A Pseudo Random Numbers Generator Based on Chaotic Iterations. Application to Watermarking

Christophe Guyeux, Qianxue Wang, and Jacques M. Bahi

University of Franche-Comte, Computer Science Laboratory LIFC,
25030 Besançon Cedex, France
{christophe.guyeux,qianxue.wang,
jacques.bahi}@univ-fcomte.fr

Abstract. In this paper, a new chaotic pseudo-random number generator (PRNG) is proposed. It combines the well-known ISAAC and XORshift generators with chaotic iterations. This PRNG possesses important properties of topological chaos and can successfully pass NIST and TestU01 batteries of tests. This makes our generator suitable for information security applications like cryptography. As an illustrative example, an application in the field of watermarking is presented.

Keywords: Internet Security; Chaotic Sequences; Statistical Tests; Discrete Chaotic Iterations; Watermarking.

1 Introduction

The extremely fast development of the Internet brings growing attention to information security issues. Among these issues, the conception of pseudo-random number generators (PRNGs) plays an important role. Secure PRNGs which can be easily implemented with simple software routines are desired. Due to the finiteness of the set of machine numbers, the sequences generated by numerous existing PRNGs are not actually random. For example, the use of stringent batteries of tests allows us to determine whether these sequences are predictable. Chaos theory plays an active role in the improvement of the quality of PRNGs [5], [14]. The advantage of using chaos in this field lies in its disordered behavior and its unpredictability.

This paper extends the study initiated in [3] and [17]. In [3], it is proven that chaotic iterations (CIs), a suitable tool for fast computing iterative algorithms, satisfy the topological chaotic property, as it is defined by Devaney [7]. In [17], the chaotic behavior of CIs is exploited in order to obtain an unpredictable behavior for a new PRNG. This generator is based on chaotic iterations and depends on two other input sequences. These two sequences are generated by two logistic maps. Our generator has successfully passed the NIST (National Institute of Standards and Technology of the U.S. Government) battery of tests. However it appeared that it is a slow generator and it can't pass TestU01 because of the input logistic maps. Moreover this logistic map has revealed serious security lacks, which make it use inadequate for cryptographic applications [1]. That is why, in this paper, we intend to develop a new fast PRNG. It will pass TestU01, widely considered as the most comprehensive and stringent battery of tests. This goal is achieved by using the ISAAC and XORshift maps in place of the two logistic maps. Chaotic properties, statistical tests and security analysis [19] allow us to consider that this generator has good pseudo-random characteristics and is capable to withstand attacks.

The rest of this paper is organized in the following way: in Section 2, some basic definitions concerning chaotic iterations and PRNGs are recalled. Then, the generator based on discrete chaotic iterations is presented in Section 3. Section 4 is devoted to its security analysis. In Section 5, we show that the proposed PRNG passes the TestU01 statistical tests. In Section 6 an application in the field of watermarking is proposed. The paper ends by a conclusion and some discussions about future work.

2 Basic recalls

This section is devoted to basic notations and terminologies in the fields of chaotic iterations and PRNGs.

2.1 Notations

$[\![1;\mathsf{N}]\!]$	→ $\{1, 2, \dots, N\}$
S^n	→ the n^{th} term of a sequence $S = (S^1, S^2, \dots)$
v_i	→ the i^{th} component of a vector $v = (v_1, v_2, \dots, v_n)$
f^k	→ k^{th} composition of a function f
strategy	→ a sequence which elements belong in $[\![1;\mathsf{N}]\!]$
$\mathbb{S}$	→ the set of all strategies
$\oplus$	→ bitwise exclusive or
$+$	→ the integer addition
$\ll$ and $\gg$	→ the usual shift operators

2.2 Chaotic iterations

Definition 1. *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 *are defined by [16]*

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

In other words, at the n^{th} iteration, only the S^n-th cell is "iterated".

