Arakhnê Project

Arakhnê Project

Arakhnê Document

Référence:	XXX
Version:	0.1
Modifié:	March 21, 2013
Statut:	Confidentiel

Auteurs:

P H D T H E S I S

to obtain the title of

PhD of Science

Defended by

Xiaole FANG

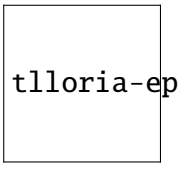
Utilization of chaotic dynamics for generating pseudorandom numbers in various contexts

---

Thesis Advisors : Jacques M. Bahi and Laurent Larger

Jury :

Reviewers :	Pierre Spitéri, Emeritus Professor	-	IRIT-ENSEEIHT
	Gilles Millérioux , Professor	-	University of Lorraine
Advisors :	Jacques M. Bahi, Professor	-	University of Franche-Comté
	Laurent Larger, Professor	-	University of Franche-Comté
President :		-	
Examinators :	Christophe Guyeux, Assistant Professor	-	University of Franche-Comté
	Philippe Guillot, Assistant Professor	-	University of Paris 8
		-	
Invited :		-	



Acknowledgments

Last thing to do :-)

Table des matières

I Preliminaries **1**

1 Significance of this thesis **3**

1.1 History of Chaotic RNG . . . 3

1.2 Motivation . . . 4

1.3 Objectives of this thesis . . . 5

1.3.1 The local research context . . . 5

1.3.2 The objectives . . . 6

1.4 Contributions . . . 6

1.4.1 Peer-reviewed International Journals . . . 7

1.4.2 Peer-reviewed International Conferences . . . 7

1.5 Thesis Organization . . . 8

1.6 Abbreviations . . . 9

1.7 Mathematical Symbols . . . 9

2 Research Background **11**

2.1 The Mathematical Theory of Chaos . . . 11

2.1.1 General Presentation . . . 11

2.1.2 On Devaney's Definition of Chaos . . . 11

2.1.3 Chaotic iterations . . . 12

2.1.4 Chaos Deflation when Computing . . . 13

2.1.5 Optical Chaos . . . 14

2.2 General Presentation of (Pseudo)Random Generation . . . 14

2.2.1 True Random Number Generators (TRNGs) . . . 15

2.2.2 Pseudo Random Number Generators (PRNGs) . . . 16

2.2.3 Chaos-based Random Number Generators . . . 16

2.3 Statistical Tests for Randomness 17
2.3.1 NIST statistical test suite 17
2.3.2 DieHARD battery of tests 18
2.3.3 ENT test program 20
2.3.4 Comparative test parameters 20
2.3.5 TestU01 Statistical Test 22
2.4 FPGA 26

II Pseudorandom Number Generator Based on Chaotic Iteration 29

3 Computer Science RNGs : an Introduction 31

4 Review of works 35
4.1 Some well-known pseudorandom generators 35
4.1.1 Blum Blum Shub 35
4.1.2 The logistic map 35
4.1.3 XORshift 36
4.1.4 ISAAC 37
4.2 CIPRNG, version 1 [**?**] 37
4.2.1 Chaotic iterations as PRNG 37
4.2.2 CIPRNG version 1 : the algorithm 38
4.3 The CIPRNG : Version 2 [**?**] 39
4.3.1 Defining the sequence m^n 39
4.3.2 Chaotic strategy 40
4.3.3 CIPRNG version 2 : the algorithm 40

5 Investigating the Design of new CIPRNGs 43
5.1 Investigating the CIPRNG Version 1 Algorithm 43
5.1.1 On the periodicity of version 1 43
5.1.2 Security Analysis 47
5.1.3 An Efficient and Cryptographically Secure Generator Based on CIPRNG Version 1 50

5.2 CIPRNG Version 2 : a Security Proof 50
5.2.1 Algorithm conversion 50
5.2.2 Proof 52
5.3 “LUT” CIPRNG(XORshift,XORshift) Version 3 54
5.3.1 Introduction 54
5.3.2 Sequence m 55
5.3.3 Defining the chaotic strategy $\mathcal{S}$ with a LUT 57
5.3.4 LUT CI(XORshift,XORshift) Algorithm 58
5.3.5 LUT CI(XORshift,XORshift) example of use 58
5.3.6 Security Analysis 59
5.4 The version 4 category of CIPRNGs 59
5.4.1 XOR CIPRNG 59
5.4.2 CIPRNG version 4 : the algorithm 60
5.4.3 Security Analysis 61
5.4.4 Efficient cryptographic secure PRNG based on CI 63

6 Randomness Quality of CIPRNGs 65
6.1 Test results for some PRNGs 65
6.2 Test results and comparative analysis for the CIPRNG version 3 68
6.3 Tests results and comparative analysis for the CIPRNG version 4 70
6.4 Assessment of four versions of CIPRNGs schemes using XORshift Generator 76
6.4.1 NIST evaluation 76
6.4.2 Diehard 76
6.4.3 Comparative test parameters 76

7 An optimization technique on pseudorandom generators based on chaotic iterations 83
7.1 Presentation of new well known generators 83
7.1.1 Introduction 83
7.1.2 Details of some Existing Generators 84
7.2 Results and discussion 85
7.2.1 Tests based on the Single CIPRNG 87

7.2.2 Tests based on the Mixed CIPRNG . . . 87
7.2.3 Tests based on the Multiple CIPRNG . . . 90
7.2.4 Results Summary . . . 90

8 Illustrative Example of use of CIPRNGs in the Computer Science Security Field 93

8.1 Introduction . . . 93
8.2 Application Evaluation . . . 97
8.2.1 The Proposed Information Hiding Method . . . 97
8.2.2 First Experimental Evaluation of the Proposed Scheme . . . 98
8.2.3 A small evaluation of Encryption . . . 101
8.3 Conclusion . . . 101

9 FPGA Acceleration of CIPRNGs 103

9.1 introduction . . . 103
9.2 CIPRNG design on FPGA . . . 104
9.2.1 Selection of the CIPRNG version . . . 104
9.2.2 Design of XORshift . . . 104
9.2.3 Design of BBS . . . 104
9.2.4 Design of the chaotic iterations . . . 106

10 Software and Hardware Implementation of a Watermarking Scheme using CI-PRNG 107

10.1 Introduction . . . 107
10.2 Definition of our Chaos-Based Information Hiding Scheme . . . 107
10.2.1 Most and least significant coefficients . . . 108
10.2.2 Stages of the algorithm . . . 109
10.3 Implementations : from software to hardware . . . 110
10.3.1 Software Implementation : Experimental Protocol . . . 110
10.3.2 Hardware Implementation . . . 113
10.4 Robustness evaluation . . . 113

III Random Number Generators Based On Optoelectronic Chaotic Laser 117

11 Introduction 119

12 Noise And Chaos Contributions in Fast Random Bit Sequences Generated From Broadband Optoelectronic Entropy Sources 121

12.1 Method for Random Bit Sequence Generation from an Optoelectronic Signal 121

12.1.1 Setup delivering a broadband optoelectronic signal 121

12.1.2 Extraction method for the random binary sequence 123

12.2 Effect of Noise on the Entropy Rate in the Binary Sequence 129

12.2.1 Introducing noise in the simulated chaotic dynamics 130

12.2.2 Entropy estimation for each bit cell 131

12.3 Statistical Tests 133

12.3.1 The tested streams 134

12.3.2 Statistical tests 134

12.4 Discussion, Conclusion, and Perspectives 137

13 CI Random Stream Generation Via Optoelectronic Chaotic Laser 143

13.1 Setup of a broadband optoelectronic chaotic laser signal 143

13.2 Random stream generation approach by applying CI method to chaotic laser 144

13.2.1 The proposed scheme 144

13.2.2 Examples of processing 145

13.3 NIST test results 146

IV Conclusion and Perspectives 149

14 Conclusion 151

14.1 CI method 151

14.2 Optical source 153

15 Perspectives 155

15.1 CI method 155

15.2 Optical source 155

I

Preliminaries

1

Significance of this thesis

Nowadays, chaos is sometimes considered as the third scientific revolution of XXth century, after the relativity theory and the quantum mechanics [?]. In 1963, American meteorologist Lorenz found random behavior in some well-defined systems, and he proposed the "butterfly" theory, which broken Laplace determinism [?]. In 1975, T.Y. Li and J. Yorke from Maryland University published a paper named "period 3 implies chaos" [?], in which chaos was first used to reveal the process that how the order turns into disorder. As a surprising branch in natural science, chaos theory was formulated during the '60s and established in '70s. It is a blanketing theory that covers all aspects of science, such as mathematics [?, ?, ?], physics [?, ?], biology [?, ?], chemistry [?, ?], and engineering [?, ?], etc. [?, ?]. Some researchers have pointed out that there exists tight relationship between chaos and randomness [?, ?]. So it is a natural idea to use chaos to enrich the design of new RNGs (Random Number Generators). In addition, since many chaotic systems have been extensively studied in past years, there are plenty of theoretical results that can be used to make performance analyses on the designed chaotic RNG.

1.1/ History of Chaotic RNG

Actually, the use of chaos in the realization of random numbers generators has been proposed and analyzed since many years [?, ?, ?, ?, ?, ?]. John von Neumann suggested the use of logistic map in 1947 [?], partly because it had a known algebraic distribution, and its iterated values could be easily transformed into any desired distribution. Then, Robert A. J. Matthews proposed to generate binary sequences based on a generalized logistic map [?] in 1989. From then on, digital chaotic RNGs attract more and more attention of many researchers from different areas. Many different chaotic systems have been used : Logistic map [?, ?], and its generalized version [?], 2-D Henon attractor [?, ?], Chebyshev map [?], piecewise linear chaotic maps [?, ?], and so on. Besides using chaotic maps, in particular, discrete-time chaotic circuit have also been served as random sources in many chaos-based random number generators, such as p-adic discrete-time chaotic systems [?], first-order non-uniformly sampling DPLL (Digital Phase-Locked Loop) circuits [?], etc.

Traditionally, the entropy source for a RNG is a physical random phenomena. Such sources of randomness include noise in resistors [**?**], phase noise of lasers [**?**], radioactive decay [**?**], are the favorite entropy sources to generate ideal random numbers. However, the rates for these true random numbers are lower than 20 Mbits/s. Fortunately, due to the development of modern technology, nonlinear dynamics in optics (laser dynamics, large delay optical, or optoelectronic cavities) with high complexity dynamics and speed are feasible [**?**]. This is why many random number generators based on semiconductor lasers operating in chaos have been recently proposed [**?**, **?**, **?**].

1.2/ Motivation

Most of chaotic RNGs originate from physics. Their state s is a real in the interval $[0, 1]$, their output bit is computed as a function of the state. In the simplest case, if $s > 0.5$, then the output is 1, else the output is 0. When they are realized in digital computers with finite computing precision, despite the desirable mathematical properties of the chaotic functions, their use in floating point based implementations of RNGs raise some problems. Indeed restricting a chaotic system to a finite universe always causes various issues such as short cycle length, non-ideal distribution and correlation functions. However, RNGs based on chaotic iterations was initiated by constructing a chaotic system on the integers domain instead of the real numbers domain. The topological chaotic properties of the generator are shown by a rigorous framework. The design goal of these generators was to take advantage of the random-like properties of real valued chaotic maps and, at the same time, secure optimal cryptographic properties. From the above discussion, I believe that the research on chaotic iterations will be helpful to benefit the conventional cryptology and open a broader road to the design of good RNGs.

Optical chaos was an exciting field of research in mid ‘80s, which has seen a recent resurgence, especially in the last 4 years, as chaotic waveforms for random number generation found a deep interest within the community of analogue broadband chaotic optical systems. In 2009, Reidler *et al.* published a paper entitled “An optical ultrafast random bit generator” [**?**], in which they presented a physical system for a random number generator based on a chaotic semiconductor laser. This generator is claimed to reach potentially the extremely high rate of 300 Gb/s. With my supervisors, we reported on analyses and experiments of their method, which has led to a discussion recalled in the third part of this thesis, about the actual origin of the randomness as well as to the actually reachable bit rate. We have shown that the actual binary sequence randomness corresponds more to complex mixing of noisy components performed by digital post-processing operations, than to chaotic properties of the signal. We have also addressed the issue of a proper way to use chaotic motion in RNGs, which is shown to necessarily involve MSBs instead of the LSBs. Without this criteria, physical noise alone is enough to retrieve a random sequence. However, in that case deterministic chaos cannot be used, e.g., for PRNG synchronization, as it would be of interest if one would like for instance to implement physically the one-time pad (the truly

RNG becoming the keystream to be used only once for a perfectly secure encryption). Different random bit sequences have been investigated in that context, both from experimental time series generated by a recently proposed chaotic optical phase generator as well as from numerical simulations.

1.3/ Objectives of this thesis

1.3.1/ The local research context

This thesis focuses on the use of chaotic dynamics from nonlinear optical devices or chaotic mathematical systems, for generating pseudorandom numbers. During this thesis, a link has been established between two research teams of the FEMTO-ST Institute (Université de Franche-Comté), namely the AND team (Distributed Numerical Algorithms) and the OPTO (Optoelectronics, Photonics, and Optical Telecommunications) one. During this thesis, these teams have worked in a complementary manner on the generation of pseudorandom numbers.

Firstly, the AND team has recently proposed a novel family of pseudorandom numbers generators (algorithms) based on mathematical chaos. A short summary of these researches is given thereafter. It has been firstly proven in [**?**, **?**] that chaotic iterations (CIs), a suitable tool for fast computing iterative algorithms, satisfies the topological chaos property, as it is defined by Devaney [**?**]. Indeed, this PRNG has been obtained by combining chaotic iterations and two generators based on the logistic map in [**?**]. The resulted PRNG shows better statistical properties than each individual component alone. Additionally, various chaos properties have been established. The advantage of having such chaotic dynamics for PRNGs lies, among other things, in their unpredictability character. These chaos properties, inherited from chaotic iterations, are not possessed by the two inputted generators. The team has shown that, in addition of being chaotic, this generator can pass the NIST battery of tests, widely considered as a comprehensive and stringent battery of tests for cryptographic applications [**?**]. Then, in the papers [**?**, **?**], the speed of the former PRNG has been improved, among other things by replacing the two logistic maps by two XORshifts in [**?**], and ISAAC with XORshift in [**?**]. Additionally, the first generator is able to pass DieHARD tests [**?**], whereas the second one can pass TestU01 [**?**]. In [**?**, **?**], which is an extension of [**?**], the team has improved the speed, security, and evaluation of the former generator and of its application in information hiding. Then, a comparative study between various generators has been carried out and statistical results have been improved.

In prior literature, the iteration function was only the vectorial Boolean negation. It is then judicious to investigate whether other functions may replace the vectorial negation in the above approach. This is why a method using graphs with strongly connected components has been proposed in [**?**] as a selection criterion for chaotic iteration functions. The approach developed along these lines solves this issue by providing a class of functions whose iterations are chaotic according to Devaney and such that resulting PRNG success statistical

tests. Then the vectorial Boolean negation has been used as a prototype, and explanations about how to modify the iteration function without deflating the good properties of the associated generator has been brought in [**?**]. Simulation results and basic security analysis have finally been presented to evaluate the randomness of this new family of generators.

Secondly, in the OPTO side, the physical source of entropy is originating from a strongly deterministic process too using chaotic lasers [**?**, **?**, **?**]. Experiments have recently demonstrated that such an optical chaos can mask 10 Gb/s of a given data signal during transmissions over an installed fiber optic link [**?**]. An important objective of the OPTO team is then to explore different physical approaches, as efficiently as possible (in terms of speed and quality of randomness), to extract pseudorandom number sequences from analog chaotic dynamics. The physical devices that generate such dynamics are available and accessible via existing routines developed by the team. Final targeted applications are the production of optical fiber communications broadband secured by chaos.

1.3.2/ The objectives

In view of the state of the art detailed above, the main objective of my thesis was to use chaotic dynamics for generating pseudorandom numbers in various contexts. In more details, the aims was to :

- Develop and implement various algorithms including Version 1 CI, Version 2 CI, Version 3 LUT CI, and Version 4 CI, and to compare their performances with software and hardware based systems to evaluate the randomness of this new family of pseudorandom generators. For illustration purposes, all the proposed PRNGs should be used in well-detailed cryptographic application.
- Evaluate the contributions of noise and chaos in fast random bit sequences generated from broadband optoelectronic entropy sources. Ways to keep a deterministic origin in the optical generation of pseudorandomness must be regarded too. Finally, an appropriate road to mix chaotic iterations with optical chaotic laser could be investigated, in order to design proper random streams that can be synchronized and used for cryptography.

1.4/ Contributions

The scientific contribution during this thesis has two aspects, related either to computer science or to optics, which are reported in the two parts that follow the introducing part of this manuscript.

On the one hand, the first part contains developments and evaluations of various CI PRNGs methods. These generators are based on discrete chaotic iterations, which satisfy the well-respected Devaney's definition of chaos. New versions of this original family of generations have been introduced, studied, and experimented. All the CIPRNG family has been completely rethought, in order to implement it in FPGA architectures, which has led to a

well-appreciated and large speed improvement. The contributions in this computer science oriented part end by application examples of use in cryptography.

On the other hand, in the second part, we have participated to a deep analysis of both the origin of chaos and the deterministic character of the chaos-based photonic RNGs proposed in [**?**, **?**]. Our analysis strongly suggest that their method essentially exploits the noisy compound of the photonic signal, which is always present in such signals, would it be with (chaos) or without (noise) deterministic motion. Using only the most significant bits (or equivalently the 1-bit ADC conversion) is, according to this contribution, the unique solution to make such an optic-based PRNG as deterministic, and thus reproducible.

This thesis has led to the submission and/or the publication of the following articles.

1.4.1/ Peer-reviewed International Journals

– Jacques Bahi, Xiaole Fang, Christophe Guyeux, and Qianxue Wang (alphabetic order). Evaluating Quality of Chaotic pseudorandom Generators. Application to Information Hiding. IJAS, International Journal On Advances in Security, 4(1-2) :118–130. 2011 [**?**].

– Jacques Bahi, Xiaole Fang, Christophe Guyeux, and Laurent Larger (alphabetic order). FPGA Design For Pseudorandom Number Generator Based On Chaotic Iteration Used In Information Hiding Application. Applied Mathematic & Information Science. Submitted in Jan. 2013 [**?**].

– Jacques Bahi, Xiaole Fang, Christophe Guyeux, and Qianxue Wang (alphabetic order). Assessment of the suitability of three chaotic iterations schemes based on XORshift Generator for application in the Internet security field. JNCA, Journal of Network and Computer Applications, accepted. To appear [**?**].

– Jacques Bahi, Xiaole Fang, Christophe Guyeux, and Qianxue Wang (alphabetic order). FPGA Acceleration of a Pseudorandom Number Generator based on Chaotic Iterations. Cryptography and Communications. submitted in Jan. 2013 [**?**].

– Xiaole Fang, Benjamin Wetzel, Jean-Marc Merolla, John M. Dudley, Laurent Larger, Christophe Guyeux, and Jacques M. Bahi. Noise and chaos contributions in fast random bit sequence generated from broadband optoelectronic entropy sources. In IEEE Transaction on Circuits and Systems. Submitted in Jan. 2013. [**?**].

1.4.2/ Peer-reviewed International Conferences

– Qianxue Wang, Jacques M. Bahi, Christophe Guyeux, and Xaole Fang. Randomness quality of CI chaotic generators. Application to internet security. In INTERNET'10. The 2nd Int. Conf. on Evolving Internet, pages 125–130, Valencia, Spain, September 2010. IEEE seccion ESPANIA. Best paper [**?**].

– Jacques M. Bahi, Xiaole Fang, Christophe Guyeux, and Qianxue Wang (alphabetic order). On the design of a family of CI pseudorandom number generators. In WICOM'11,

7th Int. IEEE Conf. on Wireless Communications, Networking and Mobile Computing, Wuhan, China, pages 1–4, September 2011 [**?**].

– Jacques Bahi, Xiaole Fang, and Christophe Guyeux (alphabetic order). An optimization technique on pseudorandom generators based on chaotic iterations. In INTERNET'2012, 4-th Int. Conf. on Evolving Internet, Venice, Italy, pages 31–36, June 2012 [**?**].
– Jacques Bahi, Xiaole Fang, and Christophe Guyeux (alphabetic order). State-of-the-art in Chaotic Iterations based pseudorandom numbers generators Application in Information Hiding. In IHTIAP'2012, 1-st Workshop on Information Hiding Techniques for Internet Anonymity and Privacy, Venice, Italy, pages 90–95, June 2012 [**?**].

1.5/ Thesis Organization

The remainder of this manuscript is divided in four parts, as detailed below.

This first part details the significance and background of this thesis. An overview of random numbers and theirs generators is provided. Some techniques to generate chaotic random number, based on physical sources or mathematical mapping functions, will be discussed too. The fundamental views and definitions of chaotic iterations are then summarized. After these materials, some existing test suites, such as NIST statistical test suite, DieHARD battery of tests, ENT test program, Comparative test parameters, and TestU01, which are commonly used to verify the randomness of a sequence, are also introduced. The FPGA platforms, on which the PRNG systems will be built on, are finally presented.

In Part II, the developments and applications of chaotic iterations (CI) based pseudorandom number generators are presented. Firstly, The basic definitions concerning CI and PRNGs are recalled in Chapter 3. Then, Chapter 4 shows some famous pseudorandom number generators and two versions of generators based on CI. Next, by understanding the random-like dynamics of chaotic iterations, some novel developments and versions of CI based PRNGs are described in Chapter 5. In the following chapter, the evaluation and comparison between the proposed CI generators, using the most famous statistical tests, are shown. In Chapter 7, the statistical analysis of the CI methods for random number generators is summarized, and their strength and weakness are also commented. The state-of-the-art of chaotic iterations-based PRNGs is given in Chapter 8. As FPGA (Field-Programmable Gate Array) chips provide sufficient logic and storage elements on which complex algorithms can be built, this technology is then used to improve the speed and performance of our CI generators in Chapter 9. Lastly, an application example and its evaluations regarding the CI technology are expressed in Chapter 10.

The third part of this manuscript focuses on optic aspects of pseudorandom number generation. It is firstly verified in Part III that to use the MSBs (Most Significant Bits, or equivalently 1-bit ADC conversion) is the key to keep a deterministic chaotic laser information in generation of random bits. Secondly, the applications of chaotic iterations to optoelectronic chaotic signals, as described thereafter, is investigated. In Chapter 11, a brief

introduction on random number generators based on photonic technique is given. Then, in Chapter 12, a recently published method [**?**, **?**] using chaotic waveform for pseudorandom generation is reprocessed and analyzed, leading to the conclusion that LSBs are definitely a major hurdle for regenerating the initial random sequence. Finally, Chapter 13 shows two examples in adapting chaotic iterations to simulated chaotic waveform, leading to the generation of random sequences.

This manuscript is then concluded in Part IV, which provides a summary of the researches that have been realized during this thesis, and discussions about possible future work in this area is finally provided.

1.6/ Abbreviations

Abbreviation	Definition
RNGs	Random Number Generators
TRNGs	True Random Number Generators
PRNG	Pseudo Random Number Generator
NIST	National Institute of Standards and Technology
VOD	Video on Demand
FPGA	Field Programmable Gate Array

1.7/ Mathematical Symbols

Symbol Meaning

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

2

Research Background

This chapter, serving as the background of this thesis, is devoted to basic notations and terminologies in the fields of chaos theory and random numbers.

2.1/ The Mathematical Theory of Chaos

2.1.1/ General Presentation

Chaos theory studies the behavior of dynamical systems that are perfectly predictable, yet appear to be wildly amorphous and without meaningful. Chaotic systems are highly sensitive to initial conditions, which is popularly referred to as the butterfly effect. In other words, small differences in initial conditions (such as those due to rounding errors in numerical computation) yield widely diverging outcomes, rendering long-term prediction impossible in general [**?**]. This happens even though these systems are deterministic, meaning that their future behavior is fully determined by their initial conditions, with no random elements involved [**?**]. In other words, the deterministic nature of these systems does not make them predictable [**?**, **?**]. This behavior is known as deterministic chaos, or simply chaos. It has been well-studied in mathematics and physics, leading among other things to the well-established definition of Devaney recalled thereafter.

2.1.2/ On Devaney's Definition of Chaos

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 [**?**], is widely accepted [**?**].

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

– Topological transitivity :

$$\forall U, V \text{ open sets of } \mathcal{X},\ \exists k > 0, f^k(U) \cap V \neq \varnothing. \tag{2}$$

Intuitively, a topologically transitive map has points that eventually move under iteration from one arbitrarily small neighborhood to any other. Consequently, the dynamical system cannot be decomposed into two disjoint open sets that are invariant under the map. Note that if a map possesses a dense orbit, then it is clearly topologically transitive.

– 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 $\mathcal{X}$:

$$\overline{P} = \mathcal{X}. \tag{3}$$

Density of periodic orbits means that every point in the space is approached arbitrarily closely by periodic orbits. Topologically mixing systems failing this condition may not display sensitivity to initial conditions presented below, and hence may not be chaotic.

– 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 $d\left(f^n(x), f^n(y)\right) \geqslant \varepsilon$.
Intuitively, a map possesses sensitive dependence on initial conditions if there exist points arbitrarily close to x that eventually separate from x by at least ε under iteration of f. Not all points near x need eventually separate from x under iteration, but there must be at least one such point in every neighborhood of x. If a map possesses sensitive dependence on initial conditions, then for all practical purposes, the dynamics of the map defy numerical computation. Small errors in computation that are introduced by round-off may become magnified upon iteration. The results of numerical computation of an orbit, no matter how accurate, may bear no resemblance whatsoever with the real orbit.

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.

2.1.3/ Chaotic iterations

Let us now introduce an example of a dynamical systems family that has the potentiality to become chaotic, depending on the choice of the iteration function. This family is the basis of the PRNGs we have studied and developed during this thesis [**?**, **?**].

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 strategy. Then, the so-called *chaotic iterations* are defined by [**?**] :

$$\left\{ \begin{array}{l} x^0 \in \mathbb{B}^{\mathsf{N}}, \\ \forall n \in \mathbb{N}^*, \forall i \in [\![1; \mathsf{N}]\!], x_i^n = \left\{ \begin{array}{ll} x_i^{n-1} & \text{if } S^n \neq i \\ f(x^{n-1})_{S^n} & \text{if } S^n = i. \end{array} \right. \end{array} \right. \tag{4}$$

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. [**?**].

Chaotic iterations generate a set of vectors ; they are defined by an initial state x^0, an iteration function f, and a chaotic strategy S [**?**, **?**]. These "chaotic iterations" can behave chaotically as defined by Devaney, depending on the choice of f [**?**]. Furthermore, they are able to circumvent the problem of quantization errors when implementing chaotic systems on computers, as explained in the next section.

2.1.4/ Chaos Deflation when Computing

In the past two decades, the use of chaotic systems in the computer science security field has become more and more frequent, for instance when designing cryptosystems, PRNGs, or hash functions. However, when chaotic systems are realized in digital computers with finite computing precision, 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 infinite sets, these properties may become useless, deceptive, or ambiguous when the phase space is highly quantized (i.e., latticed) with a finite computing precision. In other words, the related problem is the dynamical degradation of continuous chaotic systems when they are implemented in finite computing precision.

When chaotic systems are realized in digital circuits or computers, the dynamical systems will be discretized both spatially and temporally. That is to say, they will become discrete-time and discrete-value chaotic systems [**?**] defined in discrete time and spatial lattice with finite elements. 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 [**?**, **?**, **?**, **?**, **?**].

The quantization errors, which are introduced at each iteration of digital chaotic systems on finite state machines, make that "pseudo orbits" depart from real ones in a very complex and uncontrolled manner. Due to the sensitivity of chaotic systems on initial conditions, even "trivial" changes of computer arithmetic can definitely change the pseudo orbits' structures. Although all quantization errors are quite deterministic when the finite precision and

the arithmetic are fixed, it is technically impossible to deal with these errors. Some random perturbation models have been proposed to depict quantization errors in digital chaotic systems, but they cannot exactly predict the actual dynamics of studied digital chaotic systems and have been criticized because of their essentially deficiencies

To sum up, continuous chaos may collapse into the digital world, and the deflation of chaotic properties is hard to measure or control when implementing chaotic continuous systems on computers. In that context, previous theoretical and practical studies directed by the AND team have concluded to the possibility to cope with this problem using chaotic iterations [**?**, **?**]. This thesis is an additional building block in that direction.

After having introduced the mathematical approach for describing chaotic dynamics together with an important example, we now roughly present the optical understanding of this multifaceted notion.

2.1.5/ Optical Chaos

Optical chaos, in this manuscript, will refer to chaos generated by laser instabilities using different schemes provided by semiconductors and fiber lasers. This optical chaos is observed in many non-linear optical systems. It as been successfully applied in a cryptographic context, for securing exchanges through optic fiber using optical chaos [**?**]. At this point, we can remark that :

– these systems are, of course, intrinsically dedicated to modern communication using optical fibers ;
– the dynamical processes involved in optical systems can be very fast, thus resulting in another interesting feature of chaos-based optical cryptosystems, their potentially very high encryption speed ;
– high complexity chaotic dynamics are obtained, whether due to intrinsic complex nonlinear coupling between light and matter interactions in lasers, or due to the presence of a large delay feedback cavity enabling dynamics with large number of degrees of freedom.

These chaotic dynamical systems will be used for pseudorandom number generation, this is why we introduce the context of such generations in the next section.

2.2/ General Presentation of (Pseudo)Random Generation

A random number generator (RNG) is a computational or physical device designed to generate a sequence of numbers or symbols that lack any pattern. As its name indicates, it appears random.

It is always a difficult task to generate good random number/sequence. Although it is accepted that rolling a dice is random, such a mechanical method is not usable in practice, at least for the applications we intend to consider. In past decades, random numbers were usually generated offline, based on some dedicated setup or devices, and the sequences were sto-

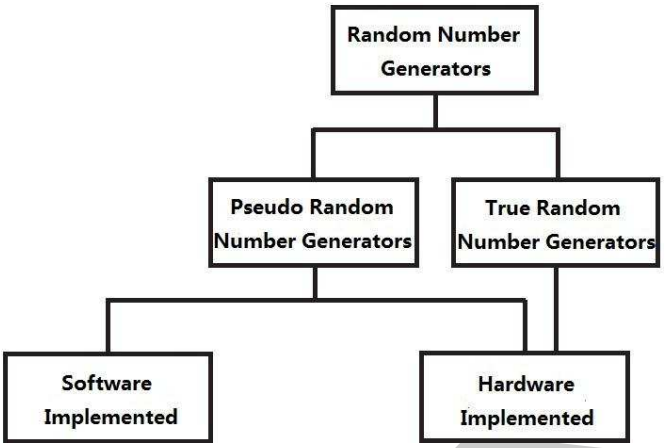


FIGURE 2.1 – Classification of random number generators

red in tables ready for use. These random tables are still available in the world-wide-web or some data CDROMs. However, due to online requirements and security issues, random tables have recently become inappropriate, and hence different ways to design new RNGs have been proposed.

