Chaos-based security under real-time and energy constraints for the Internet of Things

Ons Jallouli

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Ons JALLOULI

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le sceau de l'Université Bretagne Loire

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Electronique/spécialité:Sciences de l'Information et de la Communication.
Unité de recherche : Institut d'Electronique et de Télécommunications de Rennes UMR CNRS 6164(IETR)

Soutenue le 27 Octobre 2017

Chaos-based security under real-time and energy constraints for the Internet of Things

JURY

Président :	**M. René LOZI**, Professeur des Universités, Université de Nice Sophia-Antipolis
Rapporteurs :	**M. Thomas GROSGES**, Professeur des Universités, Université de Technologie de Troyes
	Mme Samia BOUZEFRANE, Maître de conférences, HDR, Conservatoire National des Arts et Métiers (CNAM Paris)
Invités :	**M. Thang HOANG MANH**, Maître de conférences, Hanoi University of Science and Technology
	M. Marc BIDAN, Professeur des Universités, Université de Nantes
Directeur de thèse :	**M. Safwan EL ASSAD**, Maître de Conférences, HDR, Université de Nantes
Co-directrice de thèse :	**Mme Maryline CHETTO**, Professeur des Universités, Université de Nantes

Contents

List of Tables

List of Figures

Acknowledgements

First of all thanks to Allah Almighty for giving me the strength to complete the thesis work and making my dream come true. I have for so many years dreamt of the day when I would acquire my PhD and the day has finally come. This thesis could never have been completed without the guidance, help and support of many people.

I would like to first thank the members of my dissertation committee - not only for their time and extreme patience, but for their intellectual contributions to my development as a researcher.

I would like to express my sincere gratitude to my supervisor Pr. Safwan EL ASSAD, for all his guidance, support whenever I needed it, patience and encouragement.

My sincere thanks also goes to my co-supervisor Pr. Maryline CHETTO for her insightful comments, academic and social support and encouragement.

I would like also to thank IETR lab directors who provided me an opportunity to join their team as intern, and who gave access to the laboratory and research facilities. Without they precious support it would not be possible to conduct this research.

I gratefully acknowledge the funding received towards my PhD from the Erasmus Program - EGOVTN.

I would like to thank deeply my parents, who have helped, guided and supported me spiritually all my life. My deepest gratitude goes to my wonderful sister and brother, for all the love, prayers and best wishes during my studies.

Last, but not least, I would like to thank all my family, friends, both in France and in my country Tunisia, for their prayers and support, especially during the difficult times of the PhD journey.

Thank you all...

1

Introduction

1.1 Preface

Nowadays, mobile and embedded devices have become ubiquitous. This is due to recent advanced technologies in communications and computer science. We find numerous applications in various fields such as digital electronics, telecommunications, computing networks, smart cards, satellite systems, military defence system equipments, research system equipments, and so on [130, 33]. These devices are interconnected either locally or over the internet. This phenomenon is called the *Internet of Things* (*IoT*) [86, 82]. The term of *IoT* was firstly introduced, as a title of a presentation made at Procter & Gamble (P&G) in 1999 by Kevin Ashton, who was laying the groundwork for what would become the IoT [30]. The core idea of this concept lies in the presence of everyday physical objects known as things which are connected to the internet. Interconnection is ensured by technologies such as Radio Frequency IDentification (RFID) [274], Wireless Sensor Networking (WSN) [65], cloud servicing, machine-to-machine interfacing (M2M) [107], etc. The IoT is currently emerging: 50 billion devices are estimated to be wirelessly connected to the Internet by 2020 [82].

The vast majority of devices that will integrate the *IoT* are expected to work under severe constrained resources such as limited battery and computing power (e.g., running on tiny batteries) as well as little memory [231]; These constraints often exacerbate each other. The term *constrained device* was introduced in 2014 by Bormann, et al. [51] to define a class of connected devices with strict resource restrictions in comparison with common desktop computers, such as limited computation power (MegaFLOPS vs. TeraFLOPS), less memory (KiloBytes vs. GigaBytes) and significantly reduced power consumption (mWatt vs. Watt): a typical node has a 8 MHz micro-controller with less than 128 KB of instructions memory and approximately 10 KB of RAM memory [218, 260]. Moreover, micro-controllers that are used for constrained devices typically provide a limited set of features, e.g, they are not commonly equipped with memory management units (MMU), which in fact prevent using operating systems such as Linux on such devices [197].

The mass deployment of pervasive devices provides on the one hand several benefits such as lower logistic costs, optimized supply-chains, higher process granularity, etc. On the other hand, many applications are very security sensitive (e.g., defence, military, financial, automotive or aerospace applications), not to mention ones which require a baseline of privacy. The communication technique among a large number of constrained devices that generate huge amount of data has an impact on security and privacy of applications. Such devices have to be invulnerable to malicious attempts of communication jamming which can limit their functionalities. Furthermore, pervasive devices have to include protection strategies against physical attacks. Consequently, it is necessary to increase the security of data to be transmitted in order to avoid hacking of informations and fraud and to not outweigh any of IoT benefits.

To provide security foundations, such as data confidentiality, data integrity and authentication, one solution is to use appropriate cryptographic algorithms.

