\right) = \frac{1}{2^N}$, which leads to $X^{n+1} \sim \mathbf{U}\left(\mathbb{B}^N\right)$. This result is true $\forall n \in \mathbb{N}$, we thus have proven that,

$$\forall K \in [0;1], Y_K = X^{N_0} \sim \mathbf{U}\left(\mathbb{B}^N\right) \text{ when } X \sim \mathbf{U}\left(\mathbb{B}^N\right),$$

which concludes the first claim of the proposition. Let us now prove the second part of it.

Due to the definition of CIDS, we have $P(Y_K = (1,1,\cdots,1)) = 0$. So there is no uniform repartition for the stego-contents Y_K. ■

VII. Conclusion and future work

In this paper, the pseudorandom generator proposed in our previous works has been improved in terms of speed and randomness. By using some well-defined Lookup Tables 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 LUT CI PRNG has been compared to its former versions and to XORshift. This comparison shows that LUT CI(XORshift, XORshift) offers a sufficient speed and level of security for a whole range of Internet usages as cryptography and data hiding. This generator has been used to develop a scheme in the information hiding domain, whose robustness and security has been detailed in the previous section. Further readings about the security of such a chaos-based watermarking scheme can be found in, *e.g.*, [24], [25].

In future work, we will continue to explore new strategies and iteration functions. Its chaotic behavior will be deepened by using the various 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] J. M. Bahi and C. Guyeux, "A new chaos-based watermarking algorithm," in *SECRYPT 2010, International conference on security and cryptography*, Athens, Greece, July 2010, pp. 1–4, to appear.

[2] S. Liu, H. Yao, W. Gao, and Y. Liu, "An image fragile watermark scheme based on chaotic image pattern and pixel-pairs," *Applied Mathematics and Computation*, vol. 185, pp. 869–882, 2007.

[3] X. Yi and E. Okamoto, "Practical internet voting system," *Journal of Network and Computer Applications*, no. 0, pp. –, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S108480451200135X

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

[5] 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.

[6] M. Blaszczyk and R. Guinee, "Experimental validation of a true random binary digit generator fusion with a pseudo random number generator for cryptographic module application," *IET Conference Publications*, vol. 2009, no. CP559, pp. 31–31, 2009. [Online]. Available: http://link.aip.org/link/abstract/IEECPS/v2009/iCP559/p31/s1

[7] 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.

[8] 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.

[9] S. J. Li, X. Q. Mou, and Y. L. Cai, "Pseudo-random bit generator based on couple chaotic systems and its applications in stream-cipher cryptography," *Proceedings of 2nd International Conference on Cryptology*, vol. 2247, pp. 316–329, 2001.

[10] J. Bahi, C. Guyeux, and Q. Wang, "A novel pseudo-random generator based on discrete chaotic iterations," in *INTERNET'09, 1-st Int. Conf. on Evolving Internet*, Cannes, France, Aug. 2009, pp. 71–76. [Online]. Available: http://dx.doi.org/10.1109/INTERNET.2009.18

[11] 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.

[12] 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.

[13] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, 2nd ed. Redwood City: Addison-Wesley, 1989.

[14] NIST Special Publication 800-22 rev1a, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," April 2010.

[15] G. Marsaglia, "Xorshift rngs," *Journal of Statistical Software*, vol. 8(14), pp. 1–6, 2003.

[16] F. Robert, *Discrete Iterations. A Metric Study*. Springer Series in Computational Mathematics, 1986, vol. 6.

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

[18] R. Simard and U. D. Montral, "Testu01: A software library in ansi c for empirical testing of random number generators. software users guide," 2002.

[19] L. Kocarev, "Chaos-based cryptography: a brief overview," *IEEE Circ Syst Mag*, vol. 7, pp. 6–21, 2001.

[20] X. Wu and Z. Guan, "A novel digital watermark algorithm based on chaotic maps," *Physical Letters A*, vol. 365, pp. 403—-406, 2007.

[21] N. Friot, C. Guyeux, and J. Bahi, "Chaotic iterations for steganography - stego-security and chaos-security," in *SECRYPT'2011, Int. Conf. on Security and Cryptography. SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*, J. Lopez and P. Samarati, Eds. Sevilla, Spain: SciTePress, Jul. 2011, pp. 218–227.

[22] L. Shujun, L. Qi, L. Wenmin, M. Xuanqin, and C. Yuanlong, "Statistical properties of digital piecewise linear chaotic maps and their roles in cryptography and pseudo-random coding," *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, vol. 1, pp. 205–221, 2001.

[23] F. Cayre, C. Fontaine, and T. Furon, "Kerckhoffs-based embedding security classes for woa data hiding," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 1, pp. 1–15, 2008.

[24] J. Bahi, N. Friot, and C. Guyeux, "Lyapunov exponent evaluation of a digital watermarking scheme proven to be secure," in *IIH-MSP'2012, 8-th Int. Conf. on Intelligent Information Hiding and Multimedia Signal Processing*. Piraeus-Athens, Greece: IEEE Computer Society, Jul. 2012, pp. 359–362. [Online]. Available: http://dx.doi.org/10.1109/IIH-MSP.2012.93

[25] J. Bahi, J.-F. Couchot, and C. Guyeux, "Steganography: a class of algorithms having secure properties," in *IIH-MSP-2011, 7-th Int. Conf. on Intelligent Information Hiding and Multimedia Signal Processing*, Dalian, China, Oct. 2011, pp. 109–112.