These RNGs are in general grouped into two classes depending on their sources of randomness, namely true random number generators (TRNGs) and pseudorandom number generators (PRNGs), as shown in Fig. 2.1.

2.2.1/ TRUE RANDOM NUMBER GENERATORS (TRNGS)

A TRNG is a physical device that generates statistically independent and unbiased bits. They are also called non-deterministic RNGs. Such devices, which exist too in computers, are often based on microscopic phenomena that generate a low-level, statistically random "noise" signal, such as thermal noise, photoelectric effect, or any other quantum phenomena. These processes are, in theory, completely unpredictable, and the theory's assertions of unpredictability are subject to experimental test.

A quantum-based hardware random number generator typically consists in a transducer that converts some aspect of a given physical phenomena, into an electrical signal. An amplifier and other electronic circuitry bring the output of the transducer into the macroscopic realm, and some analog to digital converter translates the output into a digital number, often a simple binary digit 0 or 1. By repeatedly sampling the randomly varying signal, a series of random numbers is thus obtained.

2.2.2/ Pseudo Random Number Generators (PRNGs)

A pseudorandom number generator (PRNG), also known as deterministic random bit generator (DRBG), is an algorithm for generating a sequence of numbers that approximates the properties of random numbers [**?**]. The sequence is not truly random in that it is completely determined by a relatively small set of initial values, called the PRNG's state. Although sequences that are closer to truly random can be generated using hardware random number generators, pseudorandom numbers are important in practice for simulations (e.g., of physical systems with the Monte Carlo method), and are central in cryptography and procedural generation due to their fundamental property of reproductibility.

Common classes of these algorithms are linear congruential generators, Lagged Fibonacci generators, linear feedback shift registers, feedback with carry shift registers, and generalized feedback shift registers. Recent instances of pseudorandom algorithms include Blum Blum Shub, Fortuna, and the Mersenne twister. All these generators will be developed later in this manuscript.

2.2.3/ Chaos-based Random Number Generators

Since the seventies in last century, the use of chaotic dynamics for the generation of random sequences has raised a lot of interests. It is clearly pointed out by some researchers that there exists a close relationship between chaos and randomness, and many works have been witnessed in the last two decades [**?**]. To do so, chaotic dynamics are usually studied in two different domains, depending on whether the generator is an hardware or a software one.

– Continuous time domain : chaotic dynamics generated by differential equations.
– Discrete time domain : recurrent sequences using chaotic maps.

Chaos possesses several distinct properties that can possibly give meaning to the desire to use such dynamics for generating pseudorandomness, namely : the sensitivity to initial conditions, the ergodicity, the wide band spectrum, and the unpredictable and random-like behavior of iterates. Although it is still controversy to equate these properties with randomness and claim a chaos-based random number generator to be good enough, a lot of designs and applications, in particular related to communications securing, have been proposed.

Nowadays, it is common to use a chaotic map for pseudorandom number generation. Due to the recent design of electronic circuits for the realization of chaotic systems, it is also possible to generate the bit sequence by observing such dynamics, as a replacement of those physical random sources.

2.3/ Statistical Tests for Randomness

A theoretical proof for the randomness of a generator is impossible to give, as such proof requires first to mathematically define what is randomness. Therefore statistical inference based on observed sample sequences produced by the generator seems to be the best option. Considering the properties of binary random sequences, various statistical tests can be designed to evaluate the assertion that the sequence is generated by a perfectly random source. We have performed certain statistical tests for various CI PRNGs we proposed. These tests include TestU01 [**?**], NIST suite [**?**], DieHARD battery of tests [**?**], and Comparative test parameters. For completeness and for reference, we give in the following subsection a brief description of each of the aforementioned tests.

2.3.1/ NIST statistical test suite

Among the numerous standard tests for pseudo-randomness, a convincing way to show the randomness of the produced sequences is to confront them to the NIST (National Institute of Standards and Technology) Statistical Test, because it is an up-to-date test suite proposed by the Information Technology Laboratory (ITL). A new version of the Statistical Test Suite (Version 2.0) has been released in August 11, 2010.

The NIST test suite SP 800-22 is a statistical package consisting of 15 tests. They were developed to test the randomness of binary sequences produced by hardware or software based cryptographic PRNGs. These tests focus on a variety of different types of non-randomness that could exist in a sequence.

For each statistical test, a set of $p - values$ (corresponding to the set of sequences) is produced. The interpretation of empirical results can be conducted in any number of ways. In this manuscript, the examination of the distribution of $p - values$ to check for uniformity ($p - value_T$) is used : the distribution of $P - values$ is examined to ensure uniformity. Concretely, this demand is verified in an usual way as follows : if $p - value_T \geqslant 0.0001$, then the sequences can be considered to be uniformly distributed.

In our experiments, 100 sequences (s = 100), each with 1,000,000-bit long, are generated and tested. If the $p - value_T$ of any test is smaller than 0.0001, the sequences are considered to be not good enough and the generating algorithm is not suitable for usage.

In what follows, the fifteen tests of the NIST Statistical tests suite, are recalled. A more detailed description for those tests could be found in [**?**].

– **Frequency (Monobit) Test (FT)** 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 (FBT)** is to determine whether the frequency of ones in an M-bit block is approximately $M/2$, as would be expected under an assumption of randomness.

- **Runs Test (RT)** 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 (LROBT)** 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 (BMRT)** is to check for linear dependence among fixed length substrings of the original sequence.
- **Discrete Fourier Transform (Spectral) Test (DFTT)** 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 (NOTMT)** is to detect generators that produce too many occurrences of a given non-periodic (aperiodic) pattern.
- **Overlapping Template Matching Test (OTMT)** is to check the number of occurrences of pre-specified target strings.
- **Maurer's "Universal Statistical" Test (MUST)** is to detect whether or not the sequence can be significantly compressed without loss of information.
- **Linear Complexity Test (LCT)** is to determine whether or not the sequence is complex enough to be considered random.
- **Serial Test (ST)** is to determine whether the number of occurrences of the 2^m m-bit overlapping patterns is approximately the same as would be expected for a random sequence.
- **Approximate Entropy Test (AET)** 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.
- **Cumulative Sums (Cusum) Test (CST)** 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 (RET)** 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 (REVT)** is to detect deviations from the expected number of visits to various states in the random walk.

2.3.2/ DieHARD battery of tests

The DieHARD battery of tests was developed in 1996 by Prof. Georges Marsaglia from the Florida State University for testing randomness of sequences of numbers [**?**]. It has been the most sophisticated standard for over a decade. Because of the stringent requirements in the DieHARD test suite, a generator passing DieHARD battery of tests can be considered good as a rule of thumb. It was supposed to give a better way of analysis in comparison to original FIPS statistical tests.

The DieHARD battery of tests consists of 18 different independent statistical tests. Each test requires binary file of about 10-12 million bytes in order to run the full set of tests.

As the NIST test suite, most of the tests in DieHARD return a $p - value$, which should be uniform on $[0, 1)$ if the input file contains truly independent random bits. Those $p - values$ are obtained by $p = F(X)$, where F is the assumed distribution of the sample random variable X (often normal). But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus occasional $p-values$ near 0 or 1, such as 0.0012 or 0.9983 can occur. Unlike the NIST test suite, the test is considered to be successful when the $p - value$ is in range $[0 + \alpha, 1 - \alpha]$, where $\sqrt{\alpha}$ is the level of significance of the test.

For example, with a level of significance of 5%, $p - values$ are expected to be in $[0.025, 0.975]$. Note that if the $p - value$ is not in this range, it means that the null hypothesis for randomness is rejected even if the sequence is truly random. These tests are :

- **Birthday Spacings.** Choose random points on a large interval. The spacings between the points should be asymptotically Poisson distributed. The name is based on the birthday paradox.
- **Overlapping Permutations.** Analyze sequences of five consecutive random numbers. The 120 possible orderings should occur with statistically equal probability
- **Ranks of matrices.** Select some number of bits from some number of random numbers to form a matrix over 0,1, then determine the rank of the matrix. Count the ranks.
- **Monkey Tests.** Treat sequences of some number of bits as "words". Count the overlapping words in a stream. The number of "words" that don't appear should follow a known distribution. The name is based on the infinite monkey theorem.
- **Count the 1's.** Count the 1 bits in each of either successive or chosen bytes. Convert the counts to "letters", and count the occurrences of five-letter "words"
- **Parking Lot Test.** Randomly place unit circles in a 100×100 square. If the circle overlaps an existing one, try again. After 12,000 tries, the number of successfully "parked" circles should follow a certain normal distribution.
- **Minimum Distance Test.** Randomly place 8,000 points in a $10,000 \times 10,000$ square, then find the minimum distance between the pairs. The square of this distance should be exponentially distributed with a certain mean.
- **Random Spheres Test.** Randomly choose 4,000 points in a cube of edge 1,000. Center a sphere on each point, whose radius is the minimum distance to another point. The smallest sphere's volume should be exponentially distributed with a certain mean.
- **The Sqeeze Test.** Multiply 231 by random floats on [0,1) until you reach 1. Repeat this 100,000 times. The number of floats needed to reach 1 should follow a certain distribution.
- **Overlapping Sums Test.** Generate a long sequence of random floats on [0,1). Add sequences of 100 consecutive floats. The sums should be normally distributed with characteristic mean and sigma.
- **Runs Test.** Generate a long sequence of random floats on [0,1). Count ascending and descending runs. The counts should follow a certain distribution.
- **The Craps Test.** Play 200,000 games of craps, counting the wins and the number of throws per game. Each count should follow a certain distribution.

2.3.3/ ENT test program

ENT test program applies various tests to sequences of bytes stored in files and reports the results of those tests. The program is useful for evaluating random number generators for encryption and statistical sampling applications, compression algorithms, and other applications where the information density of a file is of interest [**?**].

There are 5 tests contained in the program :

1. Entropy test : Entropy testing, in bits per character (or byte), which corresponds to the incompressibility of the sequence (as a perfectly random sequence cannot be compressed, since no part of it can be expressed in terms of other parts). Hence entropy of 8 bits/byte means perfect randomness in the sense of incompressibility.
2. χ^2 test : χ^2 testing is very common for goodness-of-fit of sample distributions of random numbers. It is known to be very sensitive to deficiencies in random number generators (when it is located between 5% to 95%, data are treated as random).
3. Sample test : Sample test means can be tested for bias in random number generation. In binary mode, the expected mean is 0.5 while for bytes, the expected mean is 127.5.
4. Monte Carlo test : a Monte Carlo approximation of π, which is simply the evaluation the area of the unit circle using the N generated random numbers (X_i,X_{i-1}), $i = 2, ..., N$.
5. Serial Correlation test : Serial correlation coefficient evaluated from $< X_i, X_{i-1} > / < X_i, X_i >$, for $i = 2, ..N$. The intended value for perfect random sequences is 0.

2.3.4/ Comparative test parameters

In this section, five well-known statistical tests [**?**] are presented to play the role of easily verifiable 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.

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 $n \geqslant 10^7$.

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 $n \geqslant 21$.

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).

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.

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 is :

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

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.

2.3.5/ TestU01 Statistical Test

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. There are seven batteries of tests in the TestU01 package, which are listed in what follows :

- **SmallCrush.** The first battery to check, with 15 $p-$values reported. This is a fast collection of tests used to be sure that the basic requirements of randomness are satisfied. In case of success, this battery should be followed by Crush and BigCrush.
- **Crush.** This battery includes many difficult tests, like those described in [**?**]. It uses approximately 2^{35} random numbers and applies 96 statistical tests (it computes a total of 144 test statistics and $p - values$).
- **BigCrush.** The BigCrush uses approximately 2^{38} random numbers and applies 106 tests (it computes 160 test statistics and $p - values$). A suite of very stringent statistical tests, and the most difficult battery to pass.
- **Rabbit.** This battery of tests reports 38 $p - values$.
- **Alphabit.** Alphabit and AlphabitFile have been designed primarily to test hardware random bits generators. 17 $p - values$ are reported here.
- **Pseudo-DieHARD.** This battery implements most of the tests contained in the popular battery DieHARD or, in some cases, close approximations to them. It is not a very stringent battery. Indeed, there is no generator that can pass Crush and BigCrush batteries and fail Pseudo-DieHARD, while the converse occurs for several defective generators. 126 $p - values$ are reported here.
- **FIPS_140_2.** As recalled previously, the NIST (National Institute of Standards and Technology) of the U.S. federal government has proposed a statistical test suite. It is used to evaluate the randomness of bitstreams produced by cryptographic random number generators. This battery reports 16 $p - values$.

Six predefined batteries of tests are available in TestU01 ; three of them are for sequences of $\mathcal{U}(0, 1)$ random numbers and the three others are for bit sequences. In the first category, we have SmallCrush, Crush, and BigCrush.

To test a RNG for general use, one could first apply the small and fast battery SmallCrush. If it passes, one could then apply the more stringent battery Crush, and finally the yet more time-consuming battery BigCrush. These batteries of tests include the classical tests described in Knuth [**?**], for example : the run, poker, coupon collector, gap, max-of-t, and permutation tests. There are collision and birthday spacings tests in 2, 3, 4, 7, 8 dimensions, several close pairs tests in 2, 3, 5, 7, 9 dimensions, and correlation tests. Some tests use the generated numbers as a sequence of “random” bits : random walk tests, linear complexity tests, a Lempel-Ziv compression test, several Hamming weights tests, matrix rank tests, run and correlation tests, among others.

The batteries Rabbit, Alphabit, and BlockAlphabit are for binary sequences (e.g., a cryptographic pseudorandom generator or a source of random bits produced by a physical device).

They were originally designed to test a finite sequence contained in a binary file. When invoking the battery, one must specify the number n_B of bits available for each test. When the bits are in a file, n_B must not exceed the number of bits in the file, and each test will reuse the same sequence of bits starting from the beginning of the file (so the tests are not independent). When the bits are produced by a generator, each test uses a different stream. In both cases, the parameters of each test are chosen automatically as a function of n_B . The batteries Alphabit and Rabbit can be applied on a binary file considered as a source of random bits. They can also be applied on a programmed generator. Alphabit has been defined primarily to test hardware random bits generators. The battery PseudoDieHARD applies most of the tests in the well-known DieHARD suite of Marsaglia [106]. The battery FIPS_140_2 implements the small suite of tests of the FIPS_140_2 standard from NIST. The batteries described in this module will write the results of each test (on standard output) with a standard level of details (assuming that the Boolean switches of module swrite have their default values), followed by a summary report of the suspect p-values obtained from the specific tests included in the batteries. It is also possible to get only the summary report in the output, with no detailed output from the tests, by setting the Boolean switch swrite_Basic to FALSE. Rabbit and Alphabit apply 38 and 17 different statistical tests, respectively.

Some of the tests compute more than one statistic (and $p - value$) using the same stream of random numbers and these statistics are thus not independent. That is why the number of statistics in the summary reports is larger than the number of tests in the description of the batteries. For a more detailed description, the reader is referred to the documentation of the TestU01 library.

– **Small Crush :**
  smarsa_BirthdaySpacings
  sknuth_Collision
  sknuth_Gap
  sknuth_SimpPoker
  sknuth_CouponCollector
  sknuth_MaxOft
  svaria_WeightDistrib
  smarsa_MatrixRank
  sstring_HammingIndep
  swalk_RandomWalk1

– **Crush :**
  smarsa_SerialOver
  smarsa_CollisionOver
  smarsa_BirthdaySpacings
  snpair_ClosePairs
  snpair_ClosePairsBitMatch
  sknuth_SimpPoker

sknuth_CouponCollector
sknuth_Gap
sknuth_Run
sknuth_Permutation
sknuth_CollisionPermut
sknuth_MaxOft
svaria_SampleProd
svaria_SampleMean
svaria_SampleCorr
svaria_AppearanceSpacings
svaria_WeightDistrib
svaria_SumCollector
smarsa_MatrixRank
smarsa_Savir2
smarsa_GCD
swalk_RandomWalk1
scomp_LinearComp
scomp_LempelZiv
sspectral_Fourier3
sstring_LongestHeadRun
sstring_PeriodsInStrings
sstring_HammingWeight2
sstring_HammingCorr
sstring_HammingIndep
sstring_Run
sstring_AutoCor

– **Big Crush :**
smarsa_SerialOver
smarsa_CollisionOver
smarsa_BirthdaySpacings
snpair_ClosePairs
sknuth_SimpPoker
sknuth_CouponCollector
sknuth_Gap
sknuth_Run
sknuth_Permutation
sknuth_CollisionPermut
sknuth_MaxOft
svaria_SampleProd
svaria_SampleMean
svaria_SampleCorr

svaria_AppearanceSpacings
svaria_WeightDistrib
svaria_SumCollector
smarsa_MatrixRank
smarsa_Savir2
smarsa_GCD
swalk_RandomWalk1
scomp_LinearComp
scomp_LempelZiv
sspectral_Fourier3
sstring_LongestHeadRun
sstring_PeriodsInStrings
sstring_HammingWeight2
sstring_HammingCorr
sstring_HammingIndep
sstring_Run
sstring_AutoCor

– **Rabbit :**
smultin_MultinomialBitsOver
snpair_ClosePairsBitMatch
svaria_AppearanceSpacings
scomp_LinearComp
scomp_LempelZiv
sspectral_Fourier1
sspectral_Fourier3
sstring_LongestHeadRun
sstring_PeriodsInStrings
sstring_HammingWeight
sstring_HammingCorr
sstring_HammingIndep
sstring_AutoCor
sstring_Run
smarsa_MatrixRank
swalk_RandomWalk1

– **Alphabit :**
smultin_MultinomialBitsOver
sstring_HammingIndep
sstring_HammingCorr
swalk_RandomWalk1

– **Pseudo-DieHARD :**
Birthday Spacings test
Overlapping 5-Permutation test
Binary Rank Tests for Matrices test
Bitstream test
OPSO test
OQSO test
DNA test
Count-the-1's test
Parking Lot test
Minimum Distance test
3-D Spheres test
Squeeze test
Overlapping Sums test
Runs test
Craps test

– **FIPS_140_2 :**
Monobit test
"poker" test
Runs test
Longest Run of Ones in a Block test

TestU01 suite implements hundreds of 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.

2.4/ FPGA

We finally introduce the field-programmable gate array (FPGA) architecture, on which our generators will be implemented.

A FPGA is an integrated circuit designed to be configured by a customer or a designer after manufacturing-hence "field-programmable" (a FPGA is indeed a VLSI chip with some special features). The most common FPGA architecture consists of an array of logic blocks (called Configurable Logic Block, CLB, or Logic Array Block, LAB, depending on vendor), I/O pads, and routing channels. Generally, all the routing channels have the same width (number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array. There are some primitives such as lookup tables (LUT) and flip-flop (FF) inside the CLB. The functions of these primitives and connections between them can be configured for different designs. Programmable routing matrices (PRM), implemented in static RAMs, are used to connect the I/O ports of the CLBs.

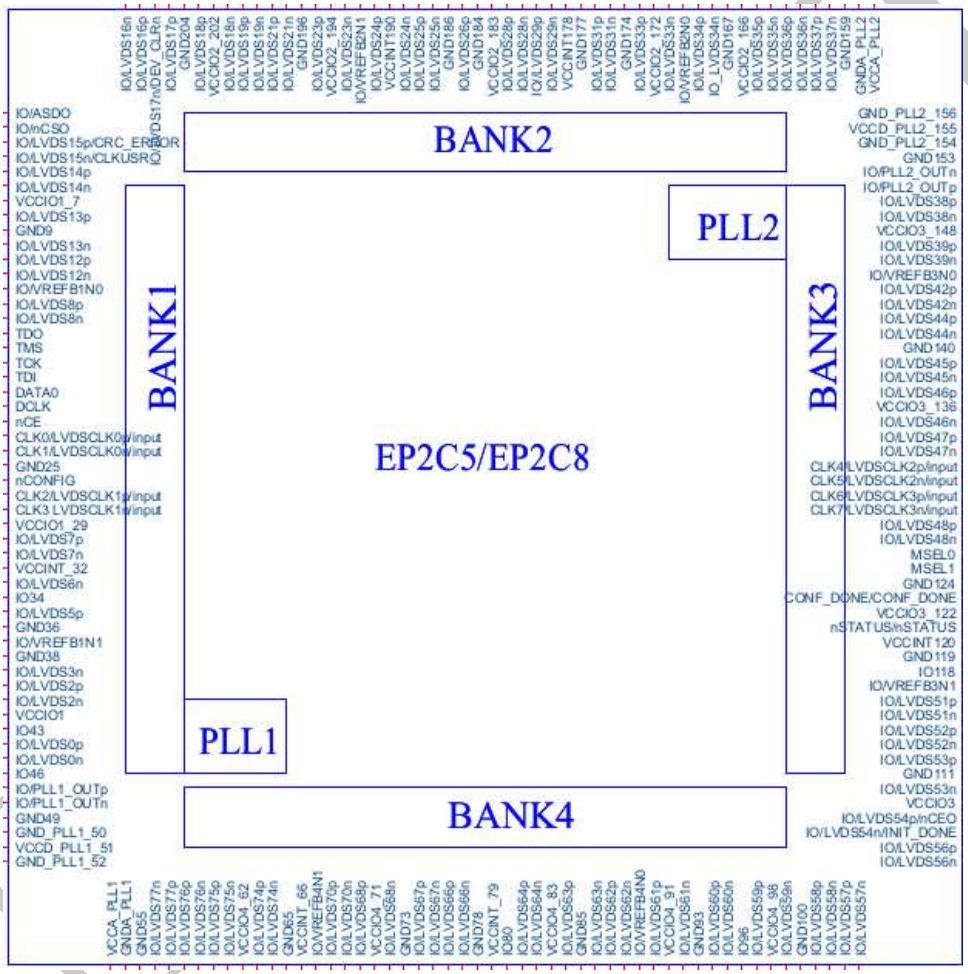


FIGURE 2.2 – FPGA EP2C8 core from ALTERA company

The advantages of FPGA designs over traditional VLSI designs are :
– The feasibility of reusing fast design to product time and chips for different designs.
– Easy simulation and debugging. Software simulator and debugger provide efficient methods of finding bugs and estimating of performance.
– Low cost prototyping for early designs.
– Design can be upgraded after deployment without hardware replacement.
– Developing hardware systems using design tools for FPGA is as easy as developing a software system.
– FPGA can be reprogrammed on the field.

To define the behavior of the FPGA, the user provides a hardware description language (HDL) or a schematic design. The most common HDLs are VHDL and Verilog, although in an attempt to reduce the complexity of designing in HDLs, which have been compared to the equivalent of assembly languages, there are moves to raise the abstraction level through the introduction of alternative languages.

A general structure outline of FPGA core (EP2C8) used in this thesis is shown in Fig.2.2

II

Pseudorandom Number Generator Based on Chaotic Iteration

3

Computer Science RNGs : an Introduction

Alone with the rapid development of Internet and universal application of multimedia technology, multimedia data including audio, image, and video has been transmitted over insecure channels, it implies the need to protect data and privacy in digital world. This development has revealed new major security issues. For example, new security concerns have recently appeared because of the evolution of the Internet to support such activities as e-Voting, VoD, and digital rights management [**?**]. The random number generators are very important cryptographic primitive widely used in the Internet security, because they are fundamental in cryptosystems and information hiding schemes.

RNGs are widely used in science and technology, it is a critical component in modern cryptographic systems, communication systems, statistical simulation systems, and any scientific area incorporating Monte Carlo methods and many others [**?**,**?**,**?**]. The random statistical quality of the generated bit sequence is measured by two aspects : the unpredictability of the bit stream and the speed at which the random bits can be produced. Other factors like system complexity, cost, reliability and so on, are also important for establishing successful RNGs. As stated before, there are usually two methods for random numbers generation. The first one, on which this part focuses, relates to deterministic algorithms implemented in hardware and software. In that context, pseudorandom numbers are degenerated from a single value called a “seed”, and using an algorithm called pseudorandom number generator (PRNG [**?**]). The second approach, debated in the next part, counts on high entropy signals, either from purely non deterministic and stochastic physical phenomena, or from deterministic but chaotic dynamical systems (necessarily mixed with an unavoidable noisy and smaller compound [**?**, **?**]). A potential advantage of the latter physical high entropy signal, resides in its deterministic features which might be used to achieve chaos synchronization as it has been already demonstrated [**?**] and widely used for secure chaos communications [**?**]. However, synchronization possibility of the random binary sequence extracted from the chaotic physical signal is still an open problem, which resolution could lead to the efficient and practical use of the Vernam cypher.

For the PRNGs algorithms, they are majorly 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. This is often either a real number in the interval (0, 1) or an integer in some finite range [**?**]. Such PRNGs can be easily implemented in any computational platform, however they suffer from the vulnerability that the future sequence can be deterministically computed if the seed or internal state of the algorithm is discovered. Additionally to their reproducible character, the main advantages of PRNGs is that no hardware cost is added and the speed is only counted on processing hardware. In a cryptographic context, these algorithms are developed to prevent guessing of the initial conditions, thus their speed are slowed down due to the increased complexity of such precaution.

Recently, some researchers have demonstrated the possibility to use chaotic dynamical systems as RNGs to reinforce the security of cryptographic algorithm, due to the unpredictability and distort-like property of chaotic dynamical systems [**?**, **?**, **?**]. These attempts are related to the hypothesis that digital chaotic systems can possibly reinforce the security of cryptographic algorithms, because the behaviors of such systems are very similar to those of physical noise sources [**?**]. For instance, in [**?**], chaos has been applied to strengthen some optical communications. More generally, the random-like and unpredictable dynamics of chaotic systems, their inherent determinism and simplicity of realization suggest their potential for exploitation as RNGs.

In chaotic cryptography, there are two main design paradigms : in the first paradigm chaotic cryptosystems are realized in analog circuits (mainly based on chaos synchronization [**?**]) whereas in the second paradigm chaotic cryptosystems are realized in digital circuits or computers (synchronization is not an issue). Generally speaking, synchronization based chaotic cryptosystems are generally designed for securing communications though noisy channels, and they cannot be directly extended to design digital ciphers in pure cryptography. However, some cryptanalytic works have shown that most synchronization based chaotic cryptosystems are not really secure, since it is possible to extract some information on the secret chaotic parameters [**?**]. Therefore, although chaos synchronization is still actively studied in research of secure communications, as it is related in the next part of this manuscript, the ideas underlying these approaches still remain disputed by conventional cryptographers. Since this dissertation is devoted to researches lying between chaotic cryptography and traditional cryptography, we will thus focus first on the second paradigm in this part of the dissertation.

Even though chaotic systems exhibit random-like behavior, they are not necessarily cryptographically secure in their discretized form, see e.g. [**?**, **?**]. The reason partly being that discretized chaotic functions do not automatically yield sufficiently complex behavior of the corresponding binary functions, which is, roughly speaking, a prerequisite for cryptographic security. It is therefore essential that the complexity of the binary functions is considered in the design phase such that necessary modifications can be made. Moreover, many suggested PRNG based on chaos suffer from reproducibility problems of the keystream due to the different handling of floating-point numbers on various processors, see e.g. [**?**]. Indeed, as outlined above, chaotic dynamical systems are usually continuous

and hence defined on the real numbers domain. The transformation from real numbers to integers may lead to the loss of the chaotic behavior. The conversion to integers needs a rigorous theoretical foundation.

In this part, some new chaotic pseudorandom bit generator is presented, which can also be used to obtain numbers uniformly distributed between 0 and 1 [1]. These generators are based on discrete chaotic iterations which satisfy Devaney's definition of chaos [**?**]. A rigorous framework is introduced, where topological chaotic properties of the generator are shown. The design goal of these generators was to take advantage of the random-like properties of real-valued chaotic maps and, at the same time, secure optimal cryptographic properties. More precisely, the design was initiated by constructing a chaotic system on the integers domain instead of the real numbers domain.

The quality of a PRNG is proven both by theoretical foundations and empirical validations. Various statistical tests are available in the literature to check empirically the statistical quality of a given sequence. The most famous and important batteries of tests for evaluating PRNGs have been presented previously. They are respectively the TestU01 [**?**], NIST (National Institute of Standards and Technology of the U.S. Government), DieHARD suites [**?**, **?**], and Comparative test parameters [**?**]. For various reasons, a generator can behave randomly according to some of these tests, but it can fail to pass some other tests. So to pass a number of tests as large as possible is important to improve the confidence put in the randomness of a given generator [**?**]. We will now introduce the PRNGs family on which my computer science researches have been focused.

1. Indeed, these bits can be grouped n by n, to obtain the floating part of $x \in [0, 1]$ represented in binary numeral system

4

Review of works

In this chapter, some basic concepts and works are reviewed. Firstly four classic PRNGs are introduced, these generators in latter parts are adapted to CI methods to generate random streams. Then a previous work is recalled : the first and second version of CI PRNGs [**?**] are described.

4.1/ Some well-known pseudorandom generators

We firstly introduce the so called BBS, logistic map, XORshift, and ISAAC PRNGs.

4.1.1/ Blum Blum Shub

The Blum Blum Shub generator [**?**] (usually denoted by BBS) takes the form :

$$x^0 \in [\![1, m-1]\!]$$
$$x^{n+1} = (x^n)^2 \ mod\ m, \quad y^{n+1} = x^{n+1} \ mod\ log(log(m)),$$

where m is the product of two prime numbers (these prime numbers need to be congruent to 3 modulus 4), x^n is an integer value, and it should be an integer that's co-prime to M. y^n is the returned binary sequence. To be noticed, for its part, *log* refers to the logarithm to base 2.

4.1.2/ The logistic map

The logistic map, given by :

$$x^{n+1} = \mu\, x^n(1 - x^n), \text{ with } x^0 \in (0, 1), \mu \in (3.99996, 4],$$

where x is a real number. Logistic map was originally introduced as a demographic model by Pierre François Verhulst in 1838. In 1947, Ulam and Von Neumann [**?**] studied it as

a PRNG. This essentially requires mapping the states of the system $(x^n)_{n\in\mathbb{N}}$ to $\{0,1\}^{\mathbb{N}}$. A simple way for turning x^n to a discrete bit symbol r is by using a threshold function as it is shown in Algo.1. A second usual way to obtain an integer sequence from a real system is to chop off the leading bits after moving the decimal point of each x to the right, as it is obtained in Algo.2.

Algorithm 1 An arbitrary round of logistic map 1

Input : the internal state x (a decimal number)
Output : r (a 1-bit word)

1: $x \leftarrow 4x(1-x)$
2: **if** $x < 0$ **then**
3: $\quad r \leftarrow 0$;
4: **else**
5: $\quad r \leftarrow 1$;
6: return r

Algorithm 2 An arbitrary round of logistic map 2

Input : the internal state x (a decimal number)
Output : r (an integer)

1: $x \leftarrow 4x(1-x)$
2: $r \leftarrow \lfloor 10000000x \rfloor$
3: return r

4.1.3/ XORshift

XORshift is a category of very fast PRNGs designed by George Marsaglia [**?**]. 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 Algo.3 is an example for 32-bit XORshift, it has a period of $2^{32} - 1 = 4.29 \times 10^9$.