Cryptography is the art and science that concerns the transformation of informations so that it is not possible to other people different from the legitimate source and destination to access these informations while it is stored or transferred over insecure networks; This can be achieved by designing crypto-systems [117, 165]. A crypto-system is a cryptographic algorithm that depends on certain parameters and initial conditions called *secret key*. Cryptography algorithms are categorized into two main categories: symmetric crypto-system and asymmetric crypto-system [96, 242, 132]. An asymmetric crypto-system uses a public key and a private key, to encrypt/decrypt a message. On the other hand, a symmetric crypto-system uses only one key to encrypt and decrypt messages, which should be distributed before transmission of the emitter and the receiver. A symmetric key primitive can be further divided into two main categories: block ciphers and stream ciphers [165, 177]. A block cipher encrypts a fixed-length n-bits of data, - known as a

block - at one time (typically equal to 64-256 bits). Stream cipher encrypts one bit or byte or a group of bytes at a time. It is based on generating an "infinite" cryptographic keystream, and using that to encrypt data. To remain secure, the keystream should be unpredictable and should be never used.

In the literature, a growing number of cryptographic techniques to secure transmitted information have been developed [234, 161, 61, 64, 170]. In recent years, chaotic cryptography has received much attention. Chaos in cryptography was introduced by Matthews in 1990s [173]. Since then, investigation on chaotic image encryption has become an active field of research due to the interesting properties of chaos such as ergodicity, sensitivity to initial conditions and parameters of the system, similarity to random behaviour, and broad-band power spectrum [127]. Many chaotic image encryption methods have been proposed in the litterature [97, 142, 175, 115, 186]. Most of them can not be applied directly in constrained devices, because their design goals focus in providing high levels of security and do not meet the specific limitations offered by such innovative technologies and capabilities devices of the IoT [205, 204]. Hence, the main challenge is to design and develop new efficient cryptographic techniques capable of guaranteeing secure data transmission and providing an optimized security/cost/performance trade-off. The implementation of cryptographic primitives under such limited resource availability represents the research field of *LightWeight Cryptography* (LWC) [74]. LWC focuses in designing primitives for constrained devices with very limited resources in terms of memory, computing power, and battery supply [20, 202].

Stream ciphers are characterized by their simplicity and high speed compared to block ciphers since they work on a few bits at a time and have relatively low memory requirements. In the other hand, block ciphers work on larger block of data and often have feedbacks from previous ones. Stream ciphers are more appropriate in some applications (e.g. some telecommunications applications) where the amount of data is either unknown, or when data must be individually processed as it is received or when buffering is limited. Also, they are less susceptible to noise in transmission since bytes are individually encrypted with no relation with other blocks of data. Contrary to block ciphers where in most ciphers modes, bytes are encrypted using previous encrypted bytes. Consequently, if one part of the data is modified, all the rest is probably unrecoverable. Due to all these reasons, stream ciphers are well suited to constrained devices. This explains the recent evolution researchers efforts in the field of lightweight stream ciphers.

Certainly, security is an important key issue for the IoT applications which is well treated by many research works. In the same context, IoT will be confronted with other severe challenges among them, producing correct output at the correct time. In some real-time applications, real time performance becomes critical. The correctness of a real-time system depends not only on the correctness of the logical result of the computation but also on the physical time when this result is produced. The role of an operating system in communication devices is important.

Embedded systems require a Real-time Operating System (RTOS) that has real time capabilities in terms of scheduling and synchronization. Using such RTOS enables the applications to guarantee timing constraints on response times, even in desktop computers. Such a system can take in charge the analysis of an application ensuring that the input is given from a real time system at a predetermined interval of time. Then, the cryptographic system will permit both the checking of the validity of the key as well as the time interval at which the key is presented to the system.

1.2 Motivations and objectives

Using cryptographic techniques provides many of the security services required by the pervasive applications such that protecting data transmission against attacks. Yet, existing cryptographic techniques developed for enterprise and desktop computing might not satisfy embedded application requirements as

they can be huge and have a high energy consumption. In fact, smart devices of the IoT, including sensors, are inherently resource constrained with regard to memory, communication band-width, processing power and energy availability. Hence, a challenging topic concerns the design of efficient and lightweight cryptographic techniques to guarantee secure data transmission in the IoT. Such techniques should fit the low energy, computation and memory capabilities of cyber-physical systems. Nonetheless, they should also provide an optimized security/cost/performance trade-off.

The purpose of this thesis is to study the problem of information security under real-time and energy constraints and to design new chaos-based crypto-systems that answer these challenges. This research work, first, will focus on designing, implementing and analysing three pseudo-chaotic number generators (PCNGs). These generators use basic chaotic maps, a weak coupling matrix or a high diffusion binary coupling matrix, and a chaotic multiplexing technique. Then, three secure chaotic stream ciphers based on the proposed PCNGs are realized. The cryptographic analysis of the chaotic systems realized shows their robustness against known attacks. The performance obtained in computational complexity indicates their uses in real-time applications. We integrated these chaotic crypto-systems through a real-time operating system called *Xenomai* [14]. We comparatively measured energy, power and processing time consumption of the three proposed chaotic systems. We showed how to adapt the degree of security of these systems according to the time energy availability.