2.3 Input sequences

In [17], we have designed a PRNG which has successfully passed the NIST tests suite. Unfortunately, this PRNG is too slow to pass the TestU01 battery of tests. Our ancient

PRNG which is called CI(Logistic, Logistic) PRNG is based on chaotic iterations and uses logistic maps as input sequences. However, chaotic systems like logistic maps work in the real numbers domain, and therefore a transformation from real numbers into integers is needed. This process leads to a degradation of the chaotic behavior of the generator and a lot of time wasted during computations. Moreover, a recent study shows that the use of logistic map for cryptographic applications is inadequate and must be discouraged [1]. Our purpose is then to design a new, faster, and more secure generator, which is able to pass the TestU01 battery of tests. This is achieved by using some faster PRNGs like ISAAC [9] and XORshift [13] as input sequences.

3 Design of CI(ISAAC,XORshift)

3.1 Chaotic iterations as PRNG

The novel generator is designed by the following process. Let $\mathsf{N} \in \mathbb{N}^*, \mathsf{N} \geqslant 2$. Some chaotic iterations are fulfilled to generate a sequence $(x^n)_{n\in\mathbb{N}} \in \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}}$ of boolean vectors: the successive states of the iterated system. Some of these vectors are randomly extracted and their components constitute our pseudo-random bit flow. Chaotic iterations are realized as follows. Initial state $x^0 \in \mathbb{B}^{\mathsf{N}}$ is a boolean vector taken as a seed and chaotic strategy $(S^n)_{n\in\mathbb{N}} \in [\![1, \mathsf{N}]\!]^{\mathbb{N}}$ is constructed with XORshift. Lastly, iterate function f is the vectorial boolean negation

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

To sum up, at each iteration only S^i-th component of state X^n is updated, as follows

$$x_i^n = \begin{cases} x_i^{n-1} \text{ if } i \neq S^i, \\ \overline{x_i^{n-1}} \text{ if } i = S^i. \end{cases} \tag{2}$$

Finally, let $\mathcal{M}$ be a finite subset of $\mathbb{N}^*$. Some x^n are selected by a sequence m^n as the pseudo-random bit sequence of our generator. The sequence $(m^n)_{n\in\mathbb{N}} \in \mathcal{M}^{\mathbb{N}}$ is computed with ISAAC. So, the generator returns the following values: the components of x^{m^0}, followed by the components of $x^{m^0+m^1}$, followed by the components of $x^{m^0+m^1+m^2}$, *etc.* In other words, the generator returns the following bits:

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

or the following integers:

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

The basic design procedure of the novel generator is summed up in Table 1. The internal state is x, the output array is r. a and b are those computed by ISAAC and XORshift generators. Lastly, c and N are constants and $\mathcal{M}$ = {c, c+1} ($c \geqslant 3\mathsf{N}$ is recommended).

```
**Input**: the internal state $x$ (an array of N bits)
**Output**: an array $r$ of N bits
$a \leftarrow ISAAC()$;
$m \leftarrow a \ mod \ 2 + c$;
for $i = 0, \ldots, m$ do
    $b \leftarrow XORshift()$;
    $S \leftarrow b \ mod$ N;
    $x_S \leftarrow \overline{x_S}$;
end
$r \leftarrow x$;
return $r$;
```

Algorithm 1: An arbitrary round of CI(ISAAC, XORshift)

m :	4					5						4				
S	2	4	2	2		5	1	1	5	5		3	2	3	3	
In thisx^0					x^4						x^9					x^{13}
1					1		$\xrightarrow{1} 0$	$\xrightarrow{1} 1$			1					1
0	$\xrightarrow{2} 1$		$\xrightarrow{2} 0$	$\xrightarrow{2} 1$	1						1		$\xrightarrow{2} 0$			0
1					1						1	$\xrightarrow{3} 0$		$\xrightarrow{3} 1$	$\xrightarrow{3} 0$	0
0		$\xrightarrow{4} 1$			1						1					1
0					0	$\xrightarrow{5} 1$			$\xrightarrow{5} 0$	$\xrightarrow{5} 1$	1					1

Binary Output: $x_1^0 x_2^0 x_3^0 x_4^0 x_5^0 x_1^4 x_2^4 x_3^4 x_4^4 x_5^4 x_1^9 x_2^9 x_3^9 x_4^9 x_5^9 x_1^{13} x_2^{13}$... = 10100111101111110... Integer Output: x^0, x^0, x^4, x^6, x^8... = 20, 30, 31, 19...