4.1.4/ ISAAC

ISAAC is an array-based PRNG and a stream cipher designed by Robert Jenkins (1996) to be cryptographically secure [**?**]. The name is an acronym for Indirection, Shift, Accumulate, Add, and Count. The ISAAC algorithm has similarities with RC4 [**?**]. It uses an array of 256 32-bit integers as the internal state, writes the results to another 256-integer array, from which they are read one at a time until empty, at which point they are recomputed. Since

Algorithm 3 An arbitrary round of XORshift algorithm

Input : the internal state z (a 32-bits word)
Output : y (a 32-bits word)

1: $z \leftarrow z \oplus (z \ll 13)$;
2: $z \leftarrow z \oplus (z \gg 17)$;
3: $z \leftarrow z \oplus (z \ll 5)$;
4: $y \leftarrow z$;
5: return y

it only takes about 19 32-bit operations for each 32-bit output word, it is extremely fast on 32-bit computers.

We give the key-stream procedure of ISAAC in Algo.4. The internal state is x, the output array is r, and the inputs a, b, and c are those computed in the previous round. The value $f(a, i)$ in Algo.4 is a 32-bit word, defined for all a and $i \in \{0, \ldots, 255\}$ as :

$$f(a,i) = \begin{cases} a \ll 13 & \text{if } i \equiv 0 \ mod\ 4, \\ a \gg 6 & \text{if } i \equiv 1 \ mod\ 4, \\ a \ll 2 & \text{if } i \equiv 2 \ mod\ 4, \\ a \gg 16 & \text{if } i \equiv 3 \ mod\ 4. \end{cases} \quad (1)$$

Algorithm 4 An arbitrary round of ISAAC algorithm

Input : a, b, c, and the internal state x, they are 32-bit words
Output : an array r of 256 32-bit words

1: $c \leftarrow c + 1$;
2: $b \leftarrow b + c$;
3: **while** $i = 0, \ldots, 255$ **do**
4: $s \leftarrow x_i$;
5: $a \leftarrow f(a, i) + x_{(i+128)\ mod\ 256}$;
6: $x_i \leftarrow a + b + x_{(x \gg 2)\ mod\ 256}$;
7: $r_i \leftarrow s + x_{(x_i \gg 10)\ mod\ 256}$;
8: $b \leftarrow r_i$;
9: return r

4.2/ CIPRNG, version 1 [?]

4.2.1/ Chaotic iterations as PRNG

This first proposed version of a generator based on chaotic iterations, denoted by CI(PRNG1,PRNG2), 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 pseudorandom bit flow.

Algorithm 5 An arbitrary round of the CI generator Version 1

Input : the internal state x (an array of N 1-bit words)
Output : an array r of N 1-bit words

1: $a \leftarrow PRNG1()$;
2: $m \leftarrow a \; mod \; 2 + c$;
3: **while** $i = 0, \ldots, m$ **do**
4: $\quad b \leftarrow PRNG2()$;
5: $\quad S \leftarrow b \; mod \; \mathsf{N}$;
6: $\quad x_S \leftarrow \overline{x_S}$;
7: $r \leftarrow x$;
8: return r ;

Chaotic iterations are realized as follows. Initial state $x^0 \in \mathbb{B}^{\mathsf{N}}$ is a Boolean vector taken as a seed and strategy $(S^n)_{n\in\mathbb{N}} \in [\![1, \mathsf{N}]\!]^{\mathbb{N}}$ is a sequence produced by PRNG2. 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 pseudorandom bit sequence of our generator, $(m^n)_{n\in\mathbb{N}} \in \mathcal{M}^{\mathbb{N}}$. 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 :

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

or the following integers :

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

4.2.2/ CIPRNG version 1 : the algorithm

The basic design procedure of the novel generator is summed up in Algo.5. The internal state is x, the output array is r. a and b are those computed by PRNG1 and PRNG2. Lastly, k and N are constants and $\mathcal{M} = \{\text{k, k+1}\}$ ($k \geqslant 3\text{N}$ is recommended, see [**?**] for more information).

4.3/ The CIPRNG : Version 2 [?]

After the proof of concept of CIPRNG version 1, a second version of generator based on chaotic iterations has been introduced in [**?**].

4.3.1/ Defining the sequence m^n

The basic idea was to prevent from changing a bit twice between two outputs, reducing by doing so the generation time. To do so, a sequence (m^n) must be introduced, which defines the number of bits to change between two outputs. The output of the sequence (y^n) is uniform in $[\![0, 2^{32}-1]\!]$. 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 $[\![0, 2^N - 1]\!]$, 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 CI PRNG Version 2. 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.*

Each value in $[\![0, 2^3-1]\!]$ must be returned with the same frequency, 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 the probability of $m = i$:

$$P(m^n = i) = \frac{C_N^i}{2^N} \tag{3}$$

Then, here is an example for the (m^n) sequence definition using a selector function g_1 :

$$m^n = g_1(y^n) = \begin{cases} 0 \text{ if } 0 \leqslant \frac{y^n}{2^{32}} < \frac{C_N^0}{2^N}, \\ 1 \text{ if } \frac{C_N^0}{2^N} \leqslant \frac{y^n}{2^{32}} < \sum_{i=0}^{1} \frac{C_N^i}{2^N}, \\ 2 \text{ if } \sum_{i=0}^{1} \frac{C_N^i}{2^N} \leqslant \frac{y^n}{2^{32}} < \sum_{i=0}^{2} \frac{C_N^i}{2^N}, \\ \vdots \quad \vdots \\ N \text{ if } \sum_{i=0}^{N-1} \frac{C_N^i}{2^N} \leqslant \frac{y^n}{2^{32}} < 1. \end{cases} \tag{4}$$

The CI PRNG Version 2 can use any reasonable function as selector like [?] :

$$m^n = g_2(y^n) = \begin{cases} N \text{ if } 0 \leqslant \frac{y^n}{2^{32}} < \frac{C_N^0}{2^N}, \\ N-1 \text{ if } \frac{C_N^0}{2^N} \leqslant \frac{y^n}{2^{32}} < \sum_{i=0}^{1} \frac{C_N^i}{2^N}, \\ N-2 \text{ if } \sum_{i=0}^{1} \frac{C_N^i}{2^N} \leqslant \frac{y^n}{2^{32}} < \sum_{i=0}^{2} \frac{C_N^i}{2^N}, \\ \vdots \quad \vdots \\ 0 \text{ if } \sum_{i=0}^{N-1} \frac{C_N^i}{2^N} \leqslant \frac{y^n}{2^{32}} < 1. \end{cases} \tag{5}$$

In this thesis, $g_1()$ is the selector function unless noted otherwise.

4.3.2/ STRATEGY

The strategy $(S^k) \in [\![1, N]\!]^{\mathbb{N}}$ is generated using a second XORshift sequence $(b^k) \in [\![1, N]\!]^{\mathbb{N}}$ (see [?] for further details). 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 this PRNG proposed in [?], 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 (see [?]).

Let $(d^1, d^2, \ldots, d^N) \in \{0, 1\}^N$ be a mark sequence, such that whenever $\sum_{i=1}^{N} d^i = m^k$, 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 $b = 142\underline{2}33414211\underline{22}34...$ 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 output (because switching the same bit twice leads to the same state).

To check the balance property, a set of 500 sequences are generated with and without decimation, each sequence containing 10^6 bits. Fig.4.1 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 [?].

4.3.3/ CIPRNG VERSION 2 : THE ALGORITHM

The basic design procedure of the novel generator is summed up in Algo.6. The internal state is x, it is an integer of N bits size. The output state is r. a and b are those computed by the two input PRNGs. The value $g_1(a)$ is an integer, defined as in Eq.(4). Lastly, N is a constant defined by the user.

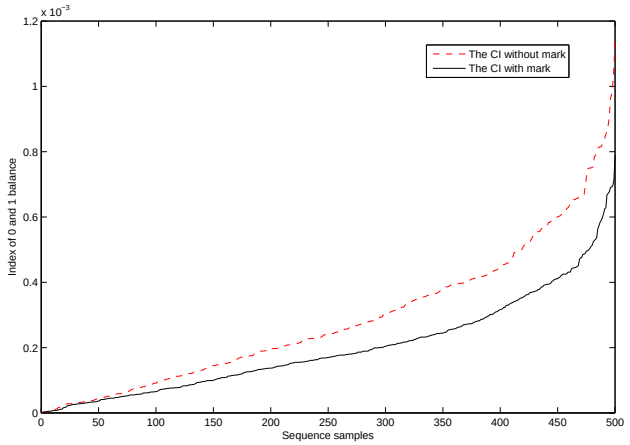


FIGURE 4.1 – Balance property

Algorithm 6 An arbitrary round of the CI generator Version 2

Input : the internal state x (N bits)
Output : a state r of N bits

for $i = 0, \ldots, N$ **do**
  $d_i \leftarrow 0$
$a \leftarrow PRNG1()$
$m \leftarrow f(a)$
$k \leftarrow m$
while $i = 0, \ldots, k$ **do**
  $b \leftarrow PRNG2() \ mod \ N$
  $S \leftarrow b$
  if $d_S = 0$ **then**
    $x_S \leftarrow \overline{x_S}$
    $d_S \leftarrow 1$
  else if $d_S = 1$ **then**
    $k \leftarrow k + 1$
$r \leftarrow x$ return r

Compare to CI Version 1, this latter version provides better statistical performance and speed [**?**].

5

Investigating the Design of new CIPRNGs

In this chapter, some formerly proposed researches on CIPRNGs versions 1 and 2 are deepened. The designs of our two brand new versions of discrete chaotic iterations based pseudorandom number generators, and satisfying Devaney's chaos, are proposed and discussed. Detail operations of the proposed approach are described in this chapter, while their performance and comparative studies will be presented in a next one. The works presented in this chapter have been formerly published in [**?**] and submitted in [**?**, **?**].

5.1/ "LUT" CIPRNG(XORshift,XORshift) Version 3

5.1.1/ Introduction

The LUT (Lookup-Table) CI generator is an improved version of the CIPRNG version 2. The key-ideas are :

1. 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.
2. 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 CIPRNG version 2, 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 equivalence is to reduce the complexity of the proposed PRNG. Indeed the LUT CIPRNG(S,m) can be written as :

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

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.

5.1.2/ 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 CIPRNG version 2 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 Algo.7. 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 divides 32.

Algorithm 7 Generation of sequence b^n

1: $c = n \ mod \ \lfloor 32/N \rfloor$
2: **if** $c = 0$ **then**
3: $\quad a = XORshift()$
4: $b^n = (a \gg (N \times c))\&(2^N - 1)$
5: Return b^n

This Algo.7 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 Eq.(5) or Eq.(4) as follows.

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

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 Eq.(2) 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 Tab.5.1 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 Algo.8, which returns a table $lut1$ such that $m^n = lut1[b^n]$.

Algorithm 8 The LUT-1 table generation

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

TABLE 5.1 – 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

5.1.3/ DEFINING THE 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 :

$$w^n = XORshift2() \ mod \ C_N^m. \tag{3}$$

With this material it is now possible to define the lookup table 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.

An example of such a LUT is shown in Tab.5.2, when Algo.10 gives a concrete procedure to obtain these tables. This procedure makes recursive calls to the function $LUT21$ defined in Algo.9. 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 Tab.5.2 is the $lut2$ obtained in Algo.9 with $N = 4$.

5.1.4/ LUT CI(XORSHIFT,XORSHIFT) ALGORITHM

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

$$x^n = x^{n-1} \oplus \mathcal{S}^n, \tag{4}$$

Algorithm 9 *LUT*21 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 Procedure
```

Algorithm 10 LUT-2 generation

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

TABLE 5.2 – 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 5.3 – 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 Algo.7 and $w^n = XORshift2()\ mod\ C_N^m$. An iteration of this generator is written in Algo.11.

Algorithm 11 LUT CI algorithm

1: $b^n = PRNG1()$
2: $m^n = lut1[b^n]$
3: $w^n = PRNG2()$
4: $S^n = lut2[m][w]$
5: $x = x \oplus S^n$
6: Return x

5.1.5/ LUT CI(XORshift,XORshift) example of use

In this example, $N = 4$ is chosen another time for easy understanding. The initial state of the system x^0 can be seeded by the decimal part t of the current time. With the $t = 484076$, then according to $t = t\ mod\ 16$, we have $x^0 = (0, 1, 0, 0)$ (or $x^0 = 4$).

Algo.8 provides the LUT-1 depicted in Tab.5.1. 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 Algo.10 is computed, leading to the LUT-2 given by Tab.5.2. So chaotic iterations of Algo.11 can be realized, to obtain in this example : 0100100101010001... or 4,9,5,1... As Tab.5.3 shows.

5.2/ The version 4 category of CIPRNGs

5.2.1/ XOR CIPRNG

Instead of updating only one cell at each iteration as the previous versions of our CIPRNGs, we can try to choose a subset of components and to update them together. Such an attempt leads to a kind of merger of the two random sequences. When the updating function is the vectorial negation, this algorithm can be rewritten as follows [**?**] :

$$\begin{cases} x^0 \in [\![0, 2^{\mathsf{N}} - 1]\!], S \in [\![0, 2^{\mathsf{N}} - 1]\!]^{\mathbb{N}} \\ d^n = S^n \\ \forall n \in \mathbb{N}^*, x^n = x^{n-1} \oplus d^n, \end{cases} \tag{5}$$

and this rewriting can be understood as follows. The $n-$th term S^n of the sequence S, which is an integer of N binary digits, presents the list of cells to update in the state x^n of the system (represented as an integer having N bits too). More precisely, the $k-$th component of this state (a binary digit) changes if and only if the $k-$th digit in the binary decomposition of S^n is 1. This generator has been called XOR CIPRNG by the AND team, which has introduced and studied it in [**?**, **?**].

The single basic component presented in Eq.(5) is of ordinary use as a good elementary brick in various PRNGs. It corresponds to the discrete dynamical system in chaotic iterations. Such XOR CIPRNG has inspired us to introduce a more general category of CIPRNGs, presented below.

5.2.2/ CIPRNG version 4 : the algorithm

It is possible to add more complexity in updating the subset at each iteration in Eq. 5. When the updating function is the vectorial negation, this algorithm can be written as follows (Algo.12) :

Algorithm 12 An arbitrary round of the version 4 CI generator

Input : the internal state x (N bits)
Output : a state r of N bits

1: **for** $i = 1, \ldots, M$ **do**
2: $S(i) = PRNG2\_i()$
3: $T = PRNG1()$
4: $r = x \oplus g_2(S(1), S(2), \ldots S(M))$,
5: return r

$S(1), S(2), ..., S(M)$ are M pseudorandom number sequences generated by M XORshifts, T^n is obtained by using a cryptographically secure PRNG like the BBS. $(t_1^n, t_2^n, \ldots, t_M^n) \in$

$\{0,1\}^M$ is the binary representation of the 2^M-bit number T^n. Indeed, (T^n) sequence's aim is to decimate the sequences $S(1), S(2), ..., S(M)$, with a *bitwise exclusive or* ($\oplus$), according to the following decimation rule :
– if $t_i^n = 0$, then $S^n(i)$ is discarded,
– else $S^n(i)$ is kept for *bitwise exclusive or* computing.
In brief, the produced output sequence x^n, based on chaotic iterations, is updated by a *bitwise exclusive or* of an irregular decimation of $S(1), S(2), ..., S(M)$, according to the bits of T^n.

The M terms $S^n(1), ..., S^n(M)$ of the $n-$th iterate of sequences $S(1), S(2), ..., S(M)$ are integers of N bits. Each term T^n of sequence T is an integer having M binary digits. Such a term T^n presents the list of cells to update in the state x^n of the system, which is an integer of N bits too. This update is provided by the function $g_2(S^n(1), S^n(2), ..., S^n(M), T^n)$, which is defined by the Algo.13. Indeed, each bit in T^n decides whether its corresponding $S^n(i)$ is used in the *bitwise exclusive or* computation defining x^n. More precisely, the $k-$th binary digit of x^{n-1} changes if and only if the $k-$th digit in the binary decomposition of $g_2(S^n(1), S^n(2), ..., S^n(M), T^n)$ is 1.

Algorithm 13 The $g_2(S^n(1), S^n(2), ..., S^n(M), T^n)$ function
Input : sequences $S^n(1), S^n(2), ..., S^n(M)$, and T^n
output : a state r (N bits)