1.3 Thesis Outline and Contributions

This thesis is organized as follows:

Chapter 2 is dedicated to explaining the fundamental concepts of cryptography primitives. We start by discussing principles of foundation and basic concepts of cryptography and the two major categories of modern cryptographic primitives, namely symmetric and asymmetric algorithms. We present block ciphers and stream ciphers. After that, we introduce chaos theory and briefly present some chaotic maps including Gauss map, Tent map, Hénon map, Lozi map, Lorenz attractor and Rössler attractor. Then, we provide the state of the art of block ciphers, pseudo-random number generators and stream ciphers based on chaotic maps.

Chapter 3 presents a security and computing performance study of some discrete chaotic maps including: Logistic, Skew Tent and PWLCM maps, as base of proposed chaos-based stream ciphers during this thesis. First, we present a collection of common and standard security tools useful to define that assessment. Second, we discretize the chaotic maps making them running over a finite precision (N=32), and we analyze their cryptographic properties and speed. Then, we introduce a perturbation technique which permits to decrease the degradation caused by the discretizing process. We perform some security analysis of chaotic maps using this perturbation technique. In order to improve the cryptographic performance of chaotic maps, we propose a recursive structure. Then, we give the security and speed performance of chaotic maps using the perturbation technique and the recursive structure.

Chapter 4 presents our first contribution. It consists of designing and implementing, in an efficient and secure manner, three stream ciphers based on three proposed robust Pseudo-Chaotic Numbers Generators (PCNGs). We describe in details the general structure of the three proposed PCNGs. The first proposed PCNG, called CM-PCNG, uses three weakly coupled chaotic maps: PWLCM, Skew Tent and Logistic and includes a multiplexing chaotic technique. In comparison with the architecture of CM-PCNG, the second PCNG - DM-PCNG - uses a binary diffusion matrix as a coupling technique. The architecture of the third proposed PCNG, named CS-PCNG, is based on using two chaotic maps, namely PWLCM and SkewTent, and includes coupling and swap chaotic techniques. We give the security and statistical analysis,

and the computing performance measures of the proposed PCNGs and the corresponding stream ciphers. The proposed crypto-systems are very secure, due to the use of chaotic coupling, swap and multiplexing techniques, while they offer a high speed performance.

Chapter 5 first focuses on studying the performance of two proposed chaotic stream ciphers CM-SC & CS-SC in terms of energy and power consumption and memory assessment. We show that the proposed stream ciphers are lightweight crypto-systems. Compared to other crypto-systems presented in the literature, we demonstrate that our designed stream ciphers are suitable for practical secure applications of the IoT in a constrained resources environment. The second part of this chapter concerns the integration of the proposed crypto-systems with real-time features. We show how to implement a crypto-system in the framework of a real time application which is managed by a well known free open-source real-time operating system, Xenomai. And we present the results of our experiment, giving execution time measures of the two proposed real time crypto-systems.

Chapter 6 concludes the manuscript. We report a summary of the main new ideas and contributions that were brought by our work in the domain of real-time cryptography. Finally, we present a short list of open problems and future research issues.

2

State of the art

2.1 Introduction

Over the last several years, there has been tremendous interest world-wide in the possibility of using chaos in numerous fields, such as electronic systems, fluid dynamics, lasers, weather and communication systems [109, 287]. Chaos theory studies the behaviour of complex dynamic systems which have high sensitivity to small change in their parameters and makes the generated results entirely "unpredictable". The idea of using chaos theory in the cryptography field to enrich the design of new ciphers, has attracted more and more attention. Many fundamental characteristics of chaos, such as the ergodicity, deterministic nature, unpredictability, random-look nature and its sensitivity to initial conditions, can be connected with the "confusion" and "diffusion" property in cryptography [167, 127]. Chaotic systems have potential applications in such cryptography algorithms as block cipher, stream cipher and pseudo random number generator.

Some proposed cryptographic algorithms are not suitable for constrained devices or *pervasive* devices in the Internet of Things (IoT), including RFID (Radio Frequency Identification) tags, Wireless Sensors and mobile phones. Such devices are inherently resource constrained with regard to memory, communication band-width, processing power and energy [100]. Therefore, due to its low computation capabilities, there is a need to build a *lightweight* security cipher that can fit these devices. The design of lightweight cryptographic algorithm is always a great challenge that the designer needs to cope with the trade-off between achieving robust security with low cost and enhanced performance [74].

We dedicate this chapter to explain first the fundamental concepts of cryptography primitives. We start by discussing principles of foundation and basic concepts of cryptography and the two major categories of modern cryptographic primitives, namely symmetric and asymmetric algorithms. We discuss in detail block ciphers and stream ciphers. Furthermore, we focus on chaos theory and briefly introduce some chaotic maps including Gauss map, Tent map, Hénon map, Lozi map, Lorenz attractor and Rössler attractor. Finally, we will provide a review of block ciphers, pseudo-random number generators and stream ciphers based on chaotic maps.