Table 1. Application example

3.2 Example

In this example, N = 5 and $\mathcal{M}$ = {4,5} are chosen for easy understanding. The initial state of the system x^0 can be seeded by the decimal part of the current time. For example, the current time in seconds since the Epoch is 1237632934.484084, so $t = 484084$. $x^0 = t \pmod{32}$ in binary digits, then $x^0 = (1, 0, 1, 0, 0)$. m and S can now be computed from ISAAC and XORshift:

- m = 4, 5, 4, 4, 4, 4, 5, 5, 5, 5, 4, 5, 4,...
- S = 2, 4, 2, 2, 5, 1, 1, 5, 5, 3, 2, 3, 3,...

Chaotic iterations are done with initial state x^0, vectorial logical negation f_0 and strategy S. The result is presented in Table 3. Let us recall that sequence m gives the states x^n to return: $x^4, x^{4+5}, x^{4+5+4}, \ldots$

So, in this example, the generated binary digits are: 10100111101111110011... Or the integers are: 20, 30, 31, 19...

3.3 Chaotic iterations and chaos

Generally the success of a PRNG depends, to a large extent, on the following criteria: uniformity, independence, storage efficiency, and reproducibility. A chaotic sequence may have these good pseudo-random criteria and also other chaotic properties, such as: ergodicity, entropy, and expansivity. A chaotic sequence is extremely sensitive to the initial states. 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 well fits the requirements of pseudo-random sequence. Contrary to ISAAC or XORshift, our generator possesses these chaotic properties.

However, despite a huge number of papers published in the field of chaos-based PRNGs, 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, require considerably more storage spaces, and lose their chaotic properties during computations. These major problems restrict their use as generators [10]. Moreover, even if the algorithm obtained by the inclusion of chaotic maps is itself chaotic, the implementation of this algorithm on a machine can cause it lose its chaotic nature. This is due to the finite nature of the machine numbers set.

In this paper we don't simply integrate chaotic maps hoping that the implemented algorithm remains chaotic. The PRNG algorithms we conceive are constituted by discrete chaotic iterations that we mathematically proved in [3], that produce topological chaos as defined by Devaney. In the same paper, we raised the question of their implementation, proving in doing so that it is possible to design a chaotic algorithm and a chaotic computer program. In conclusion, the generator proposed in this paper does not inherit its chaotic properties from a continuous real chaotic map, but from discrete chaotic iterations defined in Section 2.2. As quoted above, it has been proven in [3] that chaotic iterations behave as chaos, as it is 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. This allows the conception of a new generation of chaotic PRNGs. Because only integers are manipulated in discrete chaotic iterations, the chaotic behavior of the system is preserved during computations, and these computations are fast.

4 Security analysis

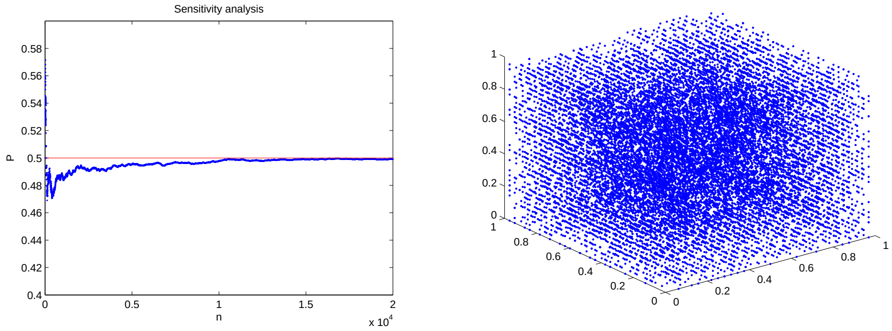
In this section a security analysis of the proposed generator is given.