```
r = 0
M = size of T^n
for i = 1 ... M do
    if T^n&(2^{i-1}) ≠ 0 then
        r = r ⊕ S^n(i)
return r
```

5.2.3/ CI Version 4 example of use

To better understand the CIPRNG Version 4 algorithm, an example is represented in this subsection. The same 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 $x^0 = 4$).

Here BBS is used as the T PRNG in Algo.12, m for BBS is set as a $32 - bit$ word, and the least four significant bits are treated as its output. Then four XORshift PRNGs used as $S(1), S(2), S(3)$ and $S(4)$ (in Algo.12, $M = 4$). Their return values are described as below :
– $T = 8, 11, 9, 7, 3$
– $S(1) = 6, 1, 0, 2, 2, ...$
– $S(2) = 7, 14, 5, 2, 3, ...$
– $S(3) = 12, 15, 3, 4, 1, ...$

n	1	2	3	...	...
y_n	8	11	9	...	...
$PRNG[1-4]$	$[6,7,12,10]$	$[1,14,15,6]$	$[0,5,3,2]$		
X^n	$X^1 = X^0 \oplus 10 = 14$	$X^2 = X^1 \oplus 1 \oplus 15 \oplus 6 = 6$	$X^3 = X^2 \oplus 2 \oplus 0 = 4$	...	...
0	1	0	0	...	...
1	1	1	0	...	...
0	1	1	1	...	...
0	0	0	0	...	...
Output = 111001100010 ...					

– $S(4) = 10, 6, 2, 3, 2, ...$

Based on the chaotic iteration of Algo.12, we can have an example of output as Tab.5.2.3. The output is returned as 111001100010... or $14, 6, 4, ...$

5.2.4/ Efficient PRNG based on CI : Version 4

Tab.5.4 describes an efficient PRNG based on chaotic iteration having a perfect statistical profile. It can be divided into two parts, as explained below.

The first part is based on Algo.12. This part is very suitable for FPGA as it can be easily arranged to be processed in parallel. In the proposed design, the BBS secured generator has been chosen due to its simplicity. Additionally, as stated in the previous section, BBS PRNG can turn to be cryptographically secure. Due to its slowness, this BBS is used to compute the T sequence of Algo.12. The size of m is 32 bits. It is well known that the $log(log(m))$ least significant bits can be securely extracted at each iteration of the BBS [**?**]. So we set $M = 3$, leading to the selection of two 64-bit word output XORshifts playing the role of S. They are denoted $XORshift1$ and $XORshift2$ in Tab.5.4. Each XORshift output is separated into two 32 bits blocks, leading to four 32 bits numbers. Three of them (namely, the two 32 bits blocks of $XORshift1$ and the first one of $XORshift2$) are controlled by the bits outputted by the BBS according to Algo.12. The last 32 bits block, on its part, is used in the second part of the algorithm. More precisely, if one bit in the bbs output is 0, then the corresponding 32 bits number is not used during the *exclusive or* processing, whereas it is considered if this BBS bit is 1.

According to our experiments, the sole first part of the algorithm cannot produce a statistically perfect output. Following the approach detailed in [**?**], we have used the chaotic iterations to improve the statistical behavior of the proposed generator. Hence, the second part of the algorithm consists in using the last 32 bits block of $XORshift2$ to realize Eq.(5) on the output of the first part.

This algorithm has a very similar design than to the efficient GPU CI version presented in [**?**], which has successfully passed the stringent TestU01 battery of statistical tests [**?**]. However, in the GPU version, no BBS (which is cryptographically secure) is used to determine which bits in the most significant binary block of size 32 of XORshift will be used in the process.

TABLE 5.4 – Efficient pseudorandom generator designed for FPGAs

Input : x (a 32-bit word)
Output : r (a 32-bit word)
$t1 = XORshift1()$; $t2 = XORshift2()$; $t4 = bbs()$; **if** $t4\&1 \neq 0$; **then** $x = x \oplus (t1\&0x0fffffffff)$; **if** $t4\&2 \neq 0$; **then** $x = x \oplus (t1 >> 32)$; **if** $t4\&4 \neq 0$; **then** $x = x \oplus (t2\&0x0fffffffff)$; $x = x \oplus (t2 >> 32)$; $r = x$; **return** r;
An arbitrary round of the algorithm

6

Randomness Quality of CIPRNGs

In this chapter, for having comparison criteria, classical PRNGs are evaluated by the statistical tests introduced in Section 2.3, namely the NIST, DieHARD, Comparative tests, and TestU01 test suites. Regarding these results, we will be able to easily verify the improvements of mixing such generators using chaotic iterations. Statistical performances of the CIPRNG versions 3 and 4 are then given in a second part of this chapter. Finally four versions of CIPRNGs are assembled together, and a comprehensive comparison among them is produced using the aforementioned tests (for the sake of fairness, all the CIPRNGs have received two XORshifts as inputted generators). These results have been formerly published in [**?**, **?**, **?**].

6.1/ Test results for some PRNGs

We present in this section the scores obtained by four well known PRNGs against the usual batteries of tests. These generators are respectively the BBS, Logistic, XORshift, and ISAAC ones (details for these generators have been given in Section 4.1).

Table 6.1 contains the $\mathbb{P}_T$ values corresponding to sequences produced by BBS, Logistic map, XORshift, and ISAAC respectively. We can see that, except for ISAAC, all the proposed generators failed in passing the NIST battery. Results are particularly bad for the BBS generator, although this PRNG is known to be cryptographically secure. It is not contradictory, as these bad scores are due to the small prime numbers that have been used here as parameters of this PRNG.

Table 6.2 contains, for its part, the results derived from applying the DieHARD battery of tests to the four generators considered in this section. Another time, ISAAC is the sole generator able to pass the whole battery, whereas the logistic map and the XORshift generators present correct results. BBS, for its part, shows another time a very bad statistical profile.

In Table 6.3 is given another comparison of the BBS, Logistic map, XORshift, and ISAAC generators using the comparative test parameters previously introduced. Finally, Table 6.4

TABLE 6.1 – NIST SP 800-22 test results ($\mathbb{P}_T$)

Test name	BBS	Logistic	XORshift	ISAAC
Frequency (Monobit) Test	0.32435	0.53414	0.14532	0.67868
Frequency Test within a Block	0.00000	0.00275	0.45593	0.10252
Runs Test	0.00000	0.00000	0.21330	0.69931
Longest Run of Ones in a Block Test	0.00000	0.08051	0.28966	0.43727
Binary Matrix Rank Test	0.00000	0.67868	0.00000	0.89776
Discrete Fourier Transform (Spectral) Test	0.00000	0.57490	0.00535	0.51412
Non-overlapping Template Matching Test*	0.00000	0.28468	0.50365	0.55515
Overlapping Template Matching Test	0.00000	0.10879	0.86769	0.63711
Universal Statistical Test	0.00000	0.02054	0.27570	0.69931
Linear Complexity Test	0.04335	0.79813	0.92407	0.03756
Serial Test* (m=10)	0.00000	0.41542	0.75792	0.32681
Approximate Entropy Test (m=10)	0.00000	0.02054	0.41902	0.30412
Cumulative Sums (Cusum) Test*	0.00000	0.60617	0.81154	0.36786
Random Excursions Test*	0.00000	0.53342	0.41923	0.50711
Random Excursions Variant Test*	0.00000	0.28507	0.52833	0.40930
Success	2/15	14/15	14/15	15/15

TABLE 6.2 – Results of DieHARD battery of tests

No.	Test name	BBS	Logistic	XORshift	ISAAC
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	Fail	Pass	Pass	Pass
4	Parking Lot	Fail	Pass	Pass	Pass
5	Birthday Spacing	Fail	Pass	Pass	Pass
6	Count the ones 1	Fail	Pass	Fail	Pass
7	Binary Rank 6×8	Fail	Pass	Pass	Pass
8	Binary Rank 31×31	Fail	Pass	Fail	Pass
9	Binary Rank 32×32	Fail	Fail	Fail	Pass
10	Count the ones 2	Fail	Pass	Pass	Pass
11	Bit Stream	Fail	Pass	Pass	Pass
12	Craps Wins	Fail	Pass	Pass	Pass
	Throws	Fail	Pass	Pass	Pass
13	Minimum Distance	Fail	Pass	Pass	Pass
14	Overlapping Perm.	Fail	Pass	Pass	Pass
15	Squeeze	Fail	Pass	Pass	Pass
16	OPSO	Fail	Pass	Pass	Pass
17	OQSO	Fail	Pass	Pass	Pass
18	DNA	Fail	Fail	Pass	Pass
	Number of tests passed	2	16	15	18

gives the results derived from applying the TestU01 battery to these PRNGs.

From the results shown in the four tables, statistical quality of each generator considered in this section can be basically deduced. BBS used in our conditions is not able to produce reasonable sequences, the bad scores of these tests mainly relates to its too small period value. For the logistic map, it sounds better than BBS. However, this generator only successfully pass the comparative parameters tests. Furthermore, its speed is far from dominating the comparison. XORshift failed to succeed the Binary Matrix Rank Test (NIST). This test focuses on the rank of disjoint sub-matrices extracted from the entire sequence. Note that this test also appears in the DieHARD battery. Seven tests were failed with a $p - value$

Table 6.3 – Comparative test parameters with a 10^7 bits sequence

Method	Threshold values	BBS	Logistic	XORshift	ISAAC
Monobit	3.8415	0.3485	0.1280	1.7053	0.1401
Serial	5.9915	2.0079	0.1302	2.1466	0.1430
Poker	316.9194	387.7216	240.2893	248.9318	236.8670
Runs	55.0027	33.2067	26.5667	18.0087	34.1273
Autocorrelation	1.6449	-0.7603	0.0373	0.5099	-2.1712

Table 6.4 – TestU01 Statistical Test

Test name	Battery	Parameters	BBS	Logistic	XORshift	ISAAC
Rabbit	32×10^9 bits	38	26	21	14	0
Alphabit	32×10^9 bits	17	9	16	9	0
Pseudo DieHARD	Standard	126	8	0	2	0
FIPS_140_2	Standard	16	0	0	0	0
Small Crush	Standard	15	10	4	5	0
Crush	Standard	144	117	95	57	0
Big Crush	Standard	160	134	125	55	0
Number of failures		516	304	261	146	0

practically equal to 0 or 1 for both the logistic map and the XORshift generators in the TestU01 battery. Such results mean that the null hypothesis H_0 must be rejected for these two bit streams, where H_0 is : "for each integer $t > 0$, the vector $(u_0, ..., u_{t-1})$ is uniformly distributed over the t-dimensional unit cube $[0, 1]^t$". Finally, ISAAC plays best in all.

6.2/ Test results and comparative analysis for the CIPRNG version 3

The generator version 3 of the CIPRNG family is based on a Lookup table (LUT). We test it here with the parameter N equal to 4. We can conclude from Table 6.5 that, except for the mixture of BBS and XORshift, all possible couples have successfully passed the NIST statistical test suite. Remark that the use of BBS, having poor statistical performances, leads to outputs of LUT-1 not very uniformly distributed.

Table 6.6 gives the results derived from applying the DieHARD battery of tests to the PRNGs considered in this chapter. For the same reasons as in the NIST test suite's results, the CIPRNG constituted by the mixture of BBS and XORshift is not able to pass all the

TABLE 6.5 – NIST SP 800-22 test results ($\mathbb{P}_T$) for Version 3 LUT CI algorithms

Test name	Version 3 LUT CI		
	XORshift + XORshift	ISAAC + XORshift	BBS + XORshift
Frequency (Monobit) Test	0.32435	0.33171	0.00000
Frequency Test within a Block	0.85643	0.42327	0.13233
Runs Test	0.11623	0.31908	0.00000
Longest Run of Ones in a Block Test	0.74254	0.86688	0.00000
Binary Matrix Rank Test	0.23224	0.88317	0.90311
Discrete Fourier Transform (Spectral) Test	0.12316	0.34578	0.59559
Non-overlapping Template Matching Test*	0.43295	0.32637	0.00000
Overlapping Template Matching Test	0.31472	0.55915	0.00000
Universal Statistical Test	0.37864	0.24925	0.06282
Linear Complexity Test	0.65723	0.31793	0.94630
Serial Test* (m=10)	0.43532	0.55190	0.00000
Approximate Entropy Test (m=10)	0.34254	0.12482	0.00000
Cumulative Sums (Cusum) Test*	0.11272	0.04065	0.14139
Random Excursions Test*	0.02003	0.32275	0.34625
Random Excursions Variant Test*	0.43554	0.234294	0.55048
Success	15/15	15/15	8/15

DieHARD battery.

We show in Table 6.7 a "comparative test parameters" comparison between the three following CIPRNGs (version 3) : CI(XORshift, XORshift), CI(ISAAC, XORshift), and CI(BBS, XORshift). The tests in this table consider sequences having 10^7 bits long. Again, the couple (BBS, XORshift) has the worst performance. Table 6.8, for its part, contains the results derived from applying the TestU01 battery to the PRNGs considered in this challenge.

The results of the TestU01, NIST, Comparative test parameters, and DieHARD batteries of tests confirm that the CIPRNGs version 3 are all able to pass these tests. They are thus better than the version 1 while using less computational resources than the version number 2. Furthermore, when using the BBS generator, this version 3 is cryptographically secure. However, in that situation, performances are completely deflated. Finally, man can remark that performances are good when using ISAAC as inputted generator. However, ISAAC is very hard to implement in hardware, while hardware implementation of CIPRNGs if one of the goal of this thesis.

From this tests study we can finally conclude that, by using some well-defined Lookup Table and due to the rewrite of the way to generate strategies, the generator version 3 based on chaotic iterations works faster and has a better statistical profile than the other generators previously tested in this chapter. Additionally, the speed of LUT CI is largely improved compared to the CIPRNGs version 1 and 2, and another time this version 3 may utilize any reasonable RNG as inputs (not necessarily XORshift, BBS, or ISAAC).

6.3/ Tests results and comparative analysis for the CIPRNG version 4

We now investigate the case of the new CIPRNG version 4, by starting first to regard the NIST battery. All the generators of this family tested against this battery used a $N = 32$ bits format. We can conclude from the results summarized in Table 6.9 that all the PRNGs of this family have successfully passed the NIST statistical test suite. Indeed, even when using the deflated BBS, the CIPRNG version 4 using the couple (BBS, XORshift) still can provide a reasonable statistical profile.

Table 6.10 gives the results derived from applying the DieHARD battery of tests to the PRNGs considered in this chapter. Results are closed to the ones produced by the NIST test suite. In particular, we can see that the mixing of BBS and XORshift shows better performance than in the other CIPRNG versions, and that all generators are successfully pass these tests. We then produce in Table 6.11 a comparison between CI(XORshift, XORshift), CI(ISAAC, XORshift), and CI(BBS, XORshift), all in version 4, using a "comparative test parameters" approach. According to these 10^7 bits long sequence tests, the considered generators all show very good statistical performances. Finally, Table 6.12 gives the results against TestU01. We can see no failure at all, which is very remarkable.

TABLE 6.6 – Results of DieHARD battery of tests for Version 3 LUT CI algorithms (N = 4)

No.	Test name	Version 3 CI			
		Logistic + Logistic	XORshift + XORshift	ISAAC + XORshift	BBS + XORshift
1	Overlapping Sum	Pass	Pass	Pass	Fail
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	Fail
4	Parking Lot	Pass	Pass	Pass	Pass
5	Birthday Spacing	Pass	Pass	Pass	Fail
6	Count the ones 1	Pass	Pass	Pass	Fail
7	Binary Rank 6×8	Pass	Pass	Pass	Pass
8	Binary Rank 31×31	Pass	Pass	Pass	Fail
9	Binary Rank 32×32	Pass	Pass	Pass	Fail
10	Count the ones 2	Pass	Pass	Pass	Fail
11	Bit Stream	Pass	Pass	Pass	Pass
12	Craps Wins	Pass	Pass	Pass	Fail
	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	Fail
17	OQSO	Pass	Pass	Pass	Fail
18	DNA	Pass	Pass	Pass	Fail
	Number of tests passed	18	18	18	8

TABLE 6.7 – Comparative test parameters for Version 3 CI(X,Y) with a 10^7 bits sequence (N = 4)

Method	Threshold values	Version 3 CI		
		XORshift + XORshift	ISAAC + XORshift	BBS + XORshift
Monobit	3.8415	3.5689	0.9036	1.5788
Serial	5.9915	3.5765	1.1229	3.378
Poker	316.9194	123.6831	173.8604	209.3320
Runs	55.0027	28.4237	40.4606	38.4153
Autocorrelation	1.6449	0.3403	0.1245	-2.0276

TABLE 6.8 – TestU01 Statistical Test for Version 3 CI algorithms (N = 4)

Test name			Version 3 CI		
			Logistic + Logistic	ISAAC + XORshift	BBS + XORshift
Rabbit	32×10^9 bits	38	0	0	18
Alphabit	32×10^9 bits	17	0	0	8
Pseudo DieHARD	Standard	126	0	0	11
FIPS_140_2	Standard	16	0	0	0
Small Crush	Standard	15	0	0	7
Crush	Standard	144	0	0	51
Big Crush	Standard	160	0	0	77
Number of failures			0	0	165

TABLE 6.9 – NIST SP 800-22 test results ($\mathbb{P}_T$) for Version 4 CI algorithms

Test name	Version 4 CI		
	XORshift + XORshift	ISAAC + XORshift	BBS + XORshift
Frequency (Monobit) Test	0.21414	0.43622	0.24563
Frequency Test within a Block	0.23423	0.43536	0.13233
Runs Test	0.56471	0.23425	0.23562
Longest Run of Ones in a Block Test	0.33252	0.86688	0.12346
Binary Matrix Rank Test	0.01450	0.25689	0.90311
Discrete Fourier Transform (Spectral) Test	0.25462	0.32324	0.59559
Non-overlapping Template Matching Test*	0.79521	0.32637	0.03984
Overlapping Template Matching Test	0.69342	0.55915	0.13839
Universal Statistical Test	0.44654	0.24925	0.06282
Linear Complexity Test	0.97319	0.31793	0.54630
Serial Test* (m=10)	0.58993	0.55190	0.98234
Approximate Entropy Test (m=10)	0.39284	0.12482	0.12345
Cumulative Sums (Cusum) Test*	0.43582	0.04065	0.14139
Random Excursions Test*	0.92001	0.32275	0.34625
Random Excursions Variant Test*	0.24567	0.234294	0.55048
Success	15/15	15/15	15/15

TABLE 6.10 – Results of DieHARD battery of tests for Version 4 CI algorithms

No.	Test name	Version 4 CI			
		Logistic + XORshift	XORshift + XORshift	ISAAC + XORshift	BBS + XORshift
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

TABLE 6.11 – Comparative test parameters for Version 4 CI(X,Y) with a 10^7 bits sequence

Method	Threshold values	Version 3 CI		
		XORshift + XORshift	ISAAC + XORshift	BBS + XORshift
Monobit	3.8415	1.0689	0.9036	0.5788
Serial	5.9915	0.5765	1.0229	1.378
Poker	316.9194	223.6831	133.8604	182.3320
Runs	55.0027	28.4237	13.4606	12.4153
Autocorrelation	1.6449	1.3403	0.1845	-0.7276

TABLE 6.12 – TestU01 Statistical Test for Version 4 CI algorithms (N = 4)

Test name			Version 4 CI		
			Logistic + XORshift	ISAAC + XORshift	BBS + XORshift
Rabbit	32×10^9 bits	0	0	0	0
Alphabit	32×10^9 bits	0	0	0	0
Pseudo DieHARD	Standard	0	0	0	0
FIPS_140_2	Standard	0	0	0	0
Small Crush	Standard	0	0	0	0
Crush	Standard	0	0	0	0
Big Crush	Standard	0	0	0	0
Number of failures		0	0	0	0

The results of the TestU01, NIST, Comparative test parameters, and DieHARD batteries of tests confirm that the proposed chaotic iteration based generators are all able to pass these tests. These results have been obtained on an INTEL i5 dual core computer, with 2.3 GHz CPUs and 4GB of RAM memory. As only 7 hours have been required to finish the biggest test (namely, the BigCrush in TestU01), we can claim that this new version of a secured CIPRNG is very efficient. Indeed, we can conclude from these tests that the CIPRNG version 4 family realizes a great compromise among security, statistical performances, and efficiency. It thus can be considered as very suitable both for software and hardware implementations.

6.4/ Assessment of four versions of CIPRNGs schemes using XORshift Generator

In this section, we investigate more deeply the statistical comparison between the four versions of CIPRNGs under statistical aspects. To do so, all the tested CIPRNGs are embedding only XORshifts as inputted generators, and the values of the parameter N are the most optimized according to previous experiments, that is, $N = 4$ for versions 1 and 3, and $N = 32$ for the other versions.

6.4.1/ NIST evaluation

We can firstly remark from Table 6.13 that XORshift has failed 1 test, whereas all the versions of CI(XORshift, XORshift) have successfully passed the NIST statistical test suite.

6.4.2/ Diehard

Table 6.14 gives the results derived from applying the DieHARD battery of tests to the PRNGs considered in this section. As it can be observed, the results of the individual tests "Count the ones 1", "Binary Rank 31×31", and "Binary Rank 32×32" show that, in the random numbers obtained with the XORshift generator alone, only the least significant bits seem to be independent. This explains the poor behavior of this PRNG in the aforementioned basic tests that evaluate the independence of real numbers. But the generator based on discrete chaotic iterations (CIPRNGs versions 1-4) can pass all the DieHARD battery of tests. This proves that the statistics of the given generator have been improved thanks to chaotic iterations.

6.4.3/ Comparative test parameters

We show in Table 6.15 a comparison between the CIPRNG(XORshift, XORshift) versions 1-4 and a simple XORshift. Time (in seconds) is related to the duration needed by each

TABLE 6.13 – NIST SP 800-22 test results (N = 4 for Version 1 and 3 CIPRNG,N = 32 for XORshift generator, Version 2 and 4 CIPRNG)

PRNG	Classic	CI PRNG versions			
Method	XORshift	1	2	3	4
FT	0.1453	0.5955	0.4744	0.3257	0.8841
FBT	0.4553	0.5524	0.8970	0.1231	0.4197
RT	0.2134	0.4551	0.8161	0.9826	0.0253
LROBT	0.2890	0.0126	0.7398	0.4892	0.3346
BMRT	0.0000	0.6126	0.2621	0.1005	0.4456
DFTT	0.0051	0.0102	0.1071	0.8331	0.6433
NOTMT*	0.5036	0.5322	0.4499	0.3031	0.7001
OTMT	0.8676	0.3345	0.5141	0.0925	0.7739
MUST	0.2757	0.0329	0.6786	0.6642	0.5031
LCT	0.9240	0.4011	0.6579	0.5173	0.8077
ST* (m=10)	0.7579	0.0133	0.4253	0.3301	0.9281
AET (m=10)	0.4190	0.1373	0.6371	0.7181	0.0345
CST*	0.8115	0.0464	0.2796	0.3872	0.4011
RET*	0.4192	0.5036	0.2874	0.9381	0.1963
REVT*	0.5283	0.3477	0.4866	0.4026	0.5120
Success	14/15	15/15	15/15	15/15	15/15

TABLE 6.14 – Results of DieHARD battery of tests

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

algorithm to generate a 2×10^5 bits long sequence. The tests have been conducted using the same computer and compiler with the same optimization settings for both algorithms (i5 dual cpu, 2300MHz, 4GB memory, gcc compiler, and UBUNTU system), in order to make the comparisons as fair as possible.

As a comparison of the overall stability of these PRNGs, similar tests have been computed for different sequence lengths (see Fig.6.1 - Fig.6.5). For the monobit test comparison (Fig.6.1), XORshift and CI(XORshift, XORshift) PRNGs versions 2-4 present the same

TABLE 6.15 – Comparison with CIPRNGs(XORshift,XORshift) for a 2×10^5 bits sequence(N = 32)

PRNGS		Classic	CI PRNG versions			
Method	The threshold values	XORshift	1	2	3	4
Monobit	3.84	1.71	2.77	0.33	0.02	0.60
Serial	5.99	2.15	2.88	0.74	1.05	0.04
Poker	316.91	248.93	222.36	262.82	217.5	209.9
Runs	55.00	18.01	21.92	16.78	22.77	17.34
Autocorrelation	1.64	0.50	0.02	0.08	1.33	0.81
Time	Second	0.10s	0.41s	0.32s	0.24s	0.12s

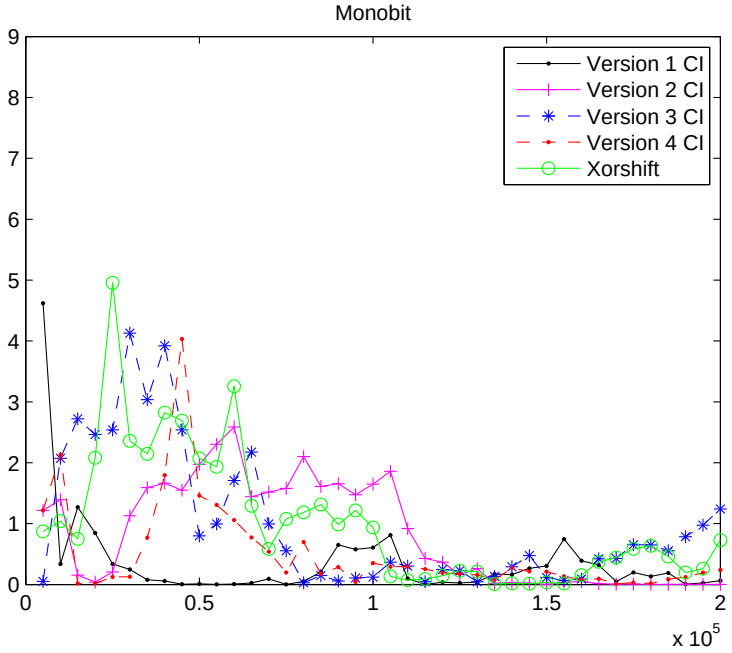


FIGURE 6.1 – Comparison of monobits tests

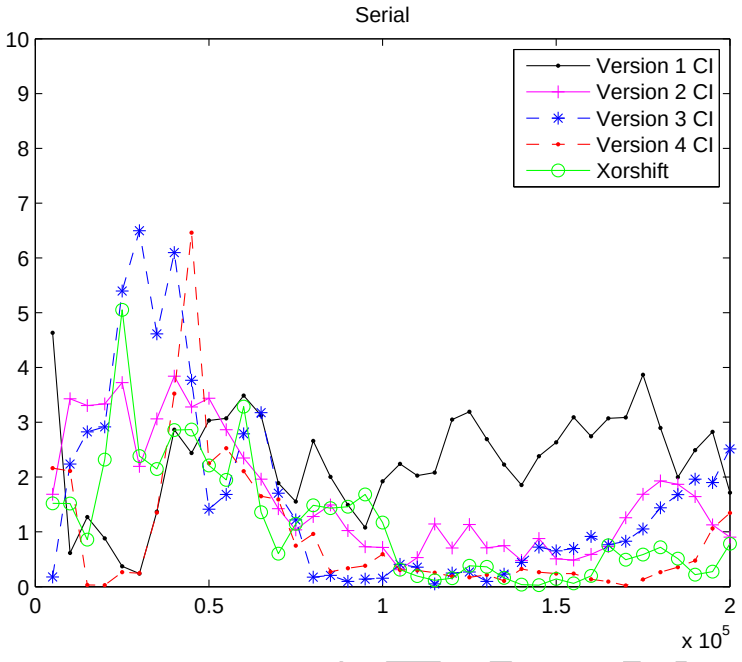


FIGURE 6.2 – Comparison of serial tests

values. They are stable in a low level that never exceeds 1.2. Indeed, the new generators distributes very randomly the zeros and ones, whatever the length of the desired sequence. It can also be remarked that the XORshift generator presents the worst performance, but the values are within the standard boundary.

Figure 6.2 shows the serial test comparison. The CIPRNGs outputs outperform this test, for lengths between 2×10^4 and 5×10^4. We can remark too that versions 2 and 3 express a little overflow, even if all generators occurrences of 00, 01, 10, and 11 are very closed to each other.

The poker test comparison with $m = 8$ is shown in Fig. 6.3. For some lengths, XORshift is not very stable, whereas all the CIPRNGs present good scores (values are lower than the given threshold). 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. By doing so, the performances of our generators in the poker test can be improved.

The graphs of the CI generators are the most stable ones during the runs test comparison (Fig.6.4). Moreover, this trend is reinforced when the lengths of the tested sequences are increased. The comparison of autocorrelation tests is presented in Fig. 6.5. We can see that the CI generators clearly dominate these tests.

As a conclusion of all this study, we can finally claim that the CIPRNGs, whose newer versions run faster than theirs former ones, outperform all of the other generators in these statistical tests, especially when producing long output sequences.

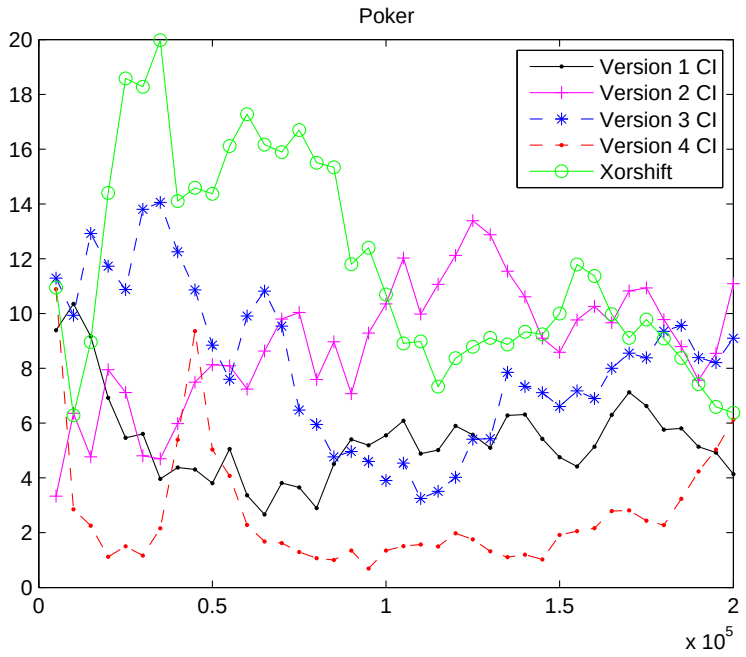


FIGURE 6.3 – Comparison of poker tests

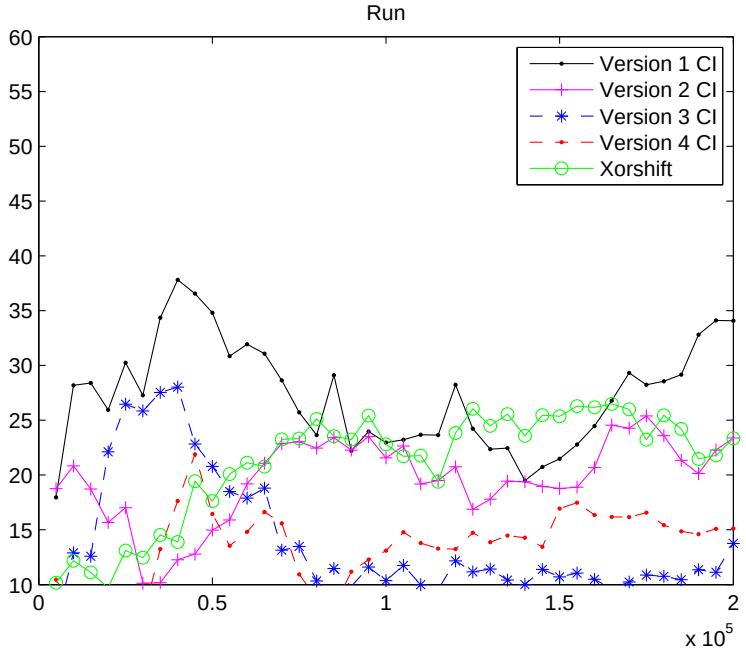


FIGURE 6.4 – Comparison of runs tests

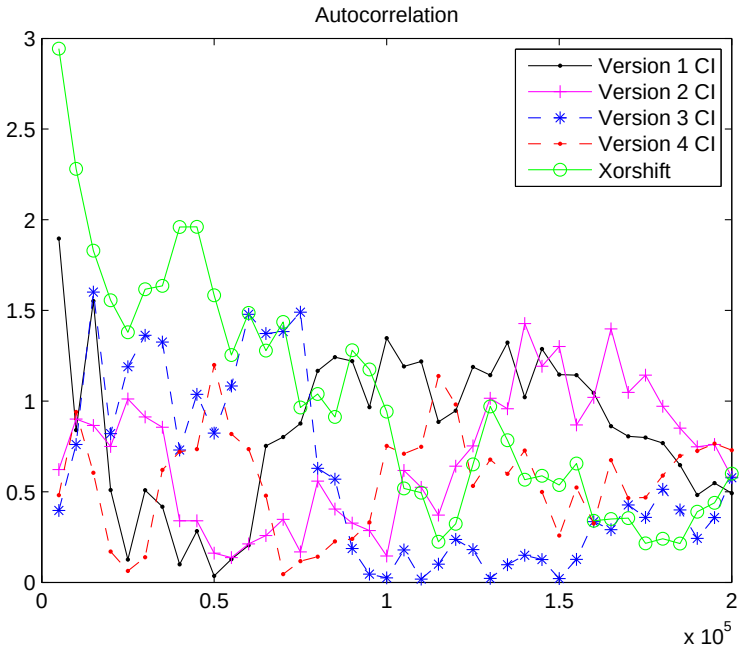


FIGURE 6.5 – Comparison of autocorrelation tests

7

An optimization technique on pseudorandom generators based on chaotic iterations

In this chapter, the behavior of our CIPRNGs regarding the statistics of the inputted generators are carried out systematically, and the results are discussed. Here, CIPRNG Version 1, Version 2 and XOR CIPRNG method are applied to experimented. Indeed PRNGs are often based on modular arithmetic, logical operations like bitwise exclusive or (XOR), and on circular shifts of bit vectors. However the security level of some PRNGs of this kind has been revealed inadequate by today's standards. Since different biased generators can possibly have their own side effects when inputted into our mixed generators, it is normal to enlarge the set of tested inputted PRNGs, to determine if the observed improvement still remains. We will thus show in this chapter that the intended statistical improvement is really effective for all of these most famous generators. To be notice, we have formally published the works in [**?**].

7.1/ Presentation of new well known generators

7.1.1/ Introduction

Knowing that there is no universal generator, it is strongly recommended to test a stochastic application with a large set of different PRNGs [**?**]. Such generators can be classified in four major classes : linear generators, lagged generators, inversive generators, and mix generators :

– **Linear generators**, defined by a linear recurrence, are the most commonly analyzed and utilized generators. The main linear generators are LCGs and MLCG.
– **Lagged generators** have a general recursive formula that use various previously computed terms in the determination of the new sequence value.
– **Inversive congruential generators** form a recent class of generators that are based on the principle of congruential inversion.

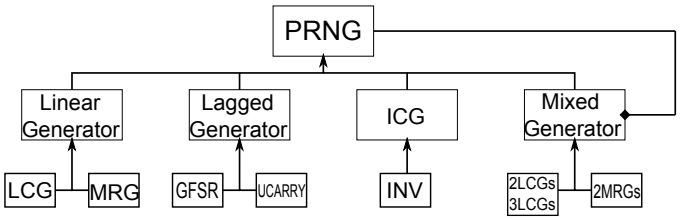


FIGURE 7.1 – Ontological class hierarchy of PRNGs

– **Mixed generators** result from the need for sequences of better and better quality, or at least longer periods. This usually leads to mix different types of PRNGs, as follows : $x^i = y^i \oplus z^i$

For instance, inversive generators are very interesting for verifying simulation results obtained with a linear congruential generator (LCG), because their internal structure and correlation behavior strongly differ from what LCGs produce. Since these generators have revealed several issues, some scientists refrain from using them. In what follows, chaotic properties will be added to these PRNGs, leading to noticeable improvements observed by statistical tests. Let us firstly explain with more details the generators studied in this research work (for a synthetic view, see Fig. 7.1).

7.1.2/ Details of some Existing Generators

Here are the modules of PRNGs we have chosen to experiment.

7.1.2.1/ LCG

This PRNG implements either the simple or the combined linear congruency generator (LCGs). The simple LCG is defined by the recurrence :

$$x^n = (ax^{n-1} + c) \; mod \; m \tag{1}$$

where a, c, and x^0 must be non-negative and less than m [**?**] integers. In what follows, 2LCGs and 3LCGs refer as two (resp. three) combinations of such LCGs. For further details, see [**?**].