2.2 Cryptography: foundation and basic concepts

Almost since the beginning of the writing language, it was necessary to find ways to hide valuable information [243]. Cryptography is the science that concerns the transformation of information so that it is not possible to other people different from the legitimate source and destination to access this information. The Cryptology process requires two different and complementary stages. The first step is *cryptography* which presents selection of the tools and the framework which guide the concealing of the information. The second one is *cryptanalysis*. It means the evaluation of the transformation system. The word 'Cryptography' has a Greek etymology. It is derived from *krýpto* "hidden" and the verb *gráfo* "to write". Cryptography is the process of converting a message (or plaintext) into unintelligible form or ciphertext and viceversa [117]. The transformation of the plaintext into the ciphertext is called *encryption*, while the inverse process is named *decryption*. The algorithm used for performing encryption is named the *cipher*. The document history of 'Cryptography' begins with ancient Egyptian hieroglyphic. It has predominantly been used by the governments and military for the confidentiality of information. The modern cryptography begins with the Shannon theory [235], in which three fundamental goals must be achieved:

— *Confidentiality:* It is roughly equivalent to privacy. It ensures that information is not made available or disclosed to an adversary who has access to a communication channel and he is not able to derive messages exchanged by the emitter and the receiver;

— *Integrity:* It is the assurance that the information is trustworthy and accurate. It ensures that an adversary who has access to a communication channel is not able to change and modify the content of messages exchanged by the emitter and the receiver;

— *Availability:* It ensures that information are available to authorized people when it is needed.

There are two major categories of modern cryptographic primitives, namely symmetric and asymmetric algorithms [96, 242, 132]. The main distinguishing property of these categories is the different usage of the

key.

Symmetric encryption algorithms use only one key for both encryption and decryption, which should be distributed before transmission to the emitter and the receiver. Figure 2.1 gives an overview on the symmetric encryption primitive. Symmetric ciphers are based on a combination of mathematics and cryptographic principles that usually call for simple primitives such as rotation, substitution, permutation, shift, bit-wise XOR etc. The key plays an important role in the encryption/decryption process. Its effectiveness directly depends on the size of key. An adversary may access and decrypt message if a weak key is used in the encryption algorithm. The strength of symmetric encryption algorithm depends also on the size of the used key. For the same algorithm, encryption is more robust and harder to break when using longer key than the one performed using smaller key. A main problem with these categories is securing the key transmission over the malicious network.

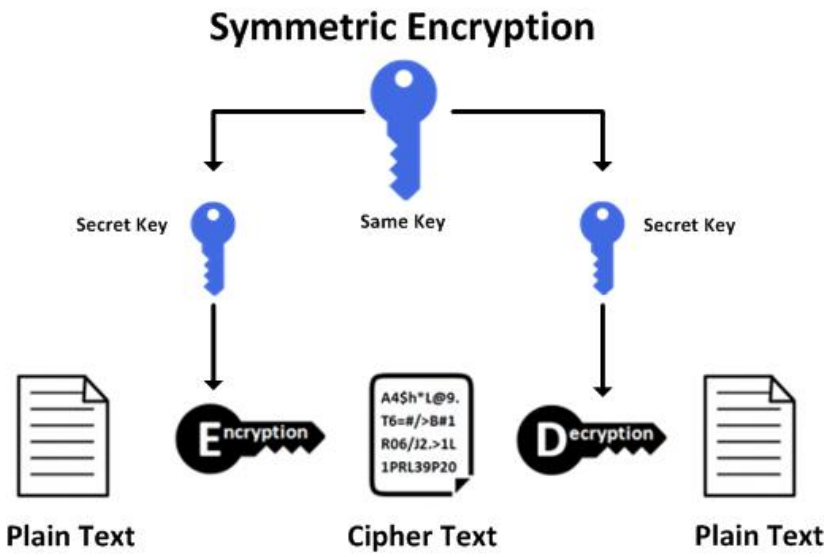


Figure 2.1 – Symmetric encryption primitive.

Asymmetric encryption algorithm is used to solve the problem of key distribution. In asymmetric algorithms, each participant possesses a pair of keys: a public key and a private key. The public key is known by all the public while the private key is known only by the user. There is no need to distribute them before transmission. The two keys are strongly related to each other and each has its own purpose: The public key is used for encryption, whereas the private key is used for decryption. We give in Fig.2.2 the principle of the asymmetric encryption. The public key encryption is based on mathematical primitives so that they are computationally intensive. Asymmetric encryption relies on mathematical functions which are computationally intensive such as modular addition, subtraction, modular multiplication, variable length rotations, etc. It is almost 1000 times slower than symmetric ones. This makes asymmetric encryption not well-suited to most of wireless sensors of the IoT which are low-cost computing devices [62, 105, 247].

In this thesis, we focus on symmetric cryptography and we introduce in next section its essential principles.

2.2.1 Symmetric encryption algorithms

There are two main kinds of modern symmetric encryption algorithms: block ciphers and stream ciphers. In the following, we describe these types in more details. In 1883, Kerckhoffs proposed six principles of designing practical symmetric encryption algorithms [121]. Among these, the most important and relevant for modern ciphers, known as *Kerckhoff's principles*, is the principle that the security of a cipher should not depend on keeping secret the cipher, but only the key must be secret. Kerckhoffs' principle was later restated Shannon as "the enemy knows the system being used", i.e., "one ought to design systems