4.1 Key space

The PRNG proposed in this document 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 PRNG combines digits of two other PRNGs: ISAAC and XORshift. Let k_1 and k_2 be the key spaces of ISAAC and XORshift. So the total key space size is close to $2^\mathsf{N} \cdot k_1 \cdot k_2$. Finally, the impact of $\mathcal{M}$ must be taken into account. This leads to conclude that the key space size is large enough to withstand attacks.

4.2 Key sensitivity

This PRNG is highly sensitive to the initial conditions. To illustrate this property proved in [3], 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 1a (x axis is sequence lengths, y axis is variance ratio P). Variance ratios approach 0.50, which indicates that the system is extremely sensitive to the initial conditions.



a. sensitivity

b. Second order distribution

Fig. 1. Security analysis

4.3 Uniform distribution

Figure 1b gives a 3D graphic representation of the distribution of a random sequence obtained by our generator. The point cloud presents a uniform distribution that tends to fill the complete 3D space, as expected for a random signal. To obtain this cloud, we have first changed the binary sequence to a N-bit integer sequence x_1, x_2, x_3, x_4... Then we have plot $\left(\frac{x_1}{2^N}, \frac{x_2}{2^N}, \frac{x_3}{2^N}\right), \left(\frac{x_2}{2^N}, \frac{x_3}{2^N}, \frac{x_4}{2^N}\right)$...

5 TestU01 Statistical Test Results

In a previous section, we have shown that the proposed PRNG has strong chaotic properties, as Devaney's chaos. In particular, this generator is better than the well-known XORshift and ISAAC, in the topological point of view. In addition to being chaotic, we will show in this section that CI(ISAAC,XORshift) is better than XORshift, and at least as good as ISAAC [18] in the statistical point of view. Indeed, similarly to ISAAC and contrary to XORshift, CI(ISAAC,XORshift) can pass the stringent Big Crush battery of tests included in TestU01. In addition, our generator achieves to pass all the batteries included in TestU01. To our best knowledge, this result has not been proven for ISAAC, and only one other generator is capable of doing this [6]

Table 2. TestU01 Statistical Test

Battery	Parameters	Statistics
Rabbit	32×10^9 bits	40
Alphabit	32×10^9 bits	17
Pseudo DieHARD	Standard	126
FIPS_140_2	Standard	16
Small Crush	Standard	15
Crush	Standard	144
Big Crush	Standard	160

5.1 TestU01

Indeed, the quality of a PRNG should be based on theoretical fundamentals but should also be tested empirically. Various statistical tests are available in the literature that test a given sequence for some level of computational indistinguishability. Major test suites for RNGs are TestU01 [11], the NIST suite [15], and the DieHARD suites [12]. The DieHARD suites, which implement many classical RNG tests, have some drawbacks and limitations. The National Institute of Standards and Technology (NIST), in the United States, has implemented a test suite (16 tests) for RNGs. It is geared mainly for the testing and certification of RNGs used in cryptographic applications. TestU01 is extremely diverse in implementing classical tests, cryptographic tests, new tests proposed in the literature, and original tests. In fact, it encompasses most of the other test suites. The proposed PRNG has been tested using TestU01 for its statistical pseudo randomness.

5.2 Batteries of tests

Table 2 lists seven batteries of tests in the TestU01 package. ”Standard” parameter in this Table refers to the built-in parameters of the battery. TestU01 suite implements 518 tests and reports $p-$values. If a $p-$value is within $[0.001, 0.999]$, the associated test is a success. A $p-$value lying outside this boundary means that its test has failed.