7.1.2.2/ MRG

This module implements multiple recursive generators (MRGs), based on a linear recurrence of order integer k, modulo m [**?**] :

$$x^n = (a^1 x^{n-1} + \; ... \; + a^k x^{n-k}) \; mod \; m \tag{2}$$

Combination of two MRGs (referred as 2MRGs) is also used in this chapter.

7.1.2.3/ UCARRY

Generators based on linear recurrences with carry are implemented in this module. This includes the add-with-carry (AWC) generator, based on the recurrence :

$$\begin{aligned} x^n &= (x^{n-r} + x^{n-s} + c^{n-1}) \; mod \; m, \\ c^n &= (x^{n-r} + x^{n-s} + c^{n-1})/m, \end{aligned} \tag{3}$$

with output : x^i/m. The k initial values $(x^0, ..., x^{k-1})$ is firstly set with random integers. the $k = max\{r, s\}$ (the bigger one assigned to k), and c contains c^0. Restrictions : $0 < s, 0 < r, r \neq s$ and $c = 0 or 1$.

The SWB generator, having the recurrence :

$$\begin{aligned} x^n &= (x^{n-r} - x^{n-s} - c^{n-1}) \; mod \; m, \\ c^n &= \begin{cases} 1 & \text{if } (x^{i-r} - x^{i-s} - c^{i-1}) < 0 \\ 0 & \text{else,} \end{cases} \end{aligned} \tag{4}$$

with output is x^i/m. The vector $S\,[0..(k-1)]$ contains the k initial values $(x^0, ...x^{k-1})$, where $k = max\{r, s\}$, and c contains $c0$. Restrictions : $0 < s, 0 < r, r6 = s$ and $c = 0$ or 1. And the SWC generator designed by R. Couture, which is based on the following recurrence :

$$\begin{aligned} x^n &= (a^1 x^{n-1} \oplus \; ... \; \oplus a^r x^{n-r} \oplus c^{n-1}) \; mod \; 2^w, \\ c^n &= (a^1 x^{n-1} \oplus \; ... \; \oplus a^r x^{n-r} \oplus c^{n-1}) \; / \; 2^w. \end{aligned} \tag{5}$$

with output is $x^i/2^w$. The vector $(x^n, ...x^{n-r+1}, c^n)$ is the state of the generator. The array $A[0..h-1]$ contains the polynomials $a^1, ..., a^r$. Each even element stands for a polynomial number and the next element stands for the corresponding nonzero coefficient number of that polynomial. The vector $S\,[0..r-1]$ gives the initial values of $(x^0, ..., x^{r-1})$ and c is the initial carry. Restrictions : $0 < r$, and $w \leq 32$.

7.1.2.4/ GFSR

This module implements the generalized feedback shift register (GFSR) generator, that is :

$$x^n = x^{n-r} \oplus x^{n-k} \tag{6}$$

Where the x sequence are initial with some random positive integers, $n > k$ and $n > r$.

7.1.2.5/ INV

Finally, this module implements the nonlinear inversive generator, as defined in [**?**], which is :

$$x^n = \begin{cases} (a^1 + a^2/z^{n-1}) \; mod \; m & \text{if } z^{n-1} \neq 0 \\ a^1 & \text{if } z^{n-1} = 0. \end{cases} \tag{7}$$

The generator computes z via the modified Euclid algorithm (see [**?**]). If m is prime and if $p(x) = x^2 - a^1 x - a^2$ is a primitive polynomial modulo m, then the generator has maximal period m. Restrictions : $0 \leq z^0 < m, 0 < a^1 < m$ and $0 < a^2 < m$. Furthermore, m must be a prime number, preferably large.

7.2/ Results and discussion

Table 7.1 shows the results on the batteries recalled in a previous chapter, indicating that almost all the PRNGs introduced here cannot pass all their embedding tests. In other words, the statistical quality of these PRNGs cannot fulfill the up-to-date standards presented previously. We will show that the CIPRNG can solve this issue.

To illustrate the effects of this CIPRNG in detail, experiments will be divided in three parts :

1. **Single CIPRNG** : The PRNGs involved into the chaotic iterations computing are of the same category.
2. **Mixed CIPRNG** : Two different types of generators are mixed during the CIPRNG process.
3. **Multiple CIPRNG** : The generator is obtained by repeating the composition of the iteration function as follows : $x^0 \in \mathbb{B}^{\mathsf{N}}$, and $\forall n \in \mathbb{N}^*, \forall i \in [\![1; \mathsf{N}]\!]$,

$$x_i^n = \begin{cases} x_i^{n-1} & \text{if } S^n \neq i \\ \forall j \in [\![1; \mathsf{m}]\!], f^m(x^{n-1})_{S^{nm+j}} & \text{if } S^{nm+j} = i. \end{cases} \tag{8}$$

m is called the *functional power*.

We have performed statistical analysis of each of the aforementioned CIPRNGs. The results are reproduced in Tab. 7.1 and 7.2. The scores written in boldface indicate that all the tests have been passed successfully, whereas an asterisk "*" means that the considered passing rate has been improved.

7.2.1/ Tests based on the Single CIPRNG

The statistical tests results of the PRNGs using the single CIPRNG method are given in Tab. 7.2. We can observe that, except for the Xor CIPRNG, all of the CIPRNGs have passed

TABLE 7.1 – NIST and DieHARD tests suite passing rates for PRNGs without CI

Types of PRNGs	Linear PRNGs		Lagged PRNGs				ICG PRNGs		Mixed PRNGs		
Tests \ *PRNG*	LCG	MRG	AWC	SWB	SWC	GFSR	INV	LCG2	LCG3	MRG2
NIST	11/15	14/15	**15/15**	**15/15**	14/15	14/15	14/15	14/15	14/15	14/15
DieHARD	16/18	16/18	15/18	16/18	**18/18**	16/18	16/18	16/18	16/18	16/18

TABLE 7.2 – NIST and DieHARD tests suite passing rates for PRNGs with CI

Types of PRNGs	Linear PRNGs		Lagged PRNGs			ICG PRNGs	Mixed PRNGs		
Tests \ *Single CIPRNG*	LCG	MRG	AWC	SWB	SWC	GFSR	INV	LCG2	LCG3	MRG2
Version 1 CIPRNG										
NIST	**15/15** *	**15/15** *	**15/15**	**15/15**	**15/15** *	**15/15** *	**15/15** *	**15/15** *	**15/15** *	**15/15**
DieHARD	**18/18** *	**18/18** *	**18/18** *	**18/18** *	**18/18**	**18/18** *	**18/18** *	**18/18** *	**18/18** *	**18/18** *
Version 2 CIPRNG										
NIST	**15/15** *	**15/15** *	**15/15**	**15/15**	**15/15** *	**15/15** *	**15/15** *	**15/15** *	**15/15** *	**15/15**
DieHARD	**18/18** *	**18/18** *	**18/18** *	**18/18** *	**18/18**	**18/18** *	**18/18** *	**18/18** *	**18/18** *	**18/18** *
Xor CIPRNG										
NIST	14/15*	**15/15** *	**15/15**	**15/15**	14/15	**15/15** *	14/15	**15/15** *	**15/15** *	**15/15**
DieHARD	16/18	16/18	17/18*	**18/18** *	**18/18**	**18/18** *	16/18	16/18	16/18	16/18

the 15 tests of the NIST battery and the 18 tests of the DieHARD one. Moreover, considering these scores, we can deduce that both the single Version 1 CIPRNG and the single Version 2 CIPRNG are relatively steadier than the single Xor CIPRNG approach, when applying them to different PRNGs. However, the Xor CIPRNG is obviously the fastest approach to generate a CI random sequence, and it still improves the statistical properties relative to each generator taken alone, although the test values are not as good as desired.

Therefore, all of these three ways are interesting, for different reasons, in the production of pseudorandom numbers and, on the whole, the single CIPRNG method can be considered to adapt to or improve all kinds of PRNGs.

To have a realization of the Xor CIPRNG that can pass all the tests embedded into the NIST battery, the Xor CIPRNG with multiple functional powers are investigated in Section 7.2.3.

7.2.2/ Tests based on the Mixed CIPRNG

To compare the previous approach with the CIPRNG design that uses a Mixed CIPRNG, we have taken into account the same inputted generators than in the previous section. These inputted couples $(PRNG_1, PRNG_2)$ of PRNGs are used in the Mixed approach as follows :

$$\begin{cases} x^0 \in [\![0, 2^{\mathsf{N}} - 1]\!], S \in [\![0, 2^{\mathsf{N}} - 1]\!]^{\mathbb{N}} \\ \forall n \in \mathbb{N}^*, x^n = x^{n-1} \oplus PRNG_1 \oplus PRNG_2, \end{cases} \tag{9}$$

With this Mixed CIPRNG approach, both the Version 1 CIPRNG and Version 2 CIPRNG continue to pass all the NIST and DieHARD suites. In addition, we can see that the PRNGs using a Xor CIPRNG approach can pass more tests than previously. The main reason of this success is that the Mixed Xor CIPRNG has a longer period. Indeed, let n_P be the period of a PRNG P, then the period deduced from the single Xor CIPRNG approach is obviously equal to :

$$n_{SXORCI} = \begin{cases} n_P & \text{if } x^0 = x^{n_P} \\ 2n_P & \text{if } x^0 \neq x^{n_P}. \end{cases} \tag{10}$$

Let us now denote by n_{P1} and n_{P2} the periods of respectively the $PRNG_1$ and $PRNG_2$ generators, then the period of the Mixed Xor CIPRNG will be :

$$n_{XXORCI} = \begin{cases} LCM(n_{P1}, n_{P2}) & \text{if } x^0 = x^{LCM(n_{P1}, n_{P2})} \\ 2LCM(n_{P1}, n_{P2}) & \text{if } x^0 \neq x^{LCM(n_{P1}, n_{P2})}. \end{cases} \tag{11}$$

In Tab. 7.3, we only show the results for the Mixed CIPRNGs that cannot pass all DieHARD suites (the NIST tests are all passed). It demonstrates that Mixed Xor CIPRNG involving LCG, MRG, LCG2, LCG3, MRG2, or INV cannot pass the two following tests, namely the "Matrix Rank 32x32" and the "COUNT-THE-1's" tests contained into the DieHARD battery. Let us recall their definitions :

– **Matrix Rank 32x32.** A random 32x32 binary matrix is formed, each row having a 32-bit

TABLE 7.3 – Scores of mixed Xor CIPRNGs when considering the DieHARD battery

$PRNG_1$ \ $PRNG_0$	LCG	MRG	INV	LCG2	LCG3	MRG2
LCG		16/18	16/18	16/18	16/18	16/18
MRG	16/18		16/18	16/18	16/18	16/18
INV	16/18	16/18		16/18	16/18	16/18
LCG2	16/18	16/18	16/18		16/18	16/18
LCG3	16/18	16/18	16/18	16/18		16/18
MRG2	16/18	16/18	16/18	16/18	16/18	

random vector. Its rank is an integer that ranges from 0 to 32. Ranks less than 29 must be rare, and their occurences must be pooled with those of rank 29. To achieve the test, ranks of 40,000 such random matrices are obtained, and a chisquare test is performed on counts for ranks 32,31,30 and for ranks ≤ 29.

– **COUNT-THE-1's TEST** Consider the file under test as a stream of bytes (four per 2 bit integer). Each byte can contain from 0 to 8 1's, with probabilities 1,8,28,56,70,56,28,8,1 over 256. Now let the stream of bytes provide a string of overlapping 5-letter words, each "letter" taking values A,B,C,D,E. The letters are determined by the number of 1's in a byte : 0,1, or 2 yield A, 3 yields B, 4 yields C, 5 yields D and 6,7, or 8 yield E. Thus we have a monkey at a typewriter hitting five keys with various probabilities (37,56,70,56,37 over 256). There are 5^5 possible 5-letter words, and from a string of 256,000 (overlapping) 5-letter words, counts are made on the frequencies for each word. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a chisquare test : Q5-Q4, the difference of the naive Pearson sums of $(OBS - EXP)^2/EXP$ on counts for 5- and 4-letter cell counts.

The reason of these fails is that the output of LCG, LCG2, LCG3, MRG, and MRG2 under the experiments are in 31-bit. Compare with the Single CIPRNG, using different PRNGs to build CIPRNG seems more efficient in improving random number quality (mixed Xor CI can 100% pass NIST, but single cannot).

7.2.3/ TESTS BASED ON THE MULTIPLE CIPRNG

Until now, the combination of at most two input PRNGs has been investigated. We now regard the possibility to use a larger number of generators to improve the statistics of the generated pseudorandom numbers, leading to the multiple functional power approach. For

Table 7.4 – Functional power m making it possible to pass the whole NIST battery

Inputted *PRNG*	LCG	MRG	SWC	GFSR	INV	LCG2	LCG3	MRG2
Threshold value m	19	7	2	1	11	9	3	4

the CIPRNGs which have already pass both the NIST and DieHARD suites with 2 inputted PRNGs (all the Old and Version 2 CIPRNGs, and some of the Xor CIPRNGs), it is not meaningful to consider their adaption of this multiple CIPRNG method, hence only the Multiple Xor CIPRNGs, having the following form, will be investigated.

$$\begin{cases} x^0 \in [\![0, 2^{\mathsf{N}} - 1]\!], S \in [\![0, 2^{\mathsf{N}} - 1]\!]^{\mathbb{N}} \\ \forall n \in \mathbb{N}^*, x^n = x^{n-1} \oplus S^{nm} \oplus S^{nm+1} \ldots \oplus S^{nm+m-1}, \end{cases} \tag{12}$$

The question is now to determine the value of the threshold m (the functional power) making the multiple CIPRNG being able to pass the whole NIST battery. Such a question is answered in Tab. 7.4.

7.2.4/ Results Summary

We can summarize the obtained results as follows.

1. The CIPRNG method is able to improve the statistical properties of a large variety of PRNGs.
2. Using different PRNGs in the CIPRNG approach is better than considering several instances of one unique PRNG.
3. The statistical quality of the outputs increases with the functional power m.

In this chapter, we first have formalized the CI methods that has been already presented in previous research articles. These CI methods are based on iterations that have been topologically proven as chaotic. Then 10 usual PRNGs covering all kinds of generators have been applied, and the NIST and DieHARD batteries have been tested. Analyses show that PRNGs using the CIPRNG methods do not only inherit the chaotic properties of the CI iterations, they also have improvements of their statistics. This is why CIPRNG techniques should be considered as post-treatments on pseudorandom number generators to improve both their randomness and security.

8

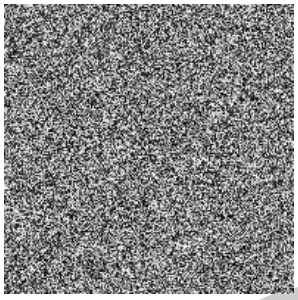
Illustrative Example of use of CIPRNGs in the Computer Science Security Field

This chapter is devoted to a first application of CIPRNGs to see whether they still remains good in concrete situations. The publication of [**?**] has formally given a description of this research.

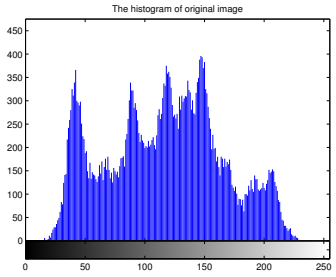
8.1/ Introduction

The confidentiality of information transmitted through the Internet requires an intensive use of pseudorandom number generators having strong security properties. For instance, these generators are used to produce encryption keys, to encrypt data with a one-time pad process, or to dissimulate information into cover media. In our previous chapters, we have shown how to use discrete chaotic iterations to build pseudorandom number generators, by receiving two inputted possibly deficient generators, and mixing them to produce pseudorandom numbers with high statistical qualities. In this chapter, we propose simple basic applications of these generators for encryption and information hiding. For each application, first experimental evaluations are given, showing that an attacker using these statistics as detection tools cannot infer the presence of an hidden message into given cover documents.

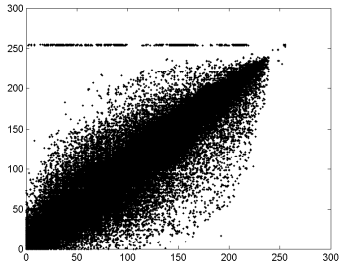
Random binary sequences will be generated by the three first CIPRNG methods using a XORshift generator. The application of these pseudorandom bits for information hiding will be carried out systematically, and results will be discussed in order to verify that an attacker, who has only access to some elementary statistical tests, cannot determine whether hidden information are embedded into cover documents or not. Notice that the aim of this chapter is not to propose an up-to-date, finalized, and secure data hiding scheme, but only to show the usability and effectiveness of our CIPRNGs.



(a)

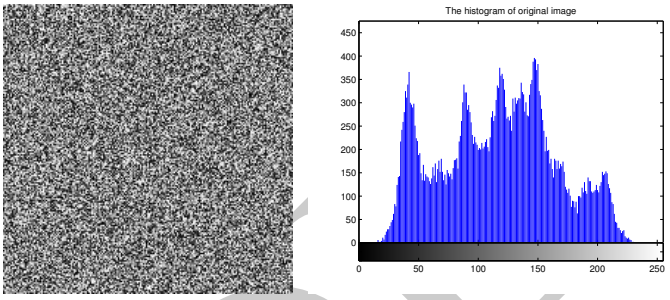


(b)

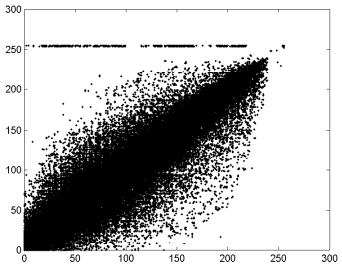


(c)

FIGURE 8.1 – (a) The encrypted Lena (one-time pad using Version 1 CI). (b) Histogram of Fig.(a). (c) Correlation distribution of two adjacent pixels in Fig.(a)

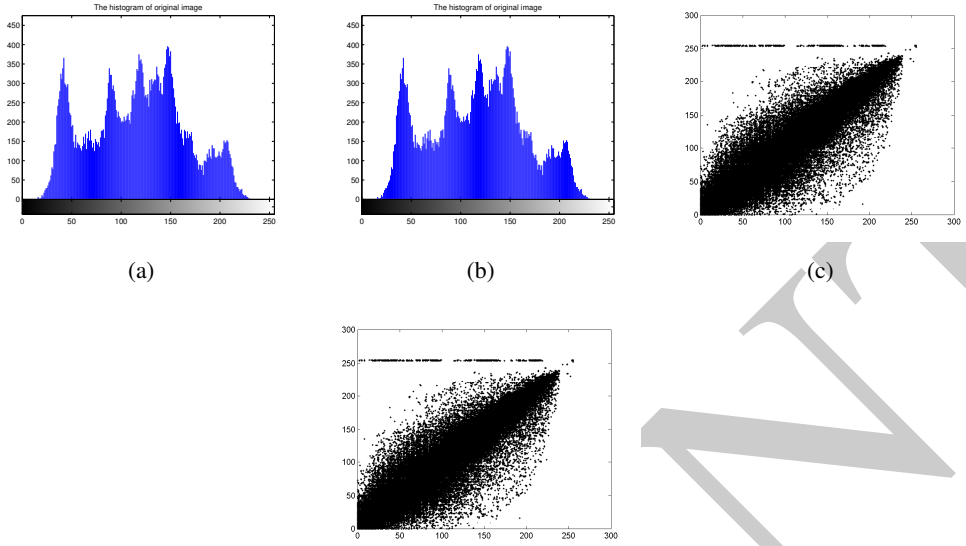


(a) (b)



(c)

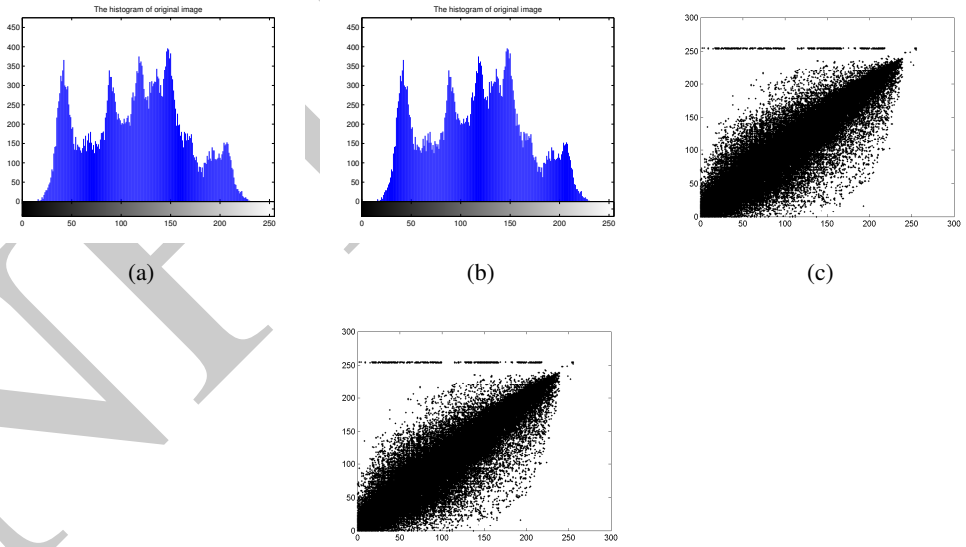
FIGURE 8.2 – (a) The encrypted Lena (one-time pad using Version 2 CI). (b) Histogram of Fig.(a). (c) Correlation distribution of two adjacent pixels in Fig.(a)



(a) (b) (c)

(d)

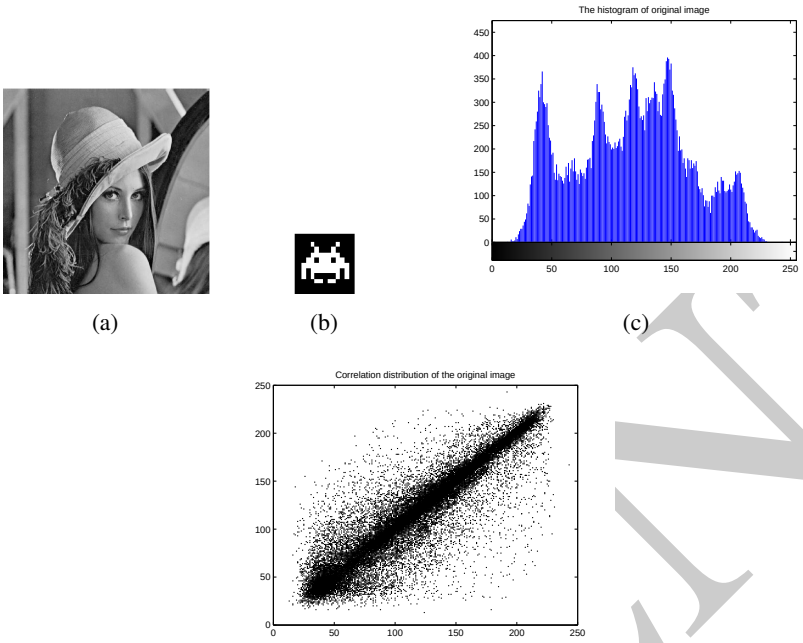
FIGURE 8.3 – (a) Histogram of pixel values when LSBs are replaced by Version 1 CI. (b) Histogram of pixel values when LBSs are an hidden message xored with Version 1 CI. (c) Correlation distribution of two adjacent pixels in Fig.(a). (d) Correlation distribution of two adjacent pixels in Fig.(b).



(a) (b) (c)

(d)

FIGURE 8.4 – (a) Histogram of pixel values when LSBs are replaced by Version 2 CI. (b) Histogram of pixel values when LBSs are an hidden message xored with Version 2 CI. (c) Correlation distribution of two adjacent pixels in Fig.(a). (d) Correlation distribution of two adjacent pixels in Fig.(b).



(a) (b) (c)

(d)

FIGURE 8.5 – (a) The original image. (b) The hidden image. (c) Correlation distribution of the original image. (d) Histogram of the original image.

8.2/ APPLICATION EVALUATION

Let us firstly introduce our toy example in the information hiding security field.

8.2.1/ THE PROPOSED INFORMATION HIDING METHOD

Suppose that the size of the image is $M \times N$. The steps of the proposed information hiding algorithm using the CIPRNG family (versions 1-3) are summed up below. Researches presented in this chapter have been formerly submitted/accepted/published in [**?**].

1. Generate a pseudorandom sequence S of length $M \times N$ using each CIPRNG method respectively.
2. Transform the image into a $M \times N$ integer sequence.
3. The LSBs (Least Significant Bits) of the image integer sequence are replaced by the generated random bits S. These random LSBs will be treated as a keystream.
4. The information (text or picture) to hide is transformed into a binary sequence.
5. The binary message is hiding into the random LSBs of the image sequence, by using the bitwise exclusive or operation between the two sequences, starting from a selected position acting as part of the secret key.

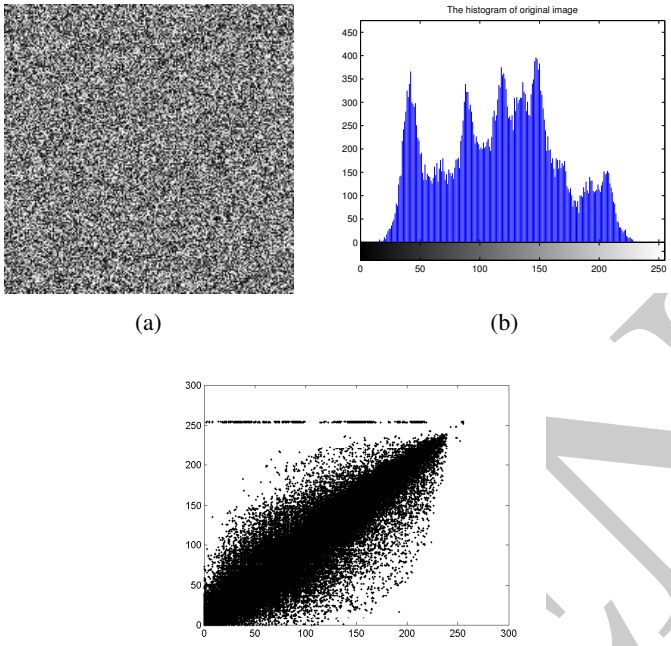


FIGURE 8.6 – (a) The encrypted Lena (one-time pad using Version 3 CI). (b) Histogram of Fig.(a). (c) Correlation distribution of two adjacent pixels in Fig.(a)

As mentioned previously, pseudorandom sequences generated by the three CI methods, with two XORshift generators and a given image, are used in this application to process to an evaluation of the scheme.

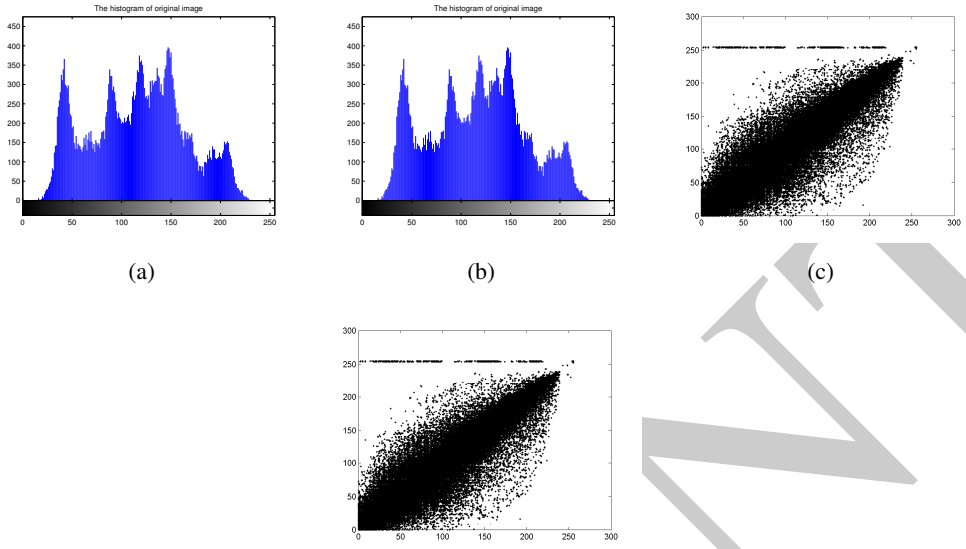
8.2.2/ First Experimental Evaluation of the Proposed Scheme

8.2.2.1/ The context

The original image of size 713×713, probably the most widely used test image for all kind of processing algorithms (such as compression and encryption), is depicted in Fig. 8.5-a. Fig. 8.5-c presents its histogram, and Fig. 8.5-d shows the correlation distribution of two horizontal adjacent pixels in this original image. Finally, information that must be hidden into it is the picture of Fig. 8.5-b, which has 89×89 pixels.

8.2.2.2/ Histogram and Horizontal Correlation

Two XORshift generators are used to generate a random sequence based on the Version 1 CI method. Results are shown in Fig. 8.3. Histograms and correlation distributions (Fig. 8.3-a,b,c,d) are very closed to each other, leading to the assumption that such a method can well



(a) (b) (c)

(d)

FIGURE 8.7 – (a) Histogram of pixel values when LSBs are replaced by Version 3 CI. (b) Histogram of pixel values when LBSs are a hidden message xored with Version 3 CI. (c) Correlation distribution of two adjacent pixels in Fig.(a). (d) Correlation distribution of two adjacent pixels in Fig.(b).

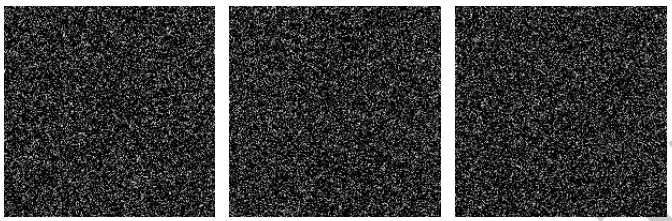
protect the hidden information when facing statistical attacks. The same experimental validation has been applied to the Version 2 CI method using two XORshift generators. Such experiments lead to results that are shown in Fig. 8.4. These first results are encouraging and confirm that simple histogram and correlation evaluations cannot detect the presence of hidden messages. The same conclusion can be claimed when using the Version 3 CI generator, as it is depicted in Fig. 8.7.

8.2.2.3/ ALL DIRECTIONS CORRELATION COEFFICIENTS ANALYSIS

Using an identical experimental evaluation than in [**?**], the correlation coefficients of the horizontal, vertical, and diagonal directions of all the concerned images (original, with random as LSBs, and with secret information in these LSBs) are shown in Table 8.1. It can be experimentally deduced that the correlation properties of these images are very similar to each other. So an attacker, whose intention is to analyze these coefficients in order to detect possible information hiding, cannot attain his/her goal by such a simple experiment.

8.2.2.4/ INITIAL CONDITION SENSITIVITY

One of the most important properties of the chaotic sequences is that they are very sensitive to their initial conditions. This property can help to face an attacker who has access to



(a) (b) (c)

FIGURE 8.8 – (a) The difference of two random LSBs image using Version 1 CI PRNG with slight change in initial condition. (b) The difference of two random LSBs image using Version 2 CI PRNG with slight change in initial condition (c) The difference of two random LSBs image using Version 3 CI PRNG with slight change in initial condition

TABLE 8.1 – Correlation coefficients of two adjacent pixels in all directions in the original image, random LSBs images and infomation intergraded random LSBs images

Direction / **Image**	Horizontal	Vertical	Diagonal
Original image	0.9793	0.9686	0.9488
Version 1 CI			
no info	0.9792	0.9686	0.9488
intergrading info	0.9792	0.9686	0.9488
Version 2 CI			
no info	0.9793	0.9686	0.9488
intergrading info	0.9793	0.9686	0.9488
Version 3 CI			
no info	0.9793	0.9686	0.9487
intergrading info	0.9793	0.9686	0.9487

the whole algorithm and to an approximation of the secret key. His/her intention, in this attack scenario, is to find the exact secret key (the seed of the keystream and the position of the message), by making small changes on this key. If the keystream and the position do not change a lot when the key is slightly updated, then the attacker can converge by small changes to the used secret key. In the experiments of Figure 8.8, we slightly alter the keys and try to extract the hidden information from the image. We can conclude that such optimistic attempts always fail in recovering the message.

8.2.3/ A small evaluation of Encryption

The dissimulation has been obtained in this paper by using the CIPRNGs recalled previously as stream cyphers : encryption is the result of the use of the bitwise exclusive or (XOR) between the given message and pseudorandom sequences generated from various CIPRNGs. We can wonder whether an attacker, who has access to the histogram of LSBs, can infer what kind of CIPRNG has been used as keystream. For obvious reasons, these histograms should at least be uniform for each PRNG.

For illustration purpose, Lena has been encrypted by such method using each of the three kind of CIPRNGs, and histogram and correlation distribution of the encrypted image have been computed. The resulting images are depicted in Fig. 8.1 when using the Version 1 CI method, in Fig. 8.2 for the Version 2 CI one, and in Fig. 8.6 for the last PRNG recalled here. We can show that this first reasonable requirement seems to be respected, even if this illustration is not a proof.

8.3/ Conclusion

We have proposed, in this short application chapter, simple illustrative examples of use for information hiding. The three first versions of the CIPRNGs family have been used here. For each generator, firsts experimental evaluations of a simple information hiding scheme have been realized, to illustrate that that an attacker using simple statistics cannot determine easily, only by regarding the form of histograms or correlation distributions, the presence of an hidden message into a given document. No evidence of dissimulation appears at first glance, when comparing histograms, correlation distribution, or all directions' correlation coefficients. Furthermore, experiments have illustrated high sensitivity to the secret parameters. These simple evaluations do not imply the security of the proposed scheme, they only illustrate that the use of the CIPRNGs for information hiding can be further investigated by more stringent tools as steganalyzers and mathematical proofs.

9 FPGA Acceleration of CIPRNGs

As well-designed information security applications frequently use a very large quantity of good pseudorandom numbers, inefficient generation of these numbers can be a significant bottleneck in various situations [**?**, **?**, **?**, **?**]. In that context, re-configurable hardware like field programmable gate arrays (FPGAs) have for many years been identified as a suitable technology having the potential to improve performance compared to traditional microprocessor based approaches.

In this chapter, our generators based on chaotic iterations are redesigned specifically for FPGA hardware, leading to an obvious improvement of the generation rate of such numbers. Analyses illustrate that statistically perfect random sequences are produced. The research has been submitted in [**?**, **?**] before.

9.1/ INTRODUCTION

PRNGs are very important primitives widely used in numerous applications like numerical simulations or security. Depending on the targeted application, these PRNGs must achieve requirements as speed, statistical quality, security, and so on. On the one hand, field programmable gate arrays (FPGAs) have been successfully used for realizing the speed requirement in pseudorandom sequence generation, due to their high parallelization capability [**?**, **?**, **?**]. Advantages of such physical generation way encompass performance, design time, power consumption, flexibility, and cost.

It has been stated in the previous chapters that chaotic iterations are good candidates to generate sequences both secure and random, due among other things to their sensitivity to initial conditions and their broadband spectrum. Our intention in this chapter, which continues the studies initiated in [**?**], is to merge these two approaches by proposing a discrete chaos-based generator designed on FPGA.

9.2/ CIPRNG design on FPGA

9.2.1/ Selection of the CIPRNG version

According to the comparison produced in Chapter 6, it can be seen that CIPRNG version 4 is the most adaptable of all the generators into this chaotic iterations based family. The loop processing that he embeds can be replaced by parallel computing to increase efficiency. Its statistical performance are good enough to pass with success the NIST, DieHARD, and TestU01 test suites (see Section 6.3).

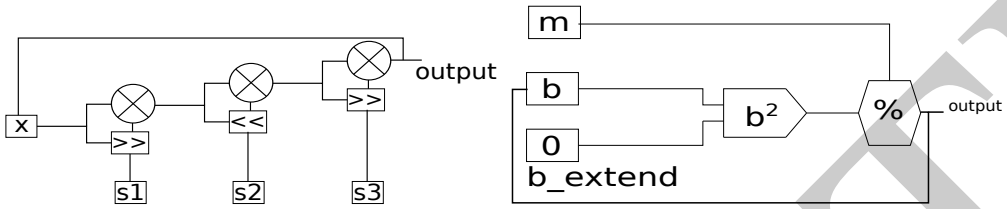
In order to take benefits from the computing power of FPGA, a whole processing needs to spread the various components of the generator into several independent blocks of threads that can be computed simultaneously. In general, the larger the number of threads is, the more logistic elements of FPGA are used, and the less branching instructions are used (if, while, ...), the better the performances on FPGA are. Obviously, having these requirements in mind, it is possible to build a program similar to the algorithm presented in Tab. 5.4, which produces pseudorandom numbers with chaotic properties on FPGA. To do so, Verilog-HDL [**?**] has been used to help programing. In this generator, there are three PRNG objects that use the exclusive or operation, two XORshifts, and a BBS, their processing are described thereafter.

9.2.2/ Design of XORshift

The structure of XORshift designed in Verilog-HDL is shown in Fig.9.1(a). There are four inputs :
– The first one is the initial state, which costs 64 bits of register units,
– the other three ones are used to define the shift operations.
Let us remark that, in FPGA, this shift operation costs nothing, as it simply consists in using different bit cells of the input. We can thus conclude that there are $64-s1+64-s2+64-s3 = 192 - s1 - s2 - s3$ logic gates elements that are required for the XORshifts processing.

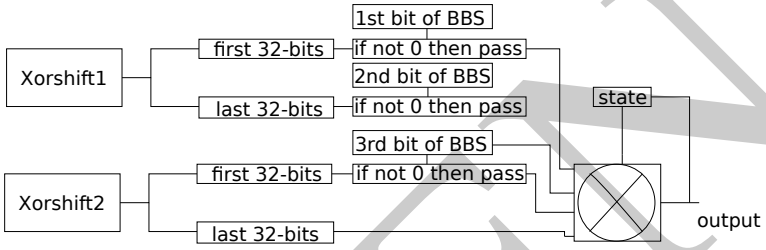
9.2.3/ Design of BBS