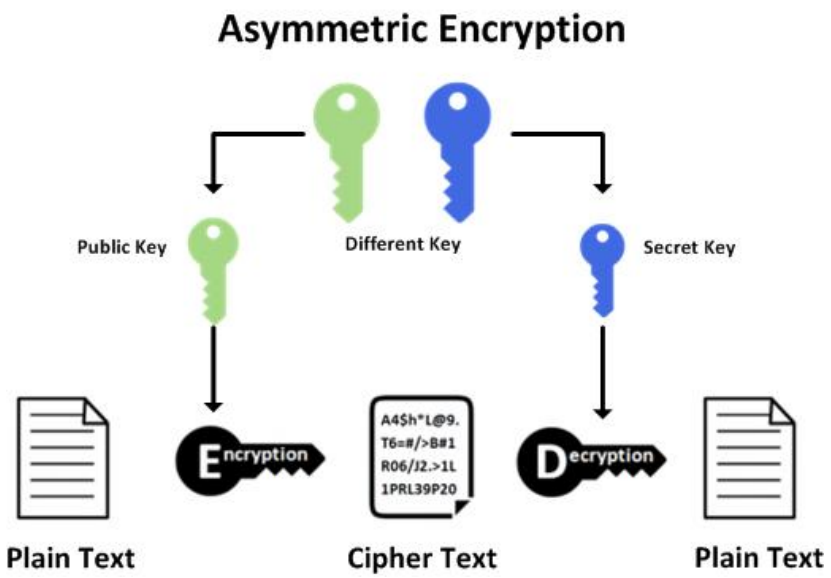


Figure 2.2 – Asymmetric encryption primitive.

under the assumption that the enemy will immediately gain full familiarity with them". In that form, it is called Shannon's maxim [236]. The idea is that the security of the cipher rests only in the keys and not in the algorithm. This presents one of the main concepts of modern cryptography. In fact, in cryptography, the key can be compromised and replaced with different one, without requirement of redesign cipher. This concept applies to the cipher security in general. We apply this principle when designing our proposed crypto-systems. We now briefly describe the above listed types of symmetric encryption algorithms.

2.2.2 Block ciphers

A block cipher is a deterministic algorithm which maps fixed-length n-bits of plaintext, called a *block*, to n-bits ciphertext blocks; n is called the *blocklength* (typically equal to 64-256 bits). Block cipher is an invertible transformation that takes as inputs the secret key K and n-bits plaintext and outputs n-bits ciphertext. It consists of two paired algorithms, one for encryption (E), and the other for decryption, (D) [69]. The decryption algorithm presents the inverse function of encryption, i.e., $D = E^{-1}$ [177]. The encryption cipher is given by the following equation:

$$E_K(P) := E(K, P) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n. \tag{2.1}$$

Where K is a secret key of length k bits and P is plaintext of length n. $E_K(P)$ returns a ciphertext C of n bits. For each K, the function $E_K(P)$ is required to be invertible mapping on $\{0,1\}^n$. The inverse for $E_K(P)$ which presents the decryption function is defined as:

$$E_K^{-1}(P) := D_K(C) := D(K, C) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n. \tag{2.2}$$

$D_K(C)$ takes a key K and a ciphertext C to return a plaintext P, such that $\forall K : D_K(E_K(P)) = P$.

In most contemporary block ciphers, the *blocklength* is at least equal to 128 bits. In order to encrypt a long plaintext P, it must first be partitionned into separate blocks, each one is small enough to be input to a block cipher. In the simplest case, called the Electronic CodeBook (ECB) mode, the plaintext is split into separate blocks and then each one is encrypted and decrypted independently. However, this native mode is generally not secure because of if plaintext block p_1,p_2,..., are encrypted twice under the same key, or equal plaintext blocks are encrypted, the same output ciphertext will be produced. Therefore, patterns in the plaintext become evident in the ciphertext output evidently. To overcome this disadvantage, several modes of employing block ciphers (so-called modes of operation) have been designed. The four most common

modes are CBC, CFB, OFB and CTR [177]. The CBC mode is used in block ciphers, the other modes (CFB, OFB and CTR) are used in stream ciphers. The general concept is to use an additional random input value, called an initial vector IV to create probabilistic encryption [40].

Many block cipher algorithms have been proposed in the literature [211, 213, 70, 230, 172, 229]. The most and best-known proposed algorithms of block ciphers are the DES and the AES. The DES was developed in the early 1970s, by IBM as a symmetric-key algorithm for the encryption of electronic data. It was selected by the National Security Agency (NSA) an official Federal Information Processing Standard (FIPS) for the United States in 1977 and became a standard for most communication protocols [250]. By the mid '90s, the DES considered to be not secure for many applications due to its short 56-bit key size. And it has been superseded by the Advanced Encryption Standard (AES) in 2001 as the US standard.

The AES algorithm is the most widely used symmetric cipher today in several industries and in many commercial systems. To date, there are no classical cryptanalysis better than brute-force attack against AES algorithm. However, the AES algorithm is vulnerable against Implementation attacks. This standard specifies the Rijndael algorithm [3], [4], [5], [6], a symmetric block cipher that can process data blocks of 128 bits, using secret keys with lengths of 128, 192, and 256 bits, and may be referred to as "AES-128", "AES-192", and "AES-256" (see Figure 2.3).