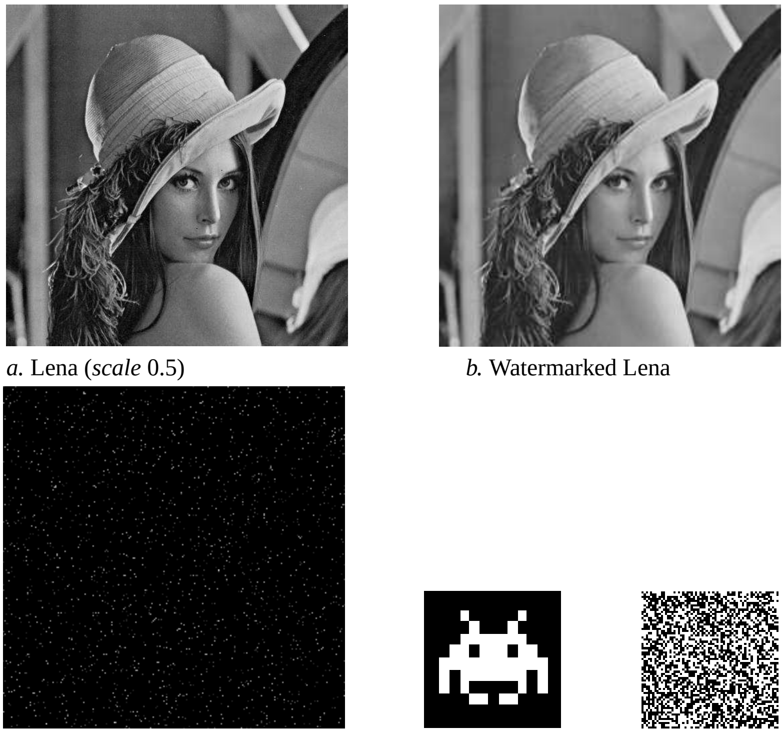
5.3 Analysis

In a sound theoretical basis, a PRNG based on discrete chaotic iterations (ICs) is a composite generator which combines the features of two PRNGs. The first generator constitutes the initial condition of the chaotic dynamical system. The second generator randomly chooses which outputs of the chaotic system must be returned. The intention of this combination is to cumulate the effects of chaotic and random behaviors, to improve the statistical and security properties relative to each generator taken alone.

This PRNG based on discrete chaotic iterations may utilize any reasonable RNG as inputs. For demonstration purposes, XORshift and ISAAC are adopted here. The PRNG with these inputs can pass all of the performed tests.

6 Application example in digital watermarking

In this section, an application example is given in the field of digital watermarking: a watermark is encrypted and embedded into a cover image using chaotic iterations and our PRNG. The carrier image is the famous Lena, which is a 256 grayscale image, and the watermark is the 64×64 pixels binary image depicted in Fig.2d. Let us encrypt the watermark by using chaotic iterations. The initial state x^0 of the system is constituted by the watermark, considered as a boolean vector. The iteration function is the vectorial logical negation f_0. The PRNG presented previously is used to obtain a sequence of integers lower than 4096, which will constitute the chaotic strategy $(S^k)_{k \in \mathbb{N}}$. Thus, the encrypted watermark is the last boolean vector generated by the chaotic iterations. An example of such an encryption, with 5000 iterations, is given in Fig.2e.


a. Lena (*scale* 0.5)

b. Watermarked Lena

c. Differences

d. Watermark

e. Encrypted watermark

Fig. 2. Original and watermarked Lena

Let L be the 256^3 booleans vector constituted by the three last bits of each pixel of Lena. We define U^k by $U^0 = S^0$ and $U^{n+1} = S^{n+1} + 2 \times U^n + n$ $[mod\ 256^3]$. The watermarked Lena I_w is obtained from the original Lena I_o, the three last bits of which are replaced by the result of 64^2 chaotic iterations with initial state L, and strategy U^k

(see Fig.2b). Spatial domain embedding has been chosen here for easy understanding, but this watermarking scheme can be adapted to frequency domain (for an example of its use in DWT domain, see [2]). The extraction of the watermark can be obtained in the same way [2]. Remark that the map $\theta \mapsto 2\theta$ of the torus, which is the well-known dyadic transformation (an example of topological chaos [7]), has been chosen to make $(U^k)_{k\leqslant 64^2}$ highly sensitive to the chaotic encryption strategy.