Fig.9.1(b) gives the proposed design of the BBS generator in FPGAs. There are two inputs of 32 bits, namely b and m. Register b stores the state of the system at each time (after the square computation). m is also a register that saves the value of M, which must not change. Another register *b_extend* is used to combine b to a data having 64 bits, with a view to avoid overflow. After the last computation, the three LSBs from the output of % are taken as output. Let us notice that a BBS is performed at each time unit.



(a) XORshift

(b) BBS



(c) The proposed CIPRNG

FIGURE 9.1 – The processing structure for BBS in FPGA (per clock step)

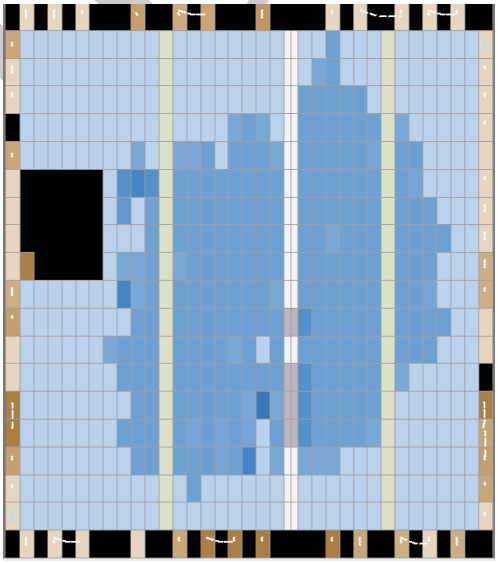


FIGURE 9.2 – The sources cost in $EP2C8Q208C8$ FPGA board

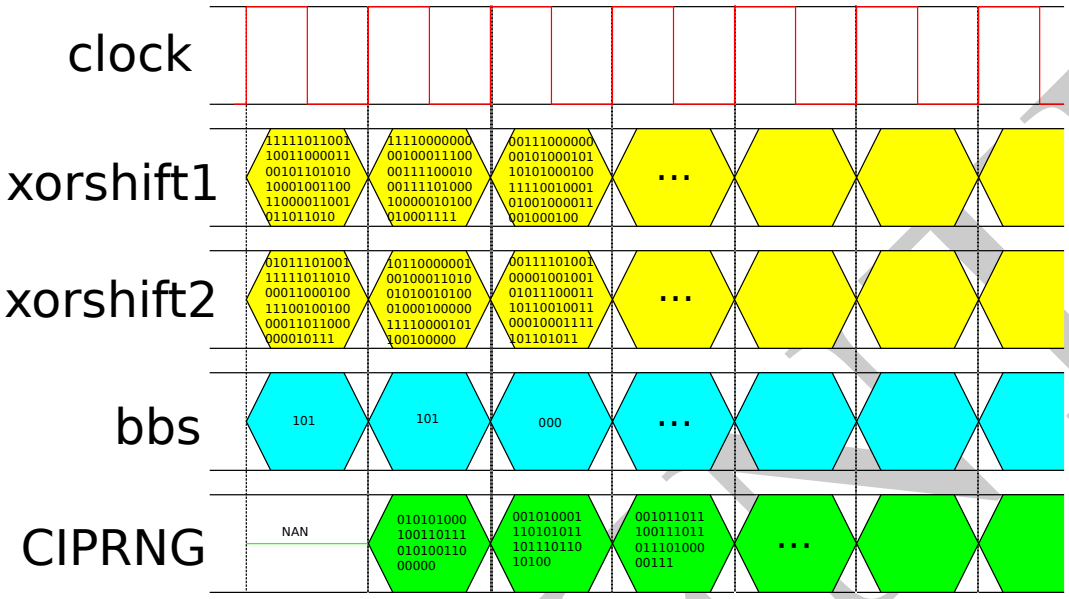


FIGURE 9.3 – Working flow of each component for CIPRNG in FPGA

9.2.4/ DESIGN OF THE CHAOTIC ITERATIONS

Two XORshifts and one BBS are connected to work together, in order to compose the proposed CIPRNG (see Fig.9.1(c)). As it can be shown, the three bits of the BBS output are switches for the corresponding 32 bits XORshift outputs. Every round of the processing costs two time units to be performed : in the first clock, the four PRNGs are processed in parallel, whereas in the second one, the results of these generators are combined with the current state of the system, in order to produce the output of 32 bits. The output sequence will be appended as Fig. 9.3 shown, started from the second clock (first clock BBS and XORshift use to initial the first output of CIPRNG).

In our experiments, the type $EP2C8Q208C8$ from Altera company's CYCLONE II FPGA series has been used. By default, its working frequency is equal to 50 MHz. However, it is possible to increase it until 400 MHz by using the phase-lock loop (PLL) device. In that situation, the CIPRNG designed on this FPGA can produce about 6400 Mbits per second (that is, $400(MHz) \div 2(times) \times 32(bits)$), while using 3358 of the 8256 logic elements in $EP2C8Q208C8$ (see Fig.9.2).

In the next section, an application of this CSPRNG designed on FPGA in the information hiding security fields is detailed, to show that this hardware pseudorandom generator is ready to use.

10

Software and Hardware Implementation of a Watermarking Scheme using CIPRNG

10.1/ Introduction

As stated in a previous chapter, information hiding has recently become a major information security technology, especially with the increasing importance and widespread distribution of digital media through the Internet [**?**]. 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 section, a watermarking algorithm based on the chaotic PRNGs presented in this part is given, as an illustration of use of these PRNG based on CIs [**?**, **?**].

10.2/ Definition of our Chaos-Based Information Hiding Scheme

Let us now introduce a more complex information hiding scheme based on chaotic iterations. It has been formerly introduced in [**?**, **?**], in which more complete explanations on details of the proposed scheme can be found.

10.2.1/ 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.

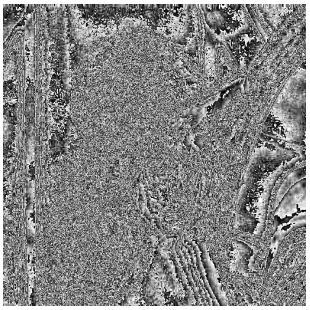
For example, in a spatial description of a grayscale image, a definition of MSCs can be the sequence constituted by the first four bits of each pixel (see Figure 10.1). In a discrete cosine frequency domain description, each 8×8 block of the carrier image is mapped onto a list of 64 coefficients. The 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 10.1). 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.

(a) Lena.

(b) MSCs of Lena.



(c) LSCs of Lena (×17).

Figure 10.1 – Example of most and least significant coefficients of Lena.

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.

10.2.2/ Stages of the scheme

Our CIPRNG version 4 generator-based information hiding scheme consists of two stages : (1) mixture of the watermark and (2) its embedding.

10.2.2.1/ 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 CIPRNG version 4. 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.

10.2.2.2/ 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 which are highly dependent on the MSCs [**?**].

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,

$$f_0 : (x_1, ..., x_{\mathsf{M}}) \in \mathbb{B}^{\mathsf{M}} \longmapsto (\overline{x_1}, ..., \overline{x_{\mathsf{M}}}) \in \mathbb{B}^{\mathsf{M}}, \tag{1}$$

the initial state is L, and the strategy is equal to U. In this case, the whole embedding stage satisfies the topological chaos properties [**?**], 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 CIPRNG version 4, but the whole process does not satisfy topological chaos [**?**]. The use of chaotic iterations is reduced to the mixture of the watermark. See the following sections for more detail.

10.2.2.3/ 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 10.2.

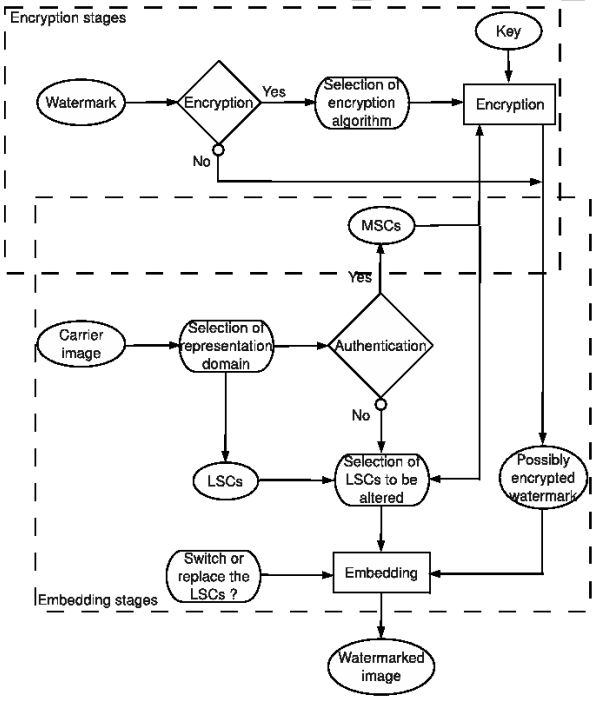
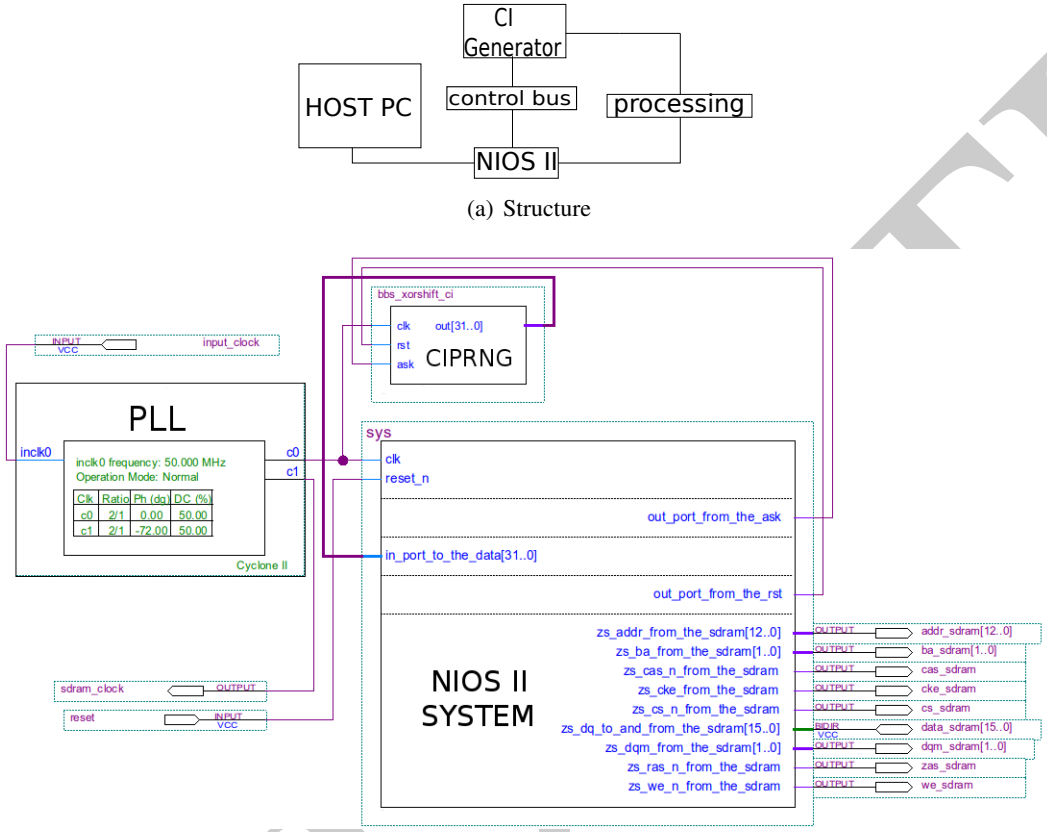


Figure 10.2 – The chaos-based data hiding decision tree.

10.3/ 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 CIPRNG version 4 (BBS, XORshift). The carrier image is the well-known Lena, which is a 256 grayscale



(a) Structure

(b) Schematic

FIGURE 10.3 – NIOS II setting in FPGA

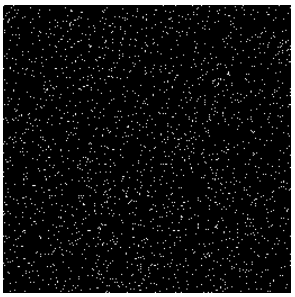
(a) The original image

(b) The watermark

FIGURE 10.4 – Original images

image, and the watermark is the 64×64 pixels binary image depicted in Figure 10.6.

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 CIPRNG version 4 (BBS, XORshift), where



(a) Differences with the original

(b) Encrypted watermark

FIGURE 10.5 – Encrypted watermark and differences

initial parameters constitute the secret key and $N = 64$. Thus, the encrypted watermark is the last boolean vector generated by these chaotic iterations. An example of such an encryption is given in Figure 10.5.

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 :

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

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 10.5).

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 [**?**]), 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.

10.4/ IMPLEMENTATIONS : FROM SOFTWARE TO HARDWARE

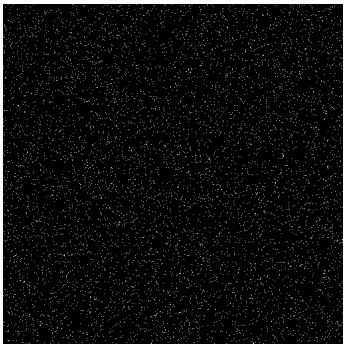
We have been in charge to implement this chaotic iterations based watermarking scheme on both software and hardware platforms, using one of the CIPRNGs we previously studied. By doing so, we will be able to study the robustness of the proposed hiding scheme, leading to the evaluation given in Section 10.5.

(a) The original image

(b) The watermark

FIGURE 10.6 – Original images

(a) Output image



(b) Difference

FIGURE 10.7 – The output of our watermarking application

10.4.1/ SOFTWARE IMPLEMENTATION : EXPERIMENTAL PROTOCOL

In this section, we details our software implementation of the information hiding application based on chaotic iterations presented in the previous sections : a watermark is encrypted and embedded into a cover image using the proposed scheme and some CIPRNGs. More precisely, the CIPRNG(BBS, XORshift) version 4 has been chosen, since it is with good statistical performance and high efficiency. The carrier image is the well-known Lena already presented, which is here a 256×256 grayscale image, and the watermark is the 64×64 pixels binary image depicted in Fig.10.6.

The application has been computed using the JAVA language [**?**]. 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 CIPRNG(BBS, XORshift) version 4, whose initial parameters constitute the secret key and $N = 64$. A simple windows interface has been implemented, as it is depicted in Fig.10.8-a. The embedding file must firstly be chosen, as shown in

Fig.10.8-b. Then the host image that will be used to carry the watermark must be selected (see Fig.10.8-c). Lastly, after processing, the output image is produced, and the differences between original image and encrypted image is produced (Fig.10.7).

10.4.2/ Hardware Implementation

In this section, to show the effectiveness and lightweight of our approach using chaotic iterations for both pseudorandom generations and information hiding, we will shortly present our hardware implementation of the whole algorithm on a FPGA. The 32-bit embedded-processor architecture designed specifically for the Altera family of FPGAs has been used for this application. Indeed, Nios II incorporates many enhancements over the original Nios architecture, making it more suitable for a wider range of embedded computing applications, from DSP to system-control, and to this watermarking scheme [**?**].

Fig.10.3(a) shows the structure of this application, the NIOS II system can read the image from the HOST computer side. Then the control bus allows the CI generator to operate, in order to produce pseudorandom bits. Finally the processed results are transmitted back into the host. In Fig.10.3(b), the NIOS II uses the most powerful version the CYCLONE II can support (that is, NIOS II/f). Then, 4 KB on chip memory and 16 MB SDRAM are set, and the *PLL* device is used to enhance the clock frequency from 50 to 200 MHz. The data bus connects NIOS II system, and the generator is in 32 bits.

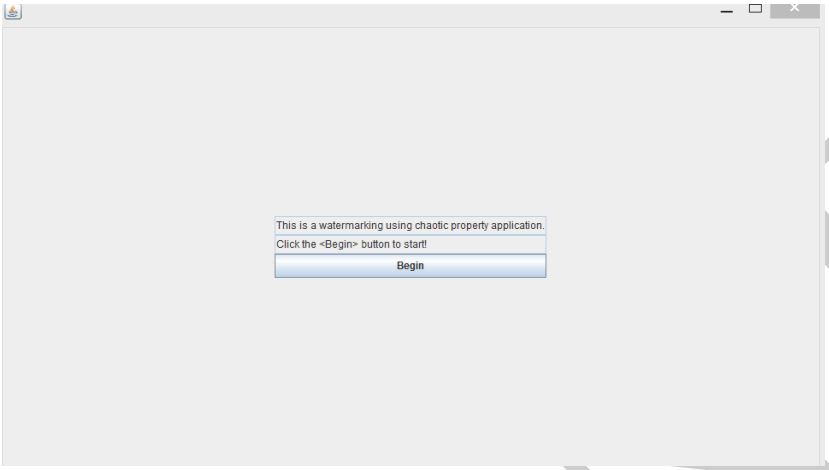
10.5/ Robustness evaluation

In what follows, the embedding domain is the spatial domain, CIPRNG(BBS,XORshift) version 4 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.

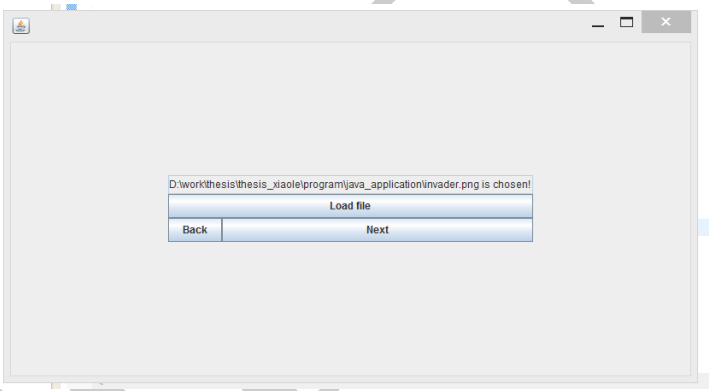
To prove the efficiency and the robustness of the proposed algorithm, some attacks are applied to our chaotic watermarked image. For each attack, a similarity percentage with the watermark is computed, this percentage is the number of equal bits between the original and the extracted watermark, shown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

10.5.0.1/ Zeroing attack

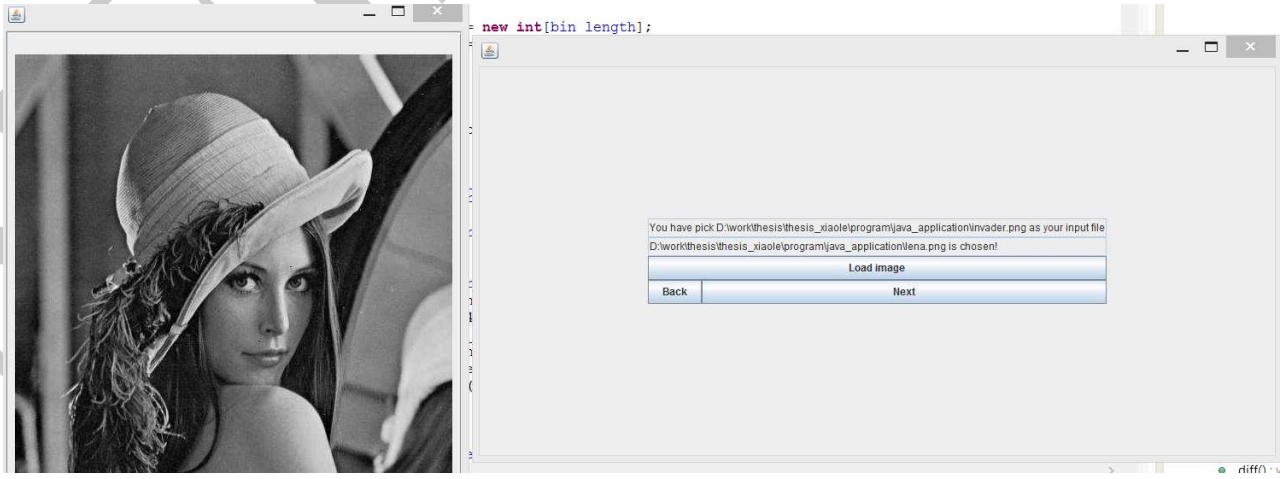
In this kind of attack, a watermarked image is zeroed, such as in Fig.10.9(a). In this case, the results in Table. 1 have been obtained. For the sake of completeness, we recall the following results taken from Qianxue's thesis [**?**].



(a) The start windows of our Java's watermarking application



(b) Choosing the contents to embed



(c) Choosing the contents to embed

FIGURE 10.8 – Java application for watermarking based on chaotic iterations

(a) Cropping attack

(b) Rotation attack

FIGURE 10.9 – Watermarked Lena after attacks.

UNAUTHENTICATION		AUTHENTICATION	
Size (pixels)	Similarity	Size (pixels)	Similarity
10	99.31%	10	92.34%
50	98.55%	50	57.11%
100	92.40%	100	54.42%
200	71.01%	200	50.93%

Table. 1. Cropping attacks

In Fig.10.10, 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).

FIGURE 10.10 – 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.

UNAUTHENTICATION		AUTHENTICATION	
Angle (degree)	Similarity	Angle (degree)	Similarity
2	97.31%	2	74.45%
5	94.02%	5	63.36%
10	89.98%	10	52.77%
25	80.84%	25	52.03%

Table. 2. Rotation attacks

10.5.0.2/ 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, which is altered as in Fig.10.9. The results in Table. 2 have been obtained.

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

10.5.0.3/ 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	85.92%	2	58.42%
5	70.45%	5	53.11%
10	64.39%	10	51.02%
20	53.94%	20	50.03%

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.

10.5.0.4/ 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	81.00%	1	56.50%
2	77.18%	2	51.92%
3	67.01%	3	52.82%
5	59.28%	5	51.76%

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.

III

Random Number Generators Based On Optoelectronic Chaotic Laser

11

Introduction

Physical RNGs rely on chaotic or stochastic physical processes. Such random number generators are building the random bits from inherently random or chaotic physical process [**?**], for example, radioactive decay [**?**], chaotic electrical and optical circuits [**?**], and so on. The implementations of physical random generators have been limited to much slower rates than PRNGs because of limitation of the mechanisms for extracting bits from physical randomness without degrading statistical properties. Typically 10 Mb/s could be achieved by using electronic oscillator jitter [**?**] and 4 Mb/s using quantum optical noise [**?**].

Considerable improvements for the rate of chaotic random bits generation have been reached by using a semiconductor laser in the presence of external feedback [**?**], a well known setup in chaotic optical systems. The dynamical processes involved in optical systems can indeed be very fast. Moreover, high complexity chaotic dynamics can be practically obtained, whether due to intrinsic complex nonlinear coupling between light and matter interactions in lasers, or due to the presence of a large delay feedback cavity enabling dynamics with large number of degrees of freedom.

Chaotic optical signal might consist of pulses with a width of few 10ps and with random amplitude and time positions, which provide attractive potentials to easily generate random bits at fast rates. In [**?**], a first attempt already reached 1.7 Gb/s RNG, the physical randomness originating from two independent chaotic semiconduct lasers. Each laser intensity signal is practically sampled at an incommensurate rate with respect to the individual optical feedback delay times ; then a threshold value is set for comparison with each signal level and to obtain a Boolean sequence ; lastly the random bit sequence is produced by executing a XOR function between the two Boolean sequences. More recently, Reidler and colleagues [**?**, **?**] claim that they successfully demonstrated another method in generating random bit sequence from ultra fast optical chaos, at much faster rate. In such method, the output of a single chaotic laser, with the optical feedback delay time incommensurate with the sampling clock frequency, was digitized by an 8-bit analog-to-digital converter (ADC, practically provided by an ultra-fast digital scope). Then the difference between adjacent, but not nearest, points from the 8-bit digitized time series is performed (it is defined as a pseudo-“derivative” operation). At last, a few LSBs only of the subtracted samples values are retained to generate the binary sequence. Following that kind of procedure, generation

rate as high as 300 Gb/s are claimed.

In this part, the study of using optical signal to generate random binary sequence according to the method proposed by Reidler, is going to be deepen. We propose to apply the same method on the chaotic waveform generated by another class of broadband photonic oscillations, and to analyze the different post-processing steps involved in this method. We will analyze three key factors in the scheme of [**?**] and [**?**] : the sampling, the difference of distant samples, and LSBs retaining.

12

Noise And Chaos Contributions in Fast Random Bit Sequences Generated From Broadband Optoelectronic Entropy Sources

We reproduce and adapt, in this chapter, the content of the publication [**?**] submitted to 'IEEE Journal of Quantum Electronics'. In this work, on which I was greatly aided by my supervisor in the OPTO team, we take a critical look on an article claiming surprising generation bit rates using a chaotic optoelectronic device.

12.1/ Method for Random Bit Sequence Generation from an Optoelectronic Signal

In this section we describe the physical setup from which we expect to obtain a fast random bit sequence. We also describe the binary sequence extraction method from the continuous time signal generated by the physical setup, as it was formerly proposed in [**?**, **?**]. Additionally, theoretical interpretation and discussion of this extraction method is proposed in terms of basic signal processing and sampling theory. These interpretation and discussion are intended to give insight on the possible mechanisms at the origin of the bit stream randomness quality.

12.1.1/ Setup delivering a broadband optoelectronic signal

In order to additionally support our work with experiments on the generation of optical broadband signals, data recorded from physical chaos generator as well as from optoelectronic noise sources, have been studied. These experiments are moreover different from the ones described in [**?**] and [**?**], although they are also originating from optoelectronic devices.

A twofold physical source of entropy has been used (see Fig.12.1), both having been tested

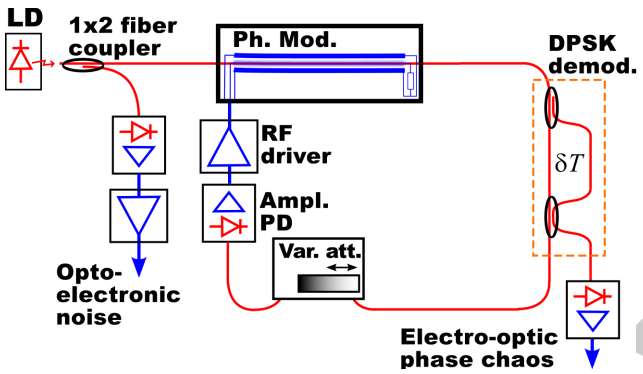


FIGURE 12.1 – Experimental setup of the optical system used to generate signal as physical random sources for the derivation of random bit sequences.

for their randomness quality. One source (referred as "Optoelectronic noise" in Fig.12.1) is originating from physical noise sources in the semiconductor laser light generation process (known as RIN : relative intensity noise), in combination with the electronic noise of the photodetector and its integrated electronic amplifier (thermal noise and semiconduction photodiode junction noise, amplified by the noise figure of the electronic amplifier). A comparable (and even cheaper) optoelectronic noise source was also proposed in [**?**].
On the contrary, the other physical source of entropy is originating from a strongly deterministic process, which was used recently for a field experiment demonstrating (analogue) chaotic optical masking of 10 Gb/s data signals, transmitted over an installed fiber optic link [**?**]. The strong determinism of this entropy source indeed enabled to implement accurate broadband chaos synchronization at the receiver, in order to remove the chaotic masking signal and thus to retrieve the original binary data stream. The dynamics of the electro-optical phase chaos generator is ruled by a nonlinear dual delay differential equation implemented in an optoelectronic and electro-optic feedback loop.

Each of these two signals obtained from noise or chaotic optoelectronic systems, has been processed by using the method proposed in [**?**]. By doing so, the aim is to support our signal processing analysis on the extraction method of the bit sequence, which is inferring that in both cases, the randomness quality must be very similar. More precisely, we claim that the actual binary sequence randomness is mainly related to the non-deterministic small background noise, and not to the deterministic large amplitude chaotic component. The deterministic chaotic signal can be approximated by the physical solution of a nonlinear dual delay dynamics ruled by the following differential equation :

$$\theta^{-1}\int_0^t x(\xi)d\xi + \tau\frac{dx}{dt}(t) + x(t) = \beta\sin^2[x_T - x_{T+\delta T} + \Phi_0], \qquad (1)$$

where x_T stands for the delayed signal $x(t-T)$, θ and τ are the characteristic times of the low and high cut–off frequency respectively, which are involved in the bandpass feedback filtering of the RF filter. From a signal processing viewpoint, such a dynamical system can

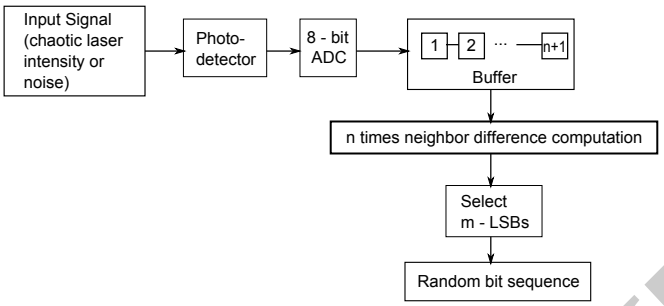


FIGURE 12.2 – Scheme of the RNG using optical signal

be interpreted as a nonlinear delayed feedback oscillator, which is ruled by the dynamics of a linear bandpass filter driven by a nonlinear transformation of two delayed (delays T and $T + \delta T$) versions of the filter output $x(t)$. The chaotic solution signal obtained when the feedback loop gain β is high enough (of the order of 5, tuned via the CW laser light intensity), is a white noise like motion covering the full spectral range of the broadband bandpass feedback RF filter, *i.e.*, ca [30 kHz−13 GHz]. This results in a fast noise-like large amplitude signal, which is expected to be suitable for high speed RNG based on a physically generated pseudorandom signal. It is worth noticing that this chaotic signal generation process can be viewed as a balanced equilibrium between the RF feedback filter aimed at limiting the spectral span of the signal $x(t)$, and the spectral broadening performed by the nonlinear transformation ($\sin^2$ –function of the difference delayed signals $x_T - x_{T+\delta T}$). The offset phase Φ_0 is typically adjusted through the average interference condition physically implemented to provide the $\sin^2$ nonlinear transformation.
A more accurate description of the generated signal $x(t)$ should also include (small amplitude) noise sources in the equation, the latter noise being actually very similar to the one at the origin of the noisy optoelectronic signal (laser and photodiode noise, also complemented here by the RF electronic amplifier noise). Equation (1) can however be confidently used solely. On the one hand, it can be used to generate numerically the obtained chaotic motion with qualitative properties that are actually very close to the ones observed in the experiment [**?**] ; on the other hand it can be used to derive analytically some of the bifurcation features both observed in the experiment [**?**] and described analytically.

12.1.2/ Extraction method for the random binary sequence

A schematic view of the algorithm used to extract a random binary sequence from a broadband physical signal, as proposed in [**?**], is depicted in Fig.12.2. On the basis of a physical setup delivering a broadband signal, as the one described in the previous section, a real time oscilloscope is firstly involved to perform an analogue to digital conversion of the output signal of the setup. This conversion is typically achieved via an 8-bit digitizer at a sampling rate of 40 GHz. In the next subsection, we will discuss from the signal theory viewpoint some particular processing issues that are suspected to significantly contribute to the actual

randomness of the final binary sequence. More precisely, sampling issues will be discussed, quantization issues, and also post-processing operations (such as distant sample difference, and LSB-only retaining). This signal processing is performed before getting the actual final random binary sequence to be tested for their randomness quality via standard statistical test suites.

12.1.2.1/ Sampling issues : aliasing for enhanced entropy

In the following, we assume that samples are originally acquired by a real time digital scope measuring a broadband complex time trace. Such equipment is designed to follow the classical Shannon sampling theorem : the sampling rate f_S is matching the instrument analogue input bandwidth, which defines the maximum Fourier frequency f_M that can be captured by the instrument. The Shannon sampling theorem indeed states that a limited bandwidth signal can be digitized without loss of information, when the sampling frequency is at least twice the maximum signal frequency. The sequence of the samples $\{s_n = x(nT),\ n \in \mathbb{Z}\}$ can be defined as a function of the continuous time as follows :

$$s(t) = x(t) \cdot \sqcup\!\sqcup_{T_S}(t), \tag{2}$$
$$\text{where } \sqcup\!\sqcup_T(t) = \sum_{k=-\infty}^{k=+\infty} \delta(t - k\,T).$$

A typical illustration of the proof for the sampling theorem is presented in Fig.12.3, as one describes the spectrum of such a sampled signal $s(t)$,

$$S(\nu) = \mathrm{FT}[s(t)] = X(\nu) \star \mathrm{FT}[\sqcup\!\sqcup_T(t)] = \frac{1}{T} X(\nu) \star \sqcup\!\sqcup_{1/T}(\nu), \tag{3}$$

where we have used the well-known result that the Fourier Transform (FT) of a comb is also a comb. The convolution product in the Fourier domain reveals that the spectrum of the sampled signal is the result of the superposition of an infinite number of regularly spaced replica of the original signal spectrum $X(\nu)$ =FT$[x(t)]$, two neighboring replica being separated by the sampling frequency $f_S = 1/T$. Thus, if the maximum frequency f_M of the bounded support of $X(\nu)$ is less than $f_S/2$, the replicated spectra do not overlap (see Fig.12.3(a)). It is then obvious that a suitable window filtering of the sampled signal $s(t)$ allows to recover in the Fourier domain exactly the same spectrum than the one of the original signal $x(t)$ (*e.g.*, a filter transmitting perfectly all the Fourier components in a frequency band such as $[-f_S/2, +f_S/2]$, and rejecting all the other Fourier components for the other frequency ranges). When undersampling is used, the *aliasing* phenomenon occurs in the Fourier domain. It consists then of overlaps between the replicated spectra due to the comb convolution. The actual spectrum of the sampled signal $s(t)$, can be viewed as a complex mixing of the frequency components of the original signal $x(t)$, due to the overlapped replica of $X(\nu)$ =FT$[x(t)]$. The procedure of selecting only one sample every n from the original sampled sequence, is thus equivalent to an aliasing operation with an

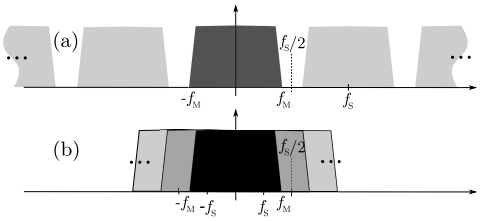


FIGURE 12.3 – Illustration of the properly fulfilled sampling theorem conditions (a), and the incorrect sampling condition leading to aliasing (b).

undersampling of order n. The original goal of cementification of the extracted sample sequence, can thus be viewed as an aliasing technique resulting in a complex mixing of the original frequency components. The consequence is an increased entropy of the output sequence, as this procedure, when viewed in the time domain, results in the vanishing of the short time correlations. On the contrary, these short time scales correlations are necessarily present when the conditions of the sampling theorem are fulfilled. Another consequence is that such an operation is unidirectional, in the sense that original information is actually lost after an aliasing process. Because of the complex mixing of the Fourier frequency components, the original spectrum cannot be recovered with a "simple" unmixing.

12.1.2.2/ FURTHER POST-PROCESSING : DIFFERENCE SEQUENCE BETWEEN DISTANT SAMPLES

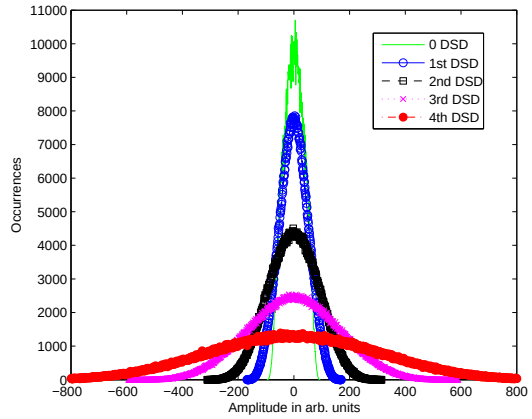
In [**?**] and [**?**], computing the difference sequence between two neighbor samples are named as "derivative". However, mathematically speaking, the term "derivative" of $x(t)$ is used for the asymptotic value $(x(t + \Delta t) - x(t))/\Delta t$ when $\Delta t \to 0$. In the physical case of a finite sampling rate, the neighbor samples are obviously not infinitely close in time, hence we prefer not to use "derivative" here. More precisely, we are dealt here with significantly separated samples in time, since strong aliasing is first operated (see Section 12.1.2.1), with an undersampling number up to $n = 16$. The initial 40 GHz sampling rate is respecting the oscilloscope analogue input bandwidth of 12 GHz, but the final series obtained after retaining 1 sample over 16, is corresponding to a 2.5 GHz undersampling rate. The samples obtained after this distant sample difference (which will be called later DSD) operation can thus be described as follows :

$$\{d_k^n\}_{k\in\mathbb{N}} = \{x[k\,T] - x[(k-n)T], k \in \mathbb{N}\}. \tag{4}$$

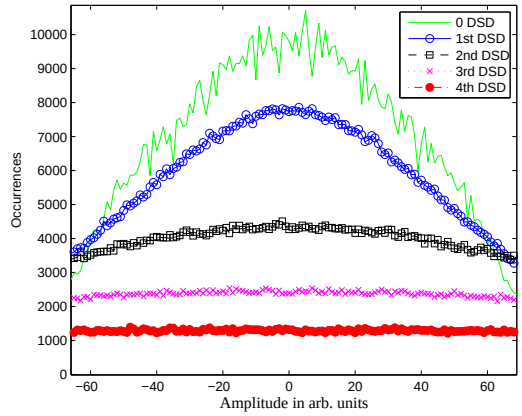
If we try again to analyze in the Fourier domain the meaning of this second processing, one obtain the following expression for the Fourier spectrum of $n-$undersampled difference signal :

$$D(\nu) = \left[2\,i\,e^{-i\pi\nu nT}\,X(\nu)\sin(\pi\nu nT)\right] \star \text{⨿⨿}_{\frac{1}{nT}}(\nu). \tag{5}$$

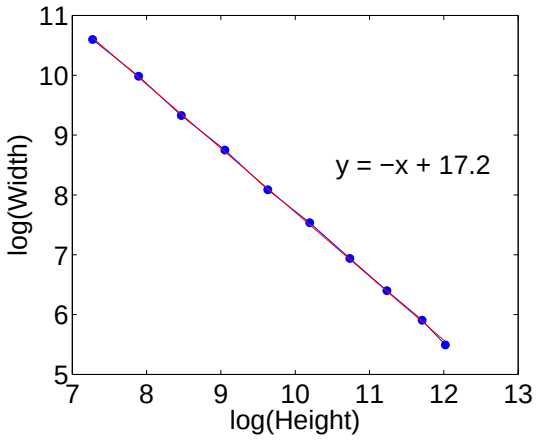
This expression reveals a so-called channeled spectrum filter, which applies a periodic sinusoidal modulation of the original spectrum $X(\nu)$. One could notice that the maximum