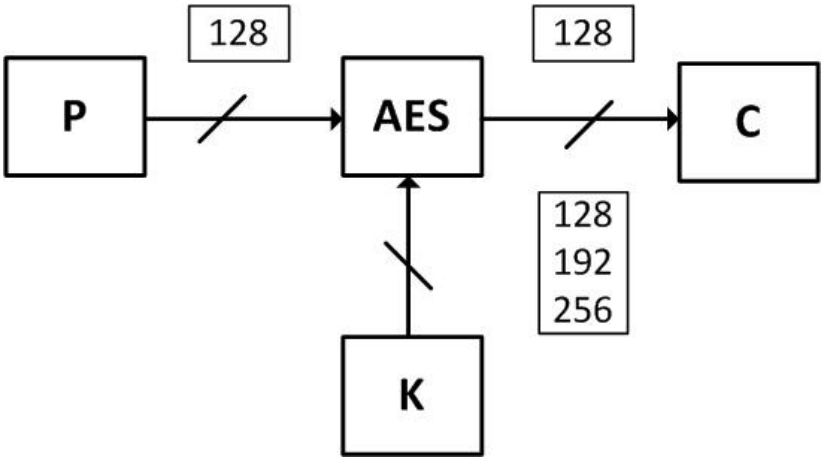


Figure 2.3 – AES input/output parameters.

AES is an iterated cipher; the number of rounds, which denoted by Nr, depends on the key length according to Table 2.1.

Key lengths	Number of rounds Nr
128 bits	10
192 bits	12
256 bits	14

Table 2.1 – Key lengths and number of rounds for AES.

Internally, the operations in the AES algorithm are performed on a two-dimensional array of bytes called the *State*. The state consists of four rows of bytes, each containing Nb bytes, where Nb = 128/32 = 4. All operation in AES are byte-oriented operations, and all variables used are considered to be formed from an appropriate number of bytes. The state is referred to as either $s_{r,c}$ or s[r, c], where r and c are the row number and the column number, with: $0 \leq r < 4$ and $0 \leq c < 4$.

At the start of the cipher or inverse cipher, the input (plain text) - the array of bytes in_0, in_1, ..., in_{15} is copied into the state array. The cipher or inverse cipher operations are then conducted on this state array, after which its final value (cipher text) is copied to the output - the array of bytes out_0, out_1, ..., out_{15}.

The input array is copied to the state array according to:

$$s[r, c] = in[r + 4c] \tag{2.3}$$

for $0 \leq r < 4$ and $0 \leq c < 4$.

At the end of the cipher or inverse cipher, the state is copied to the output array as follows:

$$out[r + 4c] = s[r, c] \tag{2.4}$$

for $0 \leq r < 4$ and $0 \leq c < 4$.

2.2.3 Stream ciphers

Stream ciphers are an important class of symmetric encryption algorithms. They encrypt, with a time-varying transformation, individual data (usually binary digits) of plaintext one at a time, contrary to block ciphers (section 2.2.2) which simultaneously operate with a fixed transformation on large groups of plaintext; thus block ciphers are *memoryless*. Stream ciphers are generally characterized by their hight speed compared to block ciphers in hardware, and less complex hardware implementation. Also, they are more appropriate in some applications (e.g. some telecommunications applications) when data must be individually processed as it is received or when buffering is limited. Moreover, synchronous stream ciphers (described in later) are more suitable in some cases where transmission errors are highly possible, since they are not affected by error-propagation.

Stream ciphers process plaintext in single bit and the cipher function may change as plaintext is processed. This is achieved by combining, bit by bit, the plaintext and a keystream sequence - also called running-key - to obtain the cipher text. The keystream is generally generated by a finite state function called the keystream generator or the running-key generator. Stream ciphers can be either synchronous or asynchronous stream ciphers [176, 223].

A synchronous stream cipher (see Fig. 2.4) is one in which the keystream is produced independently of the plaintext message and of the ciphertext. It depends only on the secret key. The OFB mode of a block cipher is an example of a synchronous stream cipher. The sender and the receiver must be synchronized. They used the same secret key to obtain proper decryption. If the decryption fails due to lost of synchronization or ciphertext bits are inserted or deleted during transmission, then decryption process can be only restored through additional re-synchronization techniques. In addition, a cipher text bit which is changed (but not deleted) does not affect the decryption of other bits of ciphertext.

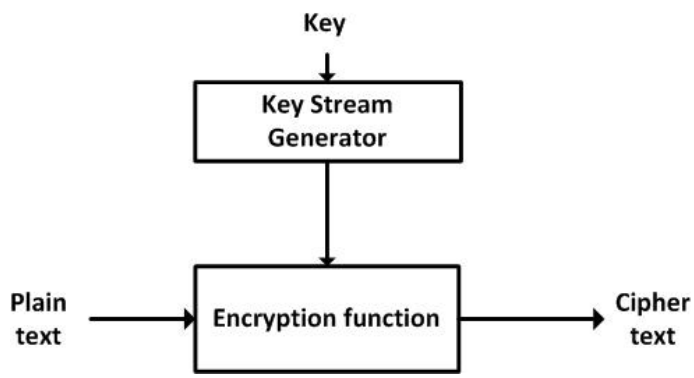


Figure 2.4 – Encryption process of a synchronous stream cipher.

In contrast, an asynchronous stream cipher or self-synchronizing stream cipher is one where the keystream also depends on the ciphertext. The keystream is produced as a function of the key and a fixed number of previous ciphertext bits. The most common example of this is provided by some block cipher in what is termed cipher-feedback (CFB) mode. The encryption process of an asynchronous stream cipher is described in Figure 2.5.