The robustness of this data hiding scheme through geometric and frequency attacks has been studied in [2]. The chaos-security and stego-security are proven in [8]. The difference with the scheme presented in these papers is the way to generate strategies, *i.e.*, the choice of the initial conditions for chaotic iterations, in the encryption and embedding stages. This improvement does not alter robustness and subspace-security. We have shown in this study that this replacement enhances the speed of the scheme. Moreover, it resolves a potential security lack related to the use of a logistic map [1] when generating the strategies: this lack might be exploited by an attacker in Watermark-Only-Attack and Known-Message-Attack setups [4]. Instead of logistic map, our PRNG has good statistical properties and can withstand such attacks. This claim will be deepened in future work.

7 Conclusions and future work

In this paper, the PRNG proposed in [17] is improved. This is achieved by using the famous ISAAC and XORshift generators and by combining these components with chaotic iterations. Thus we obtain a faster generator which satisfies chaotic properties. In addition to passing the NIST tests suite, this new generator successfully passes all the stringent TestU01 battery of tests. The randomness and disorder generated by this algorithm has been evaluated. It offers a sufficient level of security for a whole range of applications in computer science. An application example in the field of data hiding is finally given. In future work, the comparison of different chaotic strategies will be explored and other iteration functions will be studied. Finally, other applications in computer science security field will be proposed, especially in cryptographic domains.

References

1. D. Arroyo, G. Alvarez, and V. Fernandez. On the inadequacy of the logistic map for cryptographic applications. *X Reunin Espaola sobre Criptologa y Seguridad de la Informacin (X RECSI)*, 1:77–82, 2008.
2. J. Bahi and C. Guyeux. A new chaos-based watermarking algorithm. In *SECRYPT 2010, International conference on security and cryptography*, pages ***–***, Athens, Greece, 2010. To appear.
3. J. M. Bahi and C. Guyeux. Chaotic iterations and topological chaos. *arXiv*, 0810.3154, 2008.
4. F. Cayre and P. Bas. Kerckhoffs-based embedding security classes for woa data hiding. *IEEE Transactions on Information Forensics and Security*, 3(1):1–15, 2008.
5. 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*, 42:847–853, 2009.

6. S. Corsaro, P.L. De Angelis, Z. Marino, F. Perla, and P. Zanetti. On parallel asset-liability management in life insurance: a forward risk-neutral approach. *Parallel Computing*, In Press, 2009.
7. R. L. Devaney. *An Introduction to Chaotic Dynamical Systems*. Redwood City: Addison-Wesley, 2nd edition, 1989.
8. C. Guyeux, N. Friot, and J. M. Bahi. A more secure information hiding scheme than spread-spectrum obtained by chaos-security. *arXiv 0032565*, 2010.
9. R. J. Jenkins. Isaac. *Fast Software Encryption*, pages 41–49, 1996.
10. L. Kocarev. Chaos-based cryptography: a brief overview. *IEEE Circ Syst Mag*, 7:6–21, 2001.
11. P. L'ecuyer and R. Simard. Testu01: A software library in ansi c for empirical testing of random number generators. *Laboratoire de simulation et doptimisation. Universit de Montral IRO*, 2009.
12. G. Marsaglia. Diehard: a battery of tests of randomness. *http://stat.fsu.edu/ geo/diehard.html*, 1996.
13. G. Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 2003.
14. L. Po-Han, C. Yi, P. Soo-Chang, and C. Yih-Yuh. Evidence of the correlation between positive lyapunov exponents and good chaotic random number sequences. *Computer Physics Communications*, 160:187–203, 2004.
15. NIST Special Publication 800-22 rev. 1. A statistical test suite for random and pseudorandom number generators for cryptographic applications. August 2008.
16. F. Robert. *Discrete Iterations. A Metric Study*, volume 6. Springer Series in Computational Mathematics, 1986.
17. Q. Wang, C. Guyeux, and J. M. Bahi. A novel pseudo-random generator based on discrete chaotic iterations for cryptographic applications. In *First International Conference on Evolving Internet*, 2009.
18. B. A. Wichmanna and I. D. Hillb. Generating good pseudo-random numbers. *Computational Statistics & Data Analysis*, 51:1614–1622, 2006.
19. 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*, 15(3):64–68, 2008.