(a)

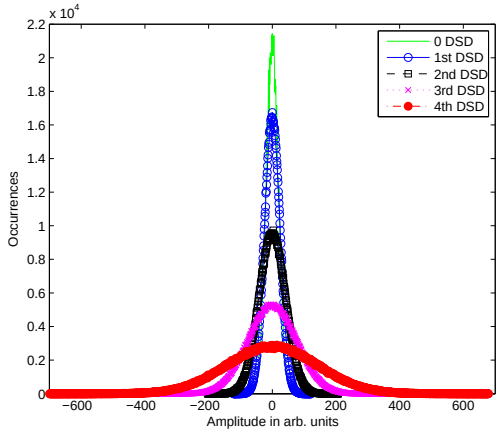


(b)

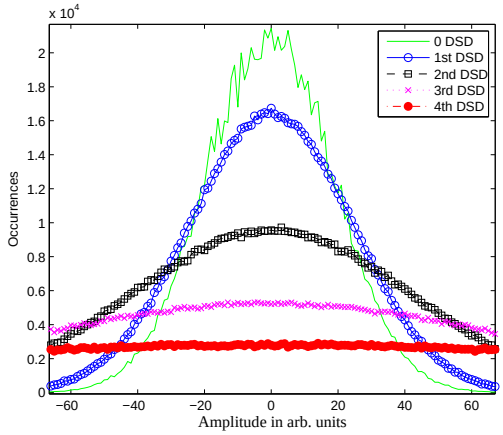


(c)

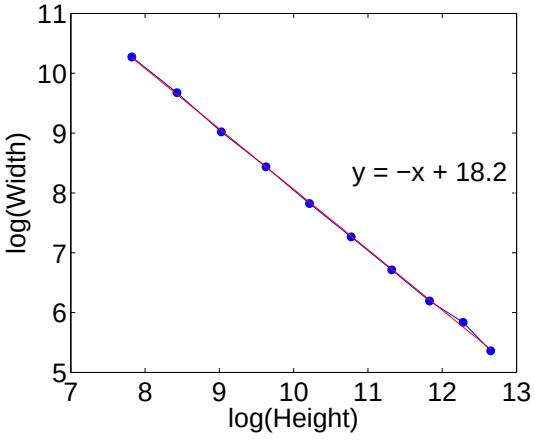
FIGURE 12.4 – (a) For 1st-4th times DSD for chaotic laser intensity sampled by 2.5GHz ADC ; (b) Zooming of the centering area of (a) ; (c) For 1st-10th times DSD, the log-log plot between weight and height, showing the hyperbolic relationship between the two as DSD is repetitively processed.



(a)



(b)



(c)

FIGURE 12.5 – (a) For 1st-4th times DSD for noisy signal sampled by 2.5GHz ADC ; (b) Zooming of the centering area of (a) ; (c) For 1st-10th times DSD, the log-log plot between weight and height, showing the hyperbolic relationship between the two as DSD is repetitively processed.

transmission of this filter is centered at half the undersampling rate ($(nT)^{-1}/2$) where aliasing is maximally symmetric (thus somehow selecting the frequency components that are most affected by aliasing), and the zero transmission frequencies are centered at zero and $\pm(nT)^{-1}$ (where the aliasing phenomenon is the less pronounced in the Fourier spectrum, as long as n is not too large). When focusing on the low frequency domain only, another comment about the action of this DSD processing could be made : the very low frequencies are filtered out, which consequence is to asymptotically set to zero the mean value of the corresponding sample set, and thus also improving the symmetry around zero of the amplitude probability distribution.

The Fourier analysis of the DSD processing is however not as obvious as for the aliasing issue in terms of randomness enhancement, or entropy amplification. A more meaningful discussion can however be made through the analysis of the statistical sample distribution of the DSD compared to the original one. More precisely, Fig.12.4a shows the evolution of the amplitude statistics when the DSD processing is iterated several times. One clearly sees that the statistics is more and more symmetric resembling closer and closer to a Gaussian distribution. This effect can be qualitatively explained through the analysis of the DSD principle. Since the difference is performed between the same sample sequence but shifted in time over a quantity large enough compared to the correlation time, one can interprete the DSD as the superposition of nearly independent pseudorandom processes. The central limit theorem can then be used to explain qualitatively the amplitude distribution convergence towards a Gaussian one (limit of the amplitude distribution for the superposition of an asymptotically large number of independent random processes).
At the same time, as more and more DSD are operated, the amplitude range is increased along the horizontal axis, whereas the maximum of the statistics along the vertical axis is inversely decreased. Figure 12.4c shows the numerical evidence of a hyperbolic relation between width and height for the successive iterated processings. Last but not least, the analysis of the statistics evolution of the sample amplitude after a few iterations of the DSD processing, allows one to realize one of the main properties for the last post-processing operation proposed in [**?**, **?**], and leading to the final random bit sequence : LSBs only retaining.

12.1.2.3/ LSB retaining, and intrinsic noise of the original sequence

When one keeps the LSBs only of the samples obtained after a few DSD processing, this means that only the small amplitudes are considered. Zooming in Fig.12.4b into the small amplitude range, *i.e.*, into the origin of the corresponding statistical histogram, one clearly sees that the statistics becomes nicely flat, thus resembling to a uniform distribution when considering the LSBs only.
A straightforward issue can then be raised about the actual source of randomness leading to the final bit sequence, as reported in [**?**, **?**]. This source of randomness has been practically attributed to the chaotic solution generated by the original physical system, a SC laser diode subjected to proper optical feedback, under such conditions that chaotic motion is

obtained. However, since only LSBs are retained in the final step of the random bit sequence extraction method, one is allowed to question about the actual influence of the always present background noise (physical noise, but also quantization noise due to the analogue to digital conversion of the fast digital scope instrument serving as the ultra-fast acquisition device). This background noise is indeed a small amplitude compound of the acquired signal, and as such, it should have a potentially important influence on the small amplitude fluctuations represented by the LSBs.

To investigate this issue, we performed a similar analysis as the one done in the previous subsections on the chaotic motion of a nonlinear electro-optic delay dynamics, but with a physical signal a priori originating from physical and digital noise sources only, without any deterministic chaotic compound. This signal is chosen to be the output of the amplified photodiode of the same setup, but without the nonlinear delayed feedback loop at the origin of the chaotic time series : the amplified photodiode signal is issued from the laser intensity noise, it comprises also the photodiode junction noise and the electronic amplifier noise (see “Optoelectronic Noise” output in Fig.12.1). Although the electrical signal level is significantly lower, we used the scope magnification to get a time trace of a comparable amplitude with respect to the scope vertical amplitude range, thus resulting in an effective digital scope quantization over a comparable number of bits with respect to the chaotic signal. Out of the physical noise sources (laser intensity noise, photodiode and electronic amplifier noise), we also have the digitization noise. The latter is typically evaluated by the scope manufacturer through an equivalent number of quantization bits when taking into account all the noise contributions involved in the digitization process. This number of effective bits is 5 to 6, meaning that the number of bits that are strongly influenced by the acquisition procedure is at least 2 (the 2 LSBs in the original time series recorded by the scope).

We have reported in Fig.12.4 and Fig.12.5 the statistics evolution of the digitally acquired optoelectronic noise signal and its width / height evolution. The figures clearly show very similar features. From this rough analysis of the influence of the two physical signals (the optoelectronic noise and the electro-optic chaos), we realize that the post-processing leads to qualitatively equivalent final bit sequence. This observation, and the previous analysis of the post-processing steps, support the assumption, at least qualitatively, that the randomness of the final bit sequence might be mainly issued from the post-processing steps. The chaotic feature of a time series appears then as actually not required for the generation of a random bit sequence, when the latter generation process is performed according to the described post-processing operations : aliasing through undersampling of the original acquired time series, DSD processing, and LSBs retaining.

12.2/ Effect of Noise on the Entropy Rate in the Binary Sequence

In this section, the time evolution of the entropy in the final binary sequence is evaluated under different choices for the method used to build the final random bit stream from

the chaotic signal. The aim is to get insight in the origin of the entropy creation mechanism involved in the construction of the final random bit stream. More precisely, we aim at discriminating under which conditions the deterministic feature of the chaotic signal (the determinism coming from the dynamics described by Eq.(1)) is indeed involved in the entropy of the extracted bit stream. To achieve this goal we reproduce the method proposed in [**?**], which is intended to measure the sensitivity to initial condition (SIC) of the deterministic chaotic motion in the presence of additional small noise. This measure consists in calculating the temporal entropy evolution for the generated binary random sequence, with respect to several different noise realizations.

12.2.1/ Introducing noise in the simulated chaotic dynamics

For the entropy calculation, we first consider a transient-free chaotic solution of Eq.(1) ($\beta = 5$). To achieve such a solution, Eq.(1) is integrated under the proper parameter conditions known to lead to a high complexity chaotic solution. This preliminary numerical integration is performed over a duration long enough compared to the slowest characteristic time scale of the dynamics (θ), so that the asymptotic trajectory is free of any transient. Once this corresponding chaotic attractor is supposed to be reached via the numerical integration, this asymptotic solution can be associated to a single temporal waveform covering only the longest time delay of the dynamics, i.e. $T + \delta T$: this is defining the initial condition of the corresponding delay based, and noise-free, chaotic dynamics, from which noise influence will be explored. We then introduce in the right hand side of Eq.(1) an arbitrary small additive noise term (small perturbation along the chaotic trajectory). The noise amplitude is arbitrarily set so that the Signal-to-Noise Ratio (SNR) is 40 dB. After further integrating Eq.(1) with the noise term and starting from the initial condition corresponding to the calculated noise free chaotic trajectory, one is able to obtain a continuously noise-perturbed chaotic trajectory. When repeating this calculation with several different noise realizations, one then expects to observe the effect of SIC when comparing the different noise perturbed chaotic trajectories. This property manifests itself through a progressive amplification (as time is running) of the small perturbations materialized by the added noise. Comparing the different calculated waveforms, they consequently looks all the same right after the noise addition, but they split apart (differently for each pair of such time series) after a typical time scale related to the inverse of the largest Lyapunov exponent of the chaotic dynamics (see Ref. [**?**] for details). Two such simulated waveforms are represented in Fig.12.6, after the undersampling procedure, and before the DSD and bit retaining processes for the final extracted binary sequence. The waveforms thus do not appear anymore as continuous in time due to undersampling. For these two realizations and with the chosen SNR for the noise amplitude, one clearly sees that the two time series separated one from each other after a typical time scale of ca. 300 ns. This time scale is of the order a few tens of the largest time delay $T + \delta T$, which is corresponding to a few tens round trips of the chaotic signal in the nonlinear delayed feedback loop. This is fully consistent with the typical order of magnitude of the inverse largest Lyapunov exponent, i.e. it is of the order

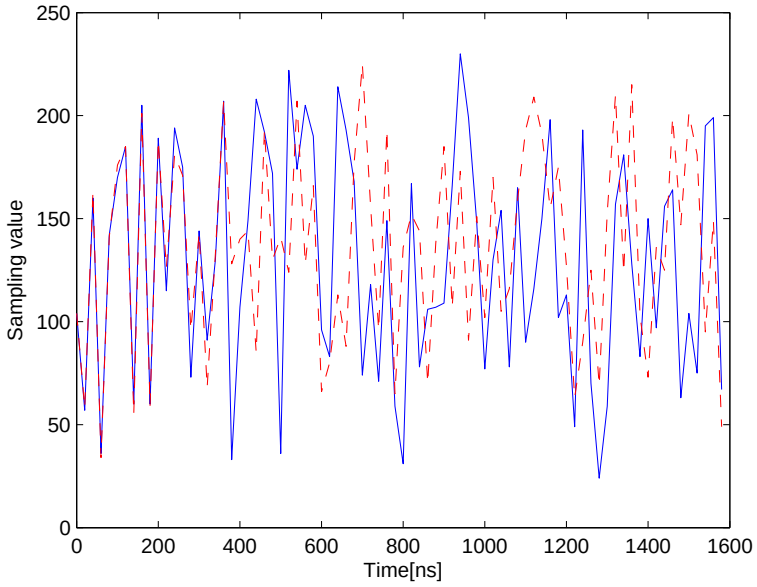


FIGURE 12.6 – Two temporal waveforms of chaotic laser intensity starting from the same initial conditions with different noise sequences added at time 0. The noise amplitude is set so that SNR is 40 dB (the signal energy being calculated on the noise-free chaotic trajectory)

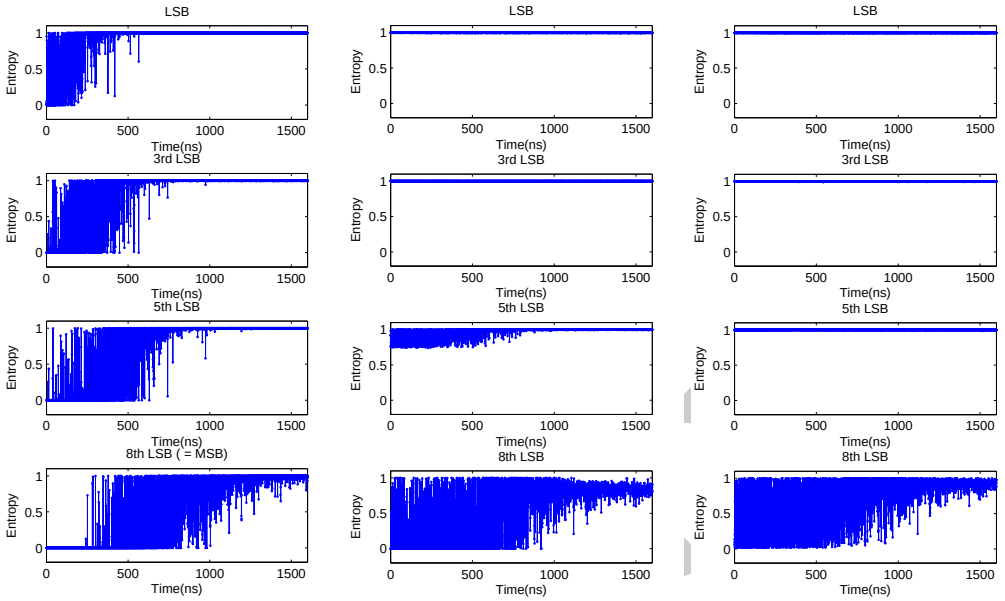
of the largest time delay in the dynamics.

12.2.2/ ENTROPY ESTIMATION FOR EACH BIT CELL

Many different noisy chaotic time series ($N = 10^3$) are simulated to generate as many random bit sequences. Each realization is calculated from the same initial conditions (the noise free chaotic waveform over one largest delay time interval), but with different added small noise perturbations. From each obtained time series, one can explore various bit stream extraction methods, e.g. with or without DSD, or even with several successive DSD processing, with the LSB retaining or with the MSB, ... For a fixed bit extraction method, the N obtained bit sequences can be used to calculate, at each time t_k of a new extracted bit, the probabilities $P_0(t_k)$ and $P_1(t_k)$ for obtaining a bit 0 or 1 respectively. This time varying probability distribution is then used to calculate how the statistical bit entropy evolves in time,

$$H(t) = -\sum_{i=0}^{1} P_i(t) \cdot \log_2 P_i(t). \tag{6}$$

As described in [**?**] if SIC of a chaotic dynamics is indeed involved in the final bit sequence, the N extracted bit sequences are initially strongly correlated. This is because the bits are originating from the same initial chaotic waveform. Consequently, the influence



(a) (b) (c)

FIGURE 12.7 – Some LSBs entropy as a funtion of time for an ensemble of time t = 0. The noise strength is −40 db, with sampling rate 2.5 GHz 8-bit ADC. (a) For sampling value ; (b) For the 1st DSD of the sampling value. (c) For the 2nd DSD of sampling value.

of the small added noise is negligible at the initial times of the deterministic chaotic dynamics : the entropy at small t is expected to be close to zero if indeed the dominating phenomena is deterministic. However, as time is evolving, SIC is amplifying the influence of the small noise amplitude on the large amplitude chaotic motion, and the N bit streams realizations are more and more decorrelated, leading progressively to a maximum binary entropy of 1. This unity entropy means an equal probability for obtaining bit zero and one, independently of any deterministic motion. The influence of the small added noise term is then dominating the output random bit stream.

Figure 12.7 represents the obtained binary entropy calculated with different bit extraction methods. In the first row, the binary entropies obtained from a direct 8-bits ADC for different bits from LSB to MSB, are plotted as a function of time. According to the description of [**?**], memory time is defined as the time required for the entropy to reach a value close to one. One clearly sees that as the bits chosen for the random bit stream moves from LSB to MSB, the memory time of the related bit cell is increasing. This illustrates that MSB is more withstand in the random bit creation process compared to LSB. Differently speaking, the small noisy compound in the chaotic signal is positively influencing randomness quality of the bit stream when extracted from the LSB. On the contrary MSBs that are related to greater amplitudes of the chaotic motion, do have a strong deterministic origin. One has to

wait for the memory time before MSBs can also lead to a good quality random bit stream. This gives an explanation already drawn in Section 12.1.2, that LSBs are more suitable for fast random bit sequence generation, because they exhibit a shorter memory time. One could also say that the speed efficiency in the random bit sequence generation process is better with the LSB retaining method, because mostly small amplitude noise is actually involved without any deterministic origin.
Figure 12.7 also reports on the entropy creation in time, when one or more DSD processing is used. This is represented in the second and third column of the figure. The contribution of DSD clearly appears in the entropy rate. DSD is shortening the memory time, thus speeding up the possible rate at which actually "good" randomness quality can be obtained. Even when intermediate bit cells are used (e.g. 5th LSB, and twice DSD), a perfect unity entropy is already achieved at the very first time. An interpretation could be that DSD is amplifying the non deterministic small noise influence, and thus is attenuating the detrimental large amplitude determinism.
The conclusion on Fig.12.7 is that fastest entropy rate (down to the actual sampling) can be achieved when LSBs are used, and when several DSD processing are performed. This corresponds to the plots on the upper right positions, for which unit entropy is already achieved very close to the time origin. On the contrary, the MSBs are showing a non-zero memory time, meaning that higher entropy for good randomness quality can only be obtained if slower undersampling is used. Two successive bits would need to be separated by the memory time in order to get a good randomness quality. The worst conditions are shown in the lower left plots, with MSBs and without DSD processing. One could notice that MSB is actually equivalent to the 1-bit ADC used in [**?**], where we can suspect that the deterministic chaotic motion plays indeed a crucial role in the final random bit stream instead. On the opposite, the LSB retaining method of [**?**] makes use of the small background noise present in the chaotic time trace, with a probably small or negligible contribution of any deterministic chaos origin.

Fig.12.8 shows the plots of every bit cell entropy averaged over 10^3 trajectories. Each plot of entropy is obtained as a function of time for an ensemble of time series starting with exactly the same initial condition at time $t = 0$. Eight plots are shown in Fig.12.8 corresponding to eight different position bit cell of the value. These curves are the smoothed versions, due to averaging, of the curves represented in the first column of Fig.12.7. Again, it can be seen that more time is required to converge to a unity entropy when using MSB compared to the use of LSB. Differently speaking, the memory time depends on bit cell selecting, MSB and LSB appearing as the slowest and fastest entropy increasing rate, respectively.

12.3/ Statistical Tests

Additionally to the previous signal theory analysis of the processing steps used in the bit extraction method, this section is intended to qualify the final bit stream in terms of their

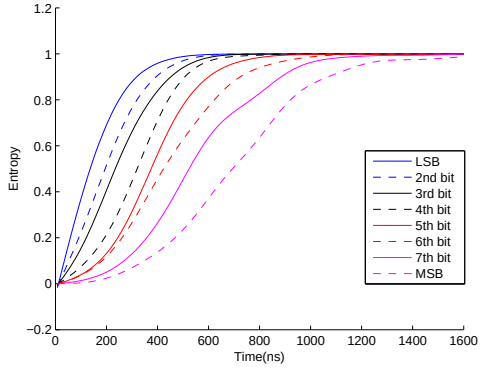


FIGURE 12.8 – Averadge growth of bit entropy and its dependence on bit cell selecting (from MSB to LSB)

benchmarking from several standard randomness test suites. We thus verify in this section that the analyzed and used method proposed in [**?**] and [**?**] have led also in our experiment to quasi-equivalent random bit stream quality, whether from the deterministic EO phase chaos generator or with the optoelectronic noise source.

12.3.1/ THE TESTED STREAMS

First of all, we give here a brief description of the tested methods that have been formerly proposed in [**?**] and [**?**].

On the one hand, in [**?**], authors have used a DSD method to generate a random bits stream. The chaotic laser signal is sampled by using a 2.5 GHz ADC, and then 5 LSBs of every first DSD value are joined together to generate the final random sequence. On the other hand, in [**?**], the chaotic laser signal is sampled thanks to a 20 GHz ADC. Then the DSD operation is processed 4 times and 8 LSBs of each value are joined to produce the pseudorandom bit stream.

These two schemes are both adapted to optoelectronic noisy signal. The generated streams sourced from chaotic laser and noise are compared by standard statistical tests in the next subsections.

12.3.2/ STATISTICAL TESTS

We have previously shown that various statistical tests can be designed to evaluate the assertion that a given sequence is generated by a perfectly random source. In this section, we have performed some statistical tests on the optoelectronic noise and electro-optic chaos generators considered here. These tests include NIST suite [**?**], DieHARD battery of tests [**?**], ENT program [**?**], and Comparative test parameters.

TABLE 12.1 – NIST SP 800-22 test results ($\mathbb{P}_T$)

Method	2.5GHz, 1st DSD, 5LSB		20GHz, 4th DSD, 8LSB	
Source	Chaotic laser	Noise	Chaotic laser	Noise
Frequency :	0.935716	0.798139	0.171867	0.834308
BlockFrequency :	0.040108	0.350485	0.289667	0.867692
CumulativeSums :	0.334152	0.575225	0.228927	0.688782
Runs :	0.595549	0.834308	0.851383	0.637119
LongestRun :	0.191687	0.964295	0.162606	0.304126
Rank :	0.534146	0.037566	0.637119	0.719747
FFT :	0.236810	0.514124	0.202268	0.249284
NonOverlappingTemplate :	0.502510	0.491449	0.521769	0.501830
OverlappingTemplate :	0.851383	0.964295	0.090936	0.574903
Universal :	0.798139	0.739918	0.102526	0.319084
ApproximateEntropy :	0.224821	0.236810	0.435436	0.419021
RandomExcursions :	0.347389	0.229729	0.471174	0.104312
RandomExcursionsVariant :	0.217344	0.209317	0.461569	0.350467
Serial :	0.300289	0.366918	0.237996	0.606177
LinearComplexity :	0.350485	0.262249	0.224821	0.935716

12.3.2.1/ NIST STATISTICAL TEST SUITE

In Tab.12.1, the random streams generated by the chaotic laser and by the noisy signal have both obtained a 100% passing rate when considering the NIST battery of tests, thus it is impossible to found a difference between the two streams using the NIST suite.

12.3.2.2/ DIEHARD BATTERY OF TESTS

Tab.12.2 gives the results derived from applying the DieHARD battery of tests to the two random streams considered in this work. As it can be observed, both of them can pass the DieHARD battery of tests. Another time, the statistical properties of the random stream taken from the chaotic laser intensity indicates similar statistical features compared to the one obtained by the optoelectronic noise source.

TABLE 12.2 – Results of DieHARD battery of tests

No.	Test name	Generation Method			
		2.5GHz, 1st DSD, 5 LSB		20GHz, 4th DSD, 8 LSBs	
	Source	Chaotic laser	Noise	Chaotic laser	Noise
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
	Passing rate	18/18	18/18	18/18	18/18

12.3.2.3/ ENT TEST PROGRAM

In Tab.12.3, it is shown that the results for each pair of random streams, considering these five tests detailed above, are very closed one to each other. They all achieved to pass the threshold of the Chi-squared test, and the results are very similar for the other tests. To sum up, all these streams satisfy the same random-like behavior according to the ENT battery.

Table 12.3 – ENT battery using 10^8 bits for each stream

Method	Using source	Entropy	Chi-square	Sample	π error	Correlation
2.5GHz, 1st	Chaotic laser	7.999984	67.18%	127.4988	0.03%	-0.000771
DSD, 5 LSBs	Noisy signal	7.999986	7.13%	127.5034	0.03%	-0.000392
20GHz, 4th	Chaotic laser	7.999986	73.48%	127.4973	0.03%	0.000481
DSD, 8 LSBs	Noisy signal	7.999985	12.37%	127.5011	0.02%	-0.000411

Table 12.4 – Comparison between the presented sources for a 2×10^7 bits sequence

Subjects	Monobit	Serial	Poker	Runs	Autocorrelation
Method	2.5GHz, 1st DSD, 5LSBs				
Chaotic laser	0.2509	1.9200	16.6650	16.6215	1.5739
Noise	0.6019	0.7144	8.5606	17.5156	1.5247
Method	20GHz, 4st DSD, 8LSBs				
Chaotic laser	1.4580	0.5199	13.1430	28.9460	1.1583
Noise	0.2554	0.7835	14.0035	22.9136	1.6739

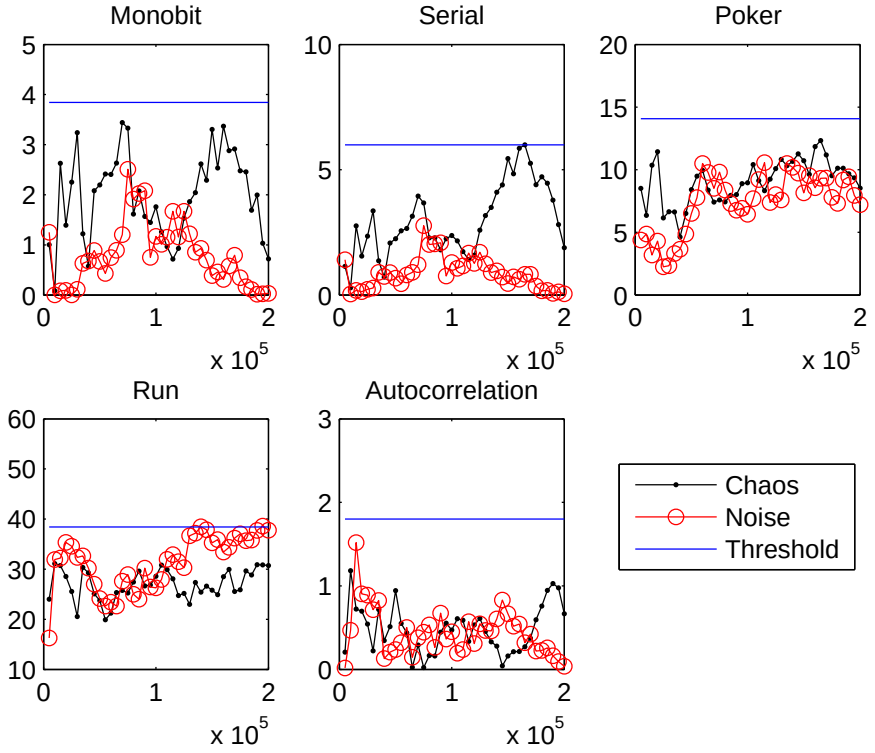
12.3.2.4/ Comparative test parameters

We show in Tab.12.4 a comparison between two random bits streams sourced respectively by the chaotic laser intensity and by the noisy optoelectronic signal. The results confirm that the proposed random streams present very closed statistical qualities. This finding implies that to have a chaos-like deterministic origin is not a required condition for high randomness quality in the proposed method.

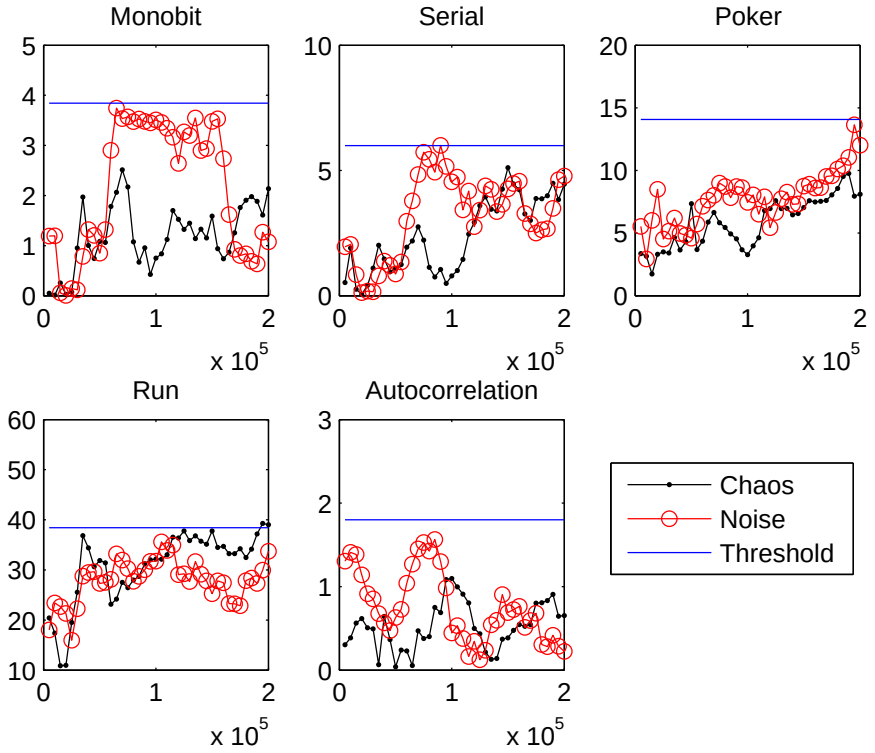
Finally a comparison of the overall stability from 5×10^3 to 2×10^5 for these generators is given in Fig.12.9. It can be seen that the trends for the amplitude movements of values are more or less in the same scale, which again indicates that all these random sequences share closed random properties.

12.4/ Discussion, Conclusion, and Perspectives

Random number generation via photonic broadband signal generation does provide nowadays a novel and interesting approach allowing for unprecedented high bit rate of random bit streams. These photonics to digital world conversion are designed so that randomness can be certified according to most of the usual randomness tests such as NIST and DieHARD suites. Among the recently proposed physical systems and related processing intended to extract bit streams from photonic analogue waveforms, two rather different ap-



(a) 2.5GHz, 1st DSD, and 5 LSB method



(b) 20GHz, 4th DSD, and 8 LSB method

FIGURE 12.9 – Overall Sequence Stability Comparison

proaches can be identified : when the source of randomness explicitly stems from photonic noise [**?**, **?**], and when deterministic chaos is claimed to be at the origin of the random bit stream [**?**, **?**]. Whereas the first approach provides obviously, and by definition, a non deterministic random bit stream, the second one has an implicit source of determinism, similarly to the algorithmic and fully digital pseudorandom bit sequence (PRBS) generators.
A major interest of the digital PRBS resides in their capability of generating a distant and synchronized random bit stream, which allows one to apply them in symmetric cryptography. A major advantage of PRNG is precisely their perfect determinism, and perfect control, due to their digital program-based generation process. This feature is also at the origin of their main drawback : the same absolute digital determinism can be used in principle for cryptanalysis, trying to guess the seed which can then deterministically and totally allow for the random bit sequence reproduction, even by an eavesdropper. Also from a more technical viewpoint, the processor based architecture of PRNGs defines some speed limitations related to the processor clock, and the number of elementary operations needed to implement the PRNG algorithm. Noise based, and chaos based, photonic RNGs provide at least a technical answer to the limited bit rate generation provided by purely algorithmic solutions. It was also reported in many attempts on photonics based RNGs, that high quality randomness is possible, since they can pass successfully all the standard NIST and DieHARD test suites. A strong open problem however still remains concerning the capability to control, and reproduce, the random bit stream provided by photonic chaos-based RNGs. On the contrary to the photonic noise based RNGs, this indeed can be expected from the chaos-based photonic RNGs, since they also originates, at least partially, on deterministic dynamics, similarly to the algorithmic PRNGs.
In that particular context, we have proposed in this article to address related issues, through an analysis of the deterministic origin of the chaos-based photonic RNGs proposed in [**?**]. Indeed, a major difference between purely digital PRNGs and photonic chaos-based RNGs resides in the presence of unavoidable non-deterministic noise sources. This noise compound can be used as an argument to support the idea that one might find an “analogue protocol”, such that a non-authorized receiver would never be able to reproduce the bit stream. This would then lead to an absolutely secure symmetric cryptographic scheme, e.g. if the Vernam cipher scheme is employed. Before reaching this absolute dream of secure communications, one has however to clearly identify how deterministic the final random bit stream, or differently speaking, how far any deterministic mechanism indeed does exist in this bit stream so that one can think about using it to reproduce the random bit stream. This is one of the main point addressed in this chapter, in the particular context of the method proposed in [**?**].