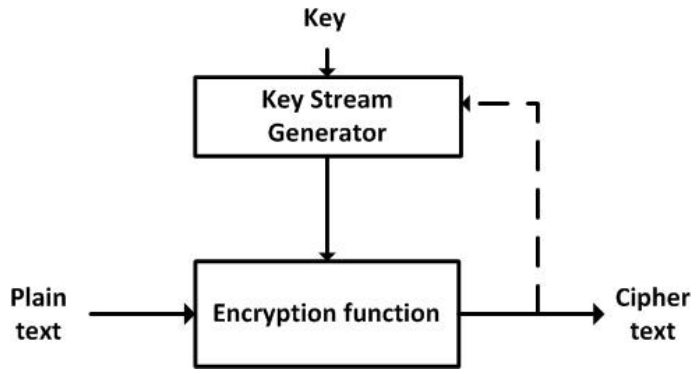


Figure 2.5 – Encryption process of an asynchronous stream cipher.

Suppose that the asynchronous stream cipher depends on l previous ciphertext bits. If one ciphertext bit is modified or deleted during transmission, then decryption of up to l ciphertext bits can be incorrect.

The most famous stream cipher that has been proposed to date in the literature is the Vernam cipher, also called One-Time Pad (OTP). It is a synchronous stream cipher in which the plaintext is combined with a random "keystream" of the same length, used only once, to generate the ciphertext, using the Boolean "exclusive or" (XOR) function. The keystream is produced by a pseudo-random sequence generator, having as an input a secret shared key K. The XOR operation is symbolised by $\bigoplus$. Figure 2.6 presents the encryption process using an OTP stream cipher. The OTP stream cipher is unconditionally secure [215].

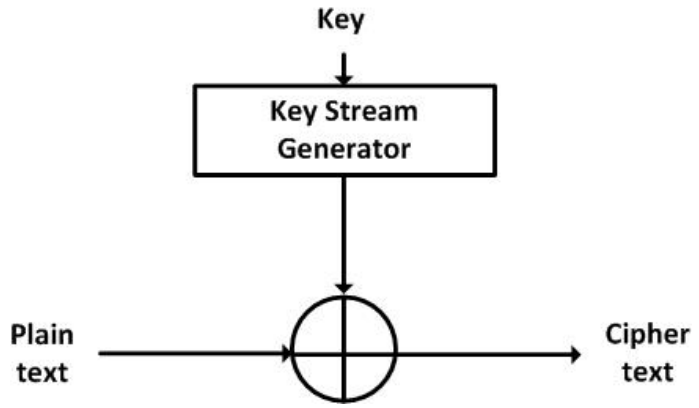


Figure 2.6 – Encryption process of OTP stream cipher.

In recent years, several research efforts have investigated secure stream cipher designs. Many of these have been proposed in software implementation.

The **RC4** stream cipher (also known as ARC4 or ARCFOUR meaning Alleged RC4,) [193, 212] was designed by Ron Rivest for RSA Data Security in 1987. It was the most commonly used in industrial applications, Internet protection using sing the SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols... RC4 was characterized by its simple implementation. It produces a pseudo-random stream of bits (the keystream). The cipher is based on using a random permutation technique, which is initialized with a variable-length key of from 1 to 256 bytes to initialize a 256-byte state vector S, using the key-scheduling algorithm (KSA). Once these operations were carried out and have been complete, the keystream is produced using the pseudo-random generation algorithm (PRGA), by outputting some values of the S permutation updated at each clock. Several papers have been published analyzing the security of RC4 [125, 179, 88, 124]. Multiple vulnerabilities have been discovered which rend it insecure [201], especially when the beginning of the generated keystream is not discarded, or when non-random or related secret keys are used [120]. In 2014, Ronald L. Rivest reconsiders the design of the RC4, and proposes an improved variant - called Spritz - which attempts to repair weak design decisions in RC4 while keeping its original principles. Spritz uses a sponge (or sponge-like) function, which can discard bits of keystream

at any time and from which one can produce pseudo random output bytes of arbitrary length. Spritz can be used as an encryption algorithm, a message-authentication code generator or as a cryptographic hash function. But it is rather slow compared to other hash functions such as SHA-3 and best known hardware implementation of RC4 [214, 44].

AES-CTR algorithm is used as a stream cipher. In fact, it is the AES block cipher in Counter mode (CTR). The CTR mode was introduced by Whitfield Diffie and Martin Hellman in 1979 [149, 116]. The AES-CTR requires an initialization vector IV and the secret key of length 128, 192 or 256 bits. The same IV and key combination must not be used more than once. Many approaches are possible to IV generation that ensures uniqueness, including incrementing a counter for each packet and linear feedback shift registers (LFSRs). The AES-CTR cipher operation consists in ciphering a counter value which must be nonce, with the secret key and xoring the obtained keystream with the corresponding block of the plaintext. The counter corresponding with a IV value is then updated to cipher in "one-time pad" mode the next plaintext block. Figure 2.7 presents the principle of the encryption process of the AES-CTR stream cipher.