The particularity of this method, is that it combines both photonic chaos, but also a significant digital post-processing before the extraction of a final bit stream with excellent randomness. We have reproduced in this chapter this method on other photonic sources of analogue entropy. One is consisting in a mainly deterministic electro-optic phase chaos generator, a broadband chaos generator recently used to demonstrate the currently fastest analogue chaos based communication scheme. Determinism in this photonic chaos setup

can be claimed as a dominating feature, since it was used and controlled to achieve a field experiment of chaos communication at 10 Gb/s. On the contrary, another photonic source of entropy was also considered in the same photonic RNG method. This second photonic source of entropy can be trustfully considered as of mainly noisy origin, without any deterministic and controllable compound : its origin is typically attributed to intrinsic diode laser RIN (relative intensity noise), photodiode detection noise, and electronic amplifiers noise. We found that the method proposed in [**?**] gave fully comparable results in terms randomness quality of the generated bit stream. This is fully consistent with the questioning already addressed in [**?**, **?**] about the actual origin, noise or chaos, of the random bit stream provided by this method. Additionally to this result, we have proposed analysis of this method on the basis of standard signal theory and sampling theory. The conclusion of our analysis strongly support that the method is essentially exploiting the noisy compound always present in a photonic signal, would it be with (chaos) or without (noise) deterministic motion. Our analysis has highlighted in the post-processing bit extraction method, two dominating mechanisms leading to a high quality random bit stream. One is related to the natural spectral mixing occurring when aliasing is involved. In the acquisition method proposed in [**?**], an undersampling of a factor 16 is indeed performed, which is equivalent to a strong aliasing condition in the photonic samples extraction. This aliasing phenomenon definitely enhances the randomness quality of the original signal (would it be of chaotic or noisy origin), through its well known complex amplitude mixing in the Fourier domain. Another post-processing which we have called DSD for distant sample difference, was also shown to contribute to the randomness quality of the final bit stream. Combined together with an LSBs retaining only, we showed numerically that the final bit stream appears to be extracted from a nearly uniform probability distribution of the amplitudes corresponding to the retained LSBs. The LSB retaining procedure also obviously gives an important role to the small amplitudes which are mainly dominated by noise sources, whereas MSB are of course more correlated with large amplitude deterministic motion. To develop an evidence of the actually most important role of the noise instead of the deterministic chaotic motion, we have proposed to analyze the binary entropy rate for each selected bit, from LSB to MSB, without or with one or several DSD post-processing. The existence of a non zero memory time was found only for large amplitude bits (close to MSB), and for a small number of DSD post-processing. This configuration which reveal a signature of the deterministic origin via the evolution of the binary entropy, is the opposite one with respect to the choice proposed in [**?**]. It thus confirms that negligible deterministic origin (and thus the chaotic light motion) actually enters in the randomness of the final bit stream.

Using the MSB (or equivalently 1-bit ADC conversion) is according to us the way to keep a deterministic origin in the generation of a random bit sequence. Further work on ultra-fast RNGs based on deterministic chaos, should thus concentrate on this approach for the bit extraction method from the original photonic chaotic waveform. Careful post-processing will however be very important, to ensure that randomness quality is indeed obtained though strong determinism is concerned. In [**?**], this was achieved through the use of two independent chaotic lasers, with careful chosen resonant frequency and external cavity delayed

feedback, and a XOR operation between their respective 1-bit ADC. Finally, the most difficult task will be to design a proper coupling scheme with a binary bit stream, so that distant random sequences can be synchronized and used for cryptography.

13

CI Random Stream Generation Via Optoelectronic Chaotic Laser

According to the analysis of the previous chapter, MSB is a key factor to preserve the deterministic origin in the generation of random streams using optoelectronic chaotic lasers. Since the MSB keeps the most large part of information on the chaotic laser signal, the statistical performance of the extracted output bit stream is not good. Hence, in this chapter, we will try to use the method based on chaotic iterations and presented in previous chapters to fulfill this drawback. Indeed, this method has been able to improve the statistical properties of various defectives PRNGs, thus we think interesting to regard weather this approach can work too for chaotic laser based pseudorandom generation.

13.1/ Setup of a broadband optoelectronic chaotic laser signal

We have presented in the previous part of this manuscript 5 generators based on chaotic iterations, namely the four versions of CIPRNG, and the XOR CI generator proposed in [**?**], which is a GPU adaptable version of the aforementioned CI methods. These schemes are not only able to assign discrete chaotic properties to the output streams, but the statistical properties of the generators set as parameters are also improved. This is why we tried to use them on an optoelectronic device similar to the one studied in the previous chapter.

Indeed, as shown in Fig. 13.1, the chaotic laser signal considered in this chapter is similar to the one described in Section 12.1, except that there is no output signal for photonic noise. This physical source of entropy presents a very strong determinism. The output chaos signal is generated by a nonlinear dual delay differential equation implemented in an optoelectronic and electro-optic feedback loop.

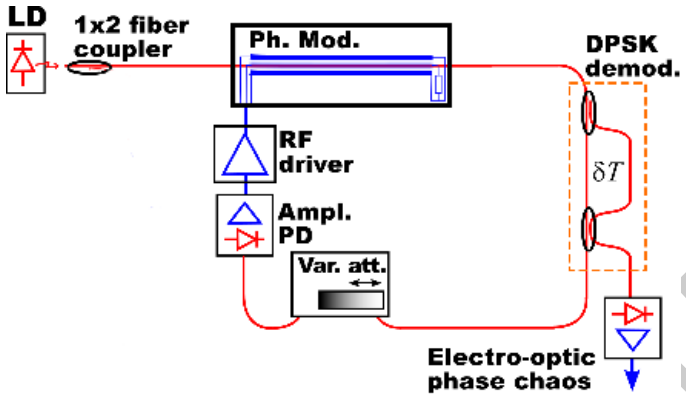


FIGURE 13.1 – chaotic laser signal setup

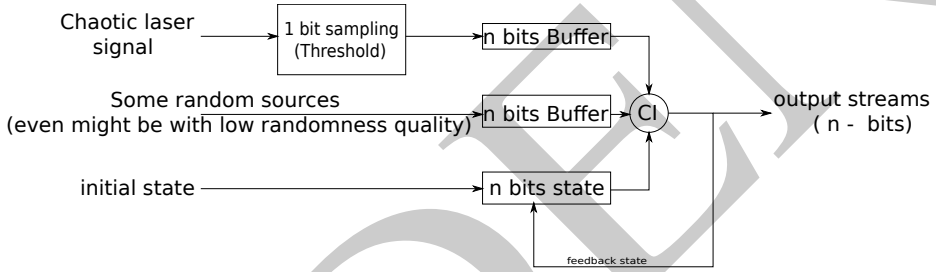


FIGURE 13.2 – The scheme of applying CI to chaotic laser

13.2/ RANDOM STREAM GENERATION APPROACH BY APPLYING CI METHOD TO CHAOTIC LASER

In this section, a last structure for random sequence generation is presented : the optoelectronic chaotic laser is post-processed by using chaotic iterations. Two examples are then shown and the NIST test suite is used to evaluate the produced outputs.

13.2.1/ THE PROPOSED SCHEME

For achieving randomness, three different inputs are used and mixed using the chaotic iterations post-processing :
– the chaotic laser sampled to 1-bit (collected as n-bits integer using a buffer),
– another given (pseudo)random source to determine (producing n-bits values),
– and the current state of the system.
As depicted in Fig.13.2, the chaotic laser signal is sampled by a 1-bit ADC. The sampling rule consists to compare the voltage of the chaotic laser signal with a threshold value : if the voltage is bigger than the threshold, then the output bit is 1, else it is 0. This binary stream provided by a 1-bit sampling of the chaotic laser cannot exhibit a good statistical profile, as

a lot of "chaos information" is lost during the sampling. This is why we propose to employ two other inputs, namely another (pseudo)random number source and the current state of the iterated system. As shown in Fig.13.2, these two inputs are mixed with the chaotic laser binary stream using chaotic iterations. Each n-bits values obtained by this way is both returned as output of the device and reused in the next iteration of the system.

13.2.2/ Examples of processing

Examples are given in this subsection by using two usual generators as random source input : XORshift and LFSR (Linear Feedback Shift Register [**?**]). They have been chosen due to their rapidity and very simple design, making them easy to implement in hardware. The CI method is a modified version of the XOR CI generator presented in [**?**]. In the next section, we give a brief description about this new XOR CI generator, we define the LFSR and recall the XORshift PRNGs for easy reading of this manuscript, and we provide the parameters of the laser setup.

13.2.2.1/ A new XOR CI generator

As recalled previously, the XOR CI method is one way to use chaotic iterations as PRNGs [**?**]. Compared to the other CIPRNG schemes, XOR CI is very compatible in physical design. In XOR CI, at each iteration, a subset of binary digits from the output value of a PRNG is chosen to be switched, this subset depending on the value of a second generator. Such an attempt leads to a sort of merger of the tow sequences. Our XOR CI algorithm only consists in adding the chaotic signal provided by the laser into his process. The corresponding algorithm can be written as follows.

Algorithm 14 An arbitrary round of the respective XOR CI generator

Input **:** the internal state x (an array of 4-bit words)
Output **:** an array r of 4-bit words

1: $b \leftarrow Chaotic\_Laser\_data()$;
2: $a \leftarrow PRNG_1()$;
3: $x = x \oplus a \oplus b$;
4: $r \leftarrow x$;
5: return r ;

For $PRNG_1$, we will use XORshift and LFSR in our simulations ; they are recalled below.

13.2.2.2/ XORshift

The definition of this well-known generator is reminded again in Algo.15 :

Algorithm 15 An arbitrary round of XORshift algorithm

Input : the internal state z (a 32-bits word)
Output : y (a 32-bits word)

1: $z \leftarrow z \oplus (z \ll 13)$;
2: $z \leftarrow z \oplus (z \gg 17)$;
3: $z \leftarrow z \oplus (z \ll 5)$;
4: $y \leftarrow z$;
5: return y

13.2.2.3/ LFSR

In computing, LFSR is a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is XOR. Thus, a LFSR is most often a shift register whose input bit is driven by the exclusive-or (XOR) of some bits of the overall shift register value. This generator, described Algo.16, has a period equal to 65535.

Algorithm 16 An arbitrary round of LFSR algorithm

Input : the internal state z (a 16-bits word, set as $0xACE1$ initially)
Output : y (1 bit)

1: $y = ((lfsr \gg 0)^{(}lfsr \gg 2)^{(}lfsr \gg 3)^{(}lfsr \gg 5))\&1$;
2: $z = (z \gg 1)|(bit \ll 15)$;
3: return y

13.2.2.4/ Photonic chaos

For experiments purpose, a chaotic laser has been stimulated in computer using the method detailed in [**?**]. The frequency of the signal is equal to $F_1 = 4000$ GHz, and other corresponding parameters are listed as :

– **Time delay** : Set XORshift as 60ns ;
– **High Frequency Cut-off** : 13ps ;
– **Ratio between high and low frequency cut-off** : 5×10^{-5} ;
– **Type of nonlinear transformation** : square of sinus ;
– **Highest complexity chaotic regime** : 5.

When the laser data are being processed, one value every 400 is selected to be 1-bit sampled (10 GHz). Since the output of XORshift PRNG is in 32 bits, thus we have $n = 32$ in Fig.13.2.

TABLE 13.1 – NIST SP 800-22 test results ($\mathbb{P}_T$)

Method	CI with XORshift	XORshift	CI with LFSR	LFSR	No CI
FT	0.935716	0.14532	0.12414	0.000000	0.171867
FBT	0.040108	0.45593	0.73425	0.000000	0.289667
CST	0.334152	0.21330	0.000000	0.000000	0.000000
RT	0.595549	0.28966	0.88330	0.000000	0.000000
LROBT	0.191687	0.000000	0.01245	0.000000	0.000000
BMRT	0.534146	0.005350	0.42803	0.000000	0.000000
FFT :	0.236810	0.50365	0.33157	0.000000	0.000000
NOTMT	0.502510	0.86769	0.51246	0.000000	0.000000
OTMT	0.851383	0.27570	0.01235	0.000000	0.090936
MUST	0.798139	0.92407	0.81246	0.000000	0.000000
AET	0.224821	0.75792	0.00000	0.000000	0.000000
RET	0.347389	0.41902	0.91235	0.000000	0.471174
REVT	0.217344	0.81154	0.12592	0.000000	0.461569
ST	0.300289	0.41923	0.000000	0.33122	0.237996
LCT	0.350485	0.52833	0.81245	0.000000	0.224821
Success	15/15	14/15	13/15	0/15	7/15

13.3/ NIST TEST RESULTS

The randomness of the output streams of these two simulated examples are evaluated by the standard NIST test suite. The results are shown in Tab.13.1. We can see that, if no CI is applied, the chaotic laser sampled bits cannot pass the battery. As usual, the generator XORshift fail one test. And none can be passed for LFSR, since this generator has a very low period.

With the chaotic iteration based post-treatment referred as CI method, the performances have been improved obviously (see Table 13.1). We can conclude that, even though the participated random sources are not as good as possible, the CI method still remains to be able to improve the statistical properties of the inputted generators.

As suggested by these short experiments, the combination of chaotic laser and our CI method may lead to statistical improvement of the physical signal. If these results are confirmed by further investigations, such a way to proceed may provide an interesting way to produce physically no periodic reproducible, ultra fast, and chaotic random streams with good statistical profiles.

IV

Conclusion and Perspectives

14

Conclusion

In this thesis, two parts of works have been dedicated. In the first part, we present some scheme for a special class of PRNGs based on chaotic iterations (CIPRNG). They have been evaluated statistically. FPGA is also applied to enhance the CIPRNG performance. Then some applications taken from the field of cryptography are finally proposed. In the second part, a preparation for the application of CI into broadband optoelectronic entropy sources to generate ultra fast random bits has been done, using the MSB (or equivalently 1-bit ADC conversion) has been proven the way to keep a deterministic origin in the generation of a random bit sequence.

14.1/ CI method

Chaos, being a special class of nonlinear dynamics, has aroused a lot of interests since its emergence and formulation in science these last decades. Various distinct behaviors, such as random-like exhibition, sensitive dependence to initial conditions, and unpredictability, together with their inherent determinism and simplicity of realization, justify this interest. More recently, some researchers have explored the possibility to apply these chaotic dynamics for cryptographic applications and designs. To become an element to take into account in cryptography, the nature of randomness and chaos have been connected. Thus interrelations between chaos-based random number generators and cryptographic systems have been investigated. However, such investigations have led to controversy, as people involved in cryptography not always understand the tools and approaches of the nonlinear community, and these latter do not deal rigorously (mathematically) with security, often forgetting to take into account one century of formalization and advances in cryptography.

These difficulties are reinforced by the fact that tools manipulated in nonlinear science are often inadequate, far from the targeted applications in computer science. Indeed, in most of the designs, the generation of chaos is obtained by a recurrence relationship or a set of state differential equations, in which perfect model is assumed. It always requires a continuous space domain so that the actual chaotic dynamics can be observed. However, this space domain implies the use of real-number or infinite bit representation in system realization. As pinpointed by some researchers, the digital dynamical properties of a chaotic

system will be far different from its continuous ones. For instance, when some piecewise linear maps are realized in finite computing precision, some severe problems such as short cycle length, non-ideal distribution, and high correlation, have been observed and reported. Having these issues in mind, it seems more realistic and practical to consider a chaotic system realized in finite precision domain. In previous work, a a family of new chaotic pseudo-random bit generators called CI PRNGs is proposed, which is going to try to fill the gap.

In this part of thesis, the CI PRNGs method is developed and evaluated. These generators are based on discrete chaotic iterations, which satisfy the well respected Devaney's definition of chaos. The rigorous framework for generate chaotic random sequence in [**?**] has been developed and renewed. Firstly the researches of the CIPRNG Version 1 method has been deepen ; Then and new version : CIPRNG Version 3 is represented by using lookup table method ; At last, CIPRNG Version 4 which is specifically designed for FPGA hardware is indicated.

The randomness and disorder generated by these versions CIPRNG algorithms have been widely evaluated, this depends on both the proof of theoretical properties and the scores on numerous statistical tests. Various statistical tests are available in the literature to check empirically the statistical quality of a given sequence. The most famous and important batteries of tests for evaluating PRNGs are, namely : the TestU01, NIST, and DieHARD batteries. To evaluate the propositions of this thesis, a comparative study between various generators and our own PRNG has been carried out and statistical results have been highlighted. They all lead to the conclusion that these generators can be considered as candidates for a large variety of applications in computer science, and in the security field too. The CI algorithm has been also modified to use in FPGA, which leads a huge speed improvement.

The PRNG we have proposed is based on the so-called discrete chaotic iterations. It is a composite generator that combines the features of two other PRNGs. The intention of this combination is to accumulate the effects of chaos (brought by the chaotic iterations) and randomness (taken from the inputted generators). The results of comparative test parameters confirm that the proposed CI PRNGs are all able to pass these tests, and improve the statistical properties relative to each generator taken alone. Also ten famous classic PRNGs are applied to show the CI method is able to enhance the statistical performance, and this provide a technology to make PRNG withstand the attacks in future.

This is why application examples in cryptography are finally given at the end of this part. We have more specifically study the possible use of such a family of generators for digital watermarking. Security has not been investigated, only robustness to some frequency and geometric attacks has been evaluated. This case study enables us to precise the details of the algorithm and to give a concrete illustrative example of the interest to possess a generator being both random and chaotic. The proposed CI PRNGs can be used to fulfill needs of image encryption. Detailed analyses show that they can provide high level of security.

14.2/ Optical source

Recently, for unprecedented high bit rate of random sequences , very attractive solution has been given by photonic broadband signal to generate random numbers. There are two different origins to extract bit streams from photonic analogue waveforms according to nowadays physical systems : the source of randomness explicitly stems from noise [**?**, **?**] and deterministic chaos [**?**,**?**]. In this context, we have proposed in the thesis to analysis a related method, though an analysis of the deterministic origin of the chaos-based photonic RNGs proposed in [**?**, **?**]. The particularity of this method, is that it mixes both photonic chaos, but also a significant digital post-processing before the extraction of a final bit stream with excellent randomness. Here in the thesis, this method on other photonic sources of analogue entropy has been redone. To be noticed, in one hand, the random streams is extracted by a mainly deterministic electro-optic phase chaos generator, which recently is used to demonstrate the currently fastest analogue chaos communication at 10 Gb/s. In the other hand, the scheme with another photonic source of entropy is also considered : the photonic source of entropy can be trustfully considered as of mainly noisy origin, without any deterministic and controllable compound : its origin is typically attributed to intrinsic diode laser RIN (relative intensity noise), photodiode detection noise, and electronic amplifiers noise (check in Fig. 12.1). During the processing, we found that the method proposed in [**?**, **?**] gave fully comparable results in terms randomness quality of the generated bit stream. This is fully consistent with the questioning already addressed in [**?**, **?**] about the actual origin, noise or chaos, of the random bit stream provided by this method. More than that, we have proposed analysis of this method on the basis of standard signal theory and sampling theory. The conclusion of our analysis strongly support that the method is essentially exploiting the noisy compound always present in a photonic signal, would it be with (chaos) or without (noise) deterministic motion. Our analysis has highlighted in the post-processing bit extraction method, two dominating mechanisms leading to a high quality random bit stream. One is related to the natural spectral mixing occurring when aliasing is involved, another post-processing which we have called DSD for distant sample difference, was also shown to contribute to the randomness quality of the final bit stream. At last, attached together with LSBs retaining only, it can shown that the the small amplitudes which are mainly dominated by noise source have given an important role to the outputs. To develop an evidence of the actually most important role of the noise instead of the deterministic chaotic motion, analyzing the binary entropy rate for each selected bit from LSB to MSB, without or with one or several DSD post-processing are proposed. It is only found the existence of a non zero memory time in parts closed to MSB, for a small number of DSD post-processing. This configuration which reveal a signature of the deterministic origin via the evolution of the binary entropy, is the opposite one with respect to the choice proposed in [**?**, **?**]. It thus confirms that negligible deterministic origin (and thus the chaotic light motion) actually enters in the randomness of the final bit stream.

Applying the MSB (or equivalently 1-bit ADC conversion) is according to us the solution that processing a deterministic origin random bit sequence generation. This has provided

the preparation for applying CI in photonic signal to generate deterministic random streams. At last two examples of mixing CI and MSB of optoelectronic chaotic signal is given, the results of NIST test suite show the improvements of randomness, in future, the practical implementation is looking forwarded.

15

Perspectives

In this Section, we mention a few question which we would like to examine further.

15.1/ CI method

We considered in detail the CIPRNG model using a random update function. It would be interesting to know if we can use the same approach for other more primitive input sequences and keep their properties. And, 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. A larger variety of tests will be considered to compare this PRNG to existing ones, and a cryptanalysis of our generator will be proposed. The chaotic behavior of the proposed generator will be deepened by using the various tools provided by the mathematical theory of chaos. 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.

15.2/ Optical source

According to us, the way to keep a deterministic origin in the generation of a random bit sequence is using the MSB (or equivalently 1-bit ADC conversion). In future work, we will concentrate on this approach for the bit extraction method from the original photonic chaotic waveform. Finally, the most difficult task will be to design a proper coupling scheme with a binary bit stream, so that distant random sequences can be synchronized and used for cryptography.

Abstract : As any well-designed information security application uses a very large quantity of good pseudorandom numbers, inefficient generation of these numbers can be a significant bottleneck in various situations. Consequently, pseudorandom number generators (PRNGs) are very important primitives widely used in numerous applications like numerical simulations or security. For instance, they are one of the most fundamental component that any cryptosystem has to embed, in order to generate encryption keys or keystreams in symmetric ciphers.
Nowadays, due to the distinct properties of chaos, including random-like dynamics and high sensitivity on initial conditions, the use of chaos in random number generation and cryptographical applications has aroused tremendous interests. In previous researches, a technique that applies some well-defined discrete iterations, satisfying the reputed Devaney's definition of chaos, has been developed. It has been proven that the generators embedding these chaotic iterations (CIs) produce truly chaotic random numbers.
In this thesis, the schemes that generate pseudorandom number based on discrete chaotic iterations are rethought and optimized. This manuscript contains mainly two parts. In the first part, the original methods are firstly researched deeper and proven to be crytographically secure, and secondly two faster generators with better statistical performance are introduced. Then the statistical analysis of the chaotic iterations methods for random number generators are summarized, while their strength and weakness are also commented, which lead us to improve the generation rate of such generators. These generators based on chaotic iterations are then redesigned specifically for Field Programmable Gate Array (FPGA) hardware. Finally, practical applications that implement CI technology in software and hardware are expressed, and their evaluations are also done.

In the second part, a scheme using optoelectronic chaotic signal to generate random streams is studied. In 2009, Reidler *et al.* published a paper entitled "An optical ultrafast random bit generator", in which they presented a physical system for random number generation based on a chaotic semiconductor laser. This generator is claimed to reach potentially the extremely high rate of 300 Gb/s. This method is analyzed in the thesis, and the discussion about actual origin of the randomness and the actually reachable bit rate have been given. We show that the actual binary sequence randomness is corresponding more to complex mixing of noisy components performed by digital post-processing operations, than to chaotic properties of the signal. We also address the issue of a proper way to use chaotic motion in RNGs, which is shown to necessarily involve MSB instead of the LSB. Then, at the end of the manuscript, two examples of adapting chaotic iterations with simulated chaotic waveform to generate random sequences are shown, and finally NIST test suite is used to evaluate the randomnesses.

Keywords : Chaotic sequences ; Statistical tests ; Discrete chaotic iterations ; Information hiding ; FPGA ; Optoelectronic