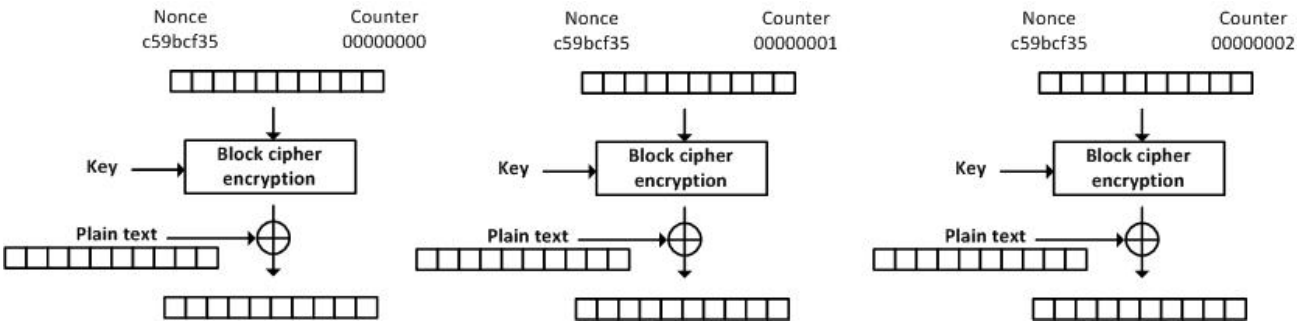


Figure 2.7 – The encryption process of the AES-CTR stream cipher.

The AES-CTR mode is one of the best known modes of the AES block cipher and recommended by Niels Ferguson and Bruce Schneier in [149]. The CTR mode has similar characteristics to the AES algorithm which has been standardized by the NIST. Thus, we say that the AES-CTR is secure. The AES-CTR model is fully parallelizable and enables effective utilization of many architectural features of modern processors including aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instruction.

A5/1 is a stream cipher used in most European country in order to provide communication privacy in the Global System for Mobile Communications (GSM) standards. A5/1 was developed in 1987. It was initially kept secret by the GSM companies, but it was entirely reverse engineered and published in 1999 by Marc Briceno from an actual GSM telephone [55, 54]. A5/1 is based on irregular clocking of three short Linear Feedback Shift Registers (LFSR) of lengths 19, 22 and 23, denoted by R1, R2, R3, respectively. The key size is equal to 64 bits and the keystream is produced by xoring the output from the three registers. A GSM transmission is organised as sequences of frames. A new session key K is used for each conversation. One frame is sent every 4.165 milliseconds and contains 114 bits representing the communication between the emitter and the receiver. For each frame, A5/1 is used to produce 114 bits of keystream which is xored with 114 bits of plaintext to produce the cipher text. Figure 2.8 presents the general structure of the A5/1 stream cipher. The three LFSRs are updated according to their primitive feedback polynomials which are summarized in Table 2.2. The output of each LFSR is the last bit. The clocking function is based on stop/go technique using a majority rule. The three bits are being examined and their majority is calculated. The register is clocked if the clocking bit agrees with the majority bit. Hence, note that at each step at least two or three LFSRs are clocked and the probability for each LFSR being clocked is equal to 3/4. Initially, all the LFSRs are set to zero. Then for 64 cycles, the 64-bits key are consecutively combined in parallel to the least significant bit of each LFSR using XOR operation: $R[0] = R[0] \oplus K[i]$. Each LFSR is then clocked. Similarly, the three LFSRs are clocked 22 times and the entire system is clocked for 100 additional clock cycles using the irregular clocking mechanism, but the output discarded. Then, finally, the three LFSRs are

clocked for 228 clock cycles, producing two 114 bit sequences of output keystream, first 114 for downlink, last 114 for uplink.

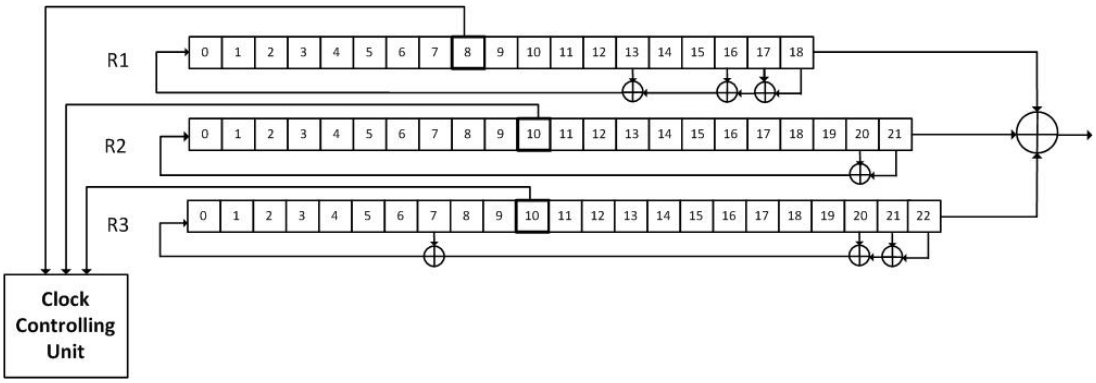


Figure 2.8 – The A5/1 stream cipher structure.

Register	Lengths (bits)	Feedback Polynomial	Clocking Bit	Bits that are Xored
R1	19	$x^{19} + x^5 + x + 1$	8	18,17,16,13
R2	22	$x^{22} + x + 1$	10	20,21
R3	23	$x^{23} + x^{15} + x^2 + x + 1$	10	22,21,20,7

Table 2.2 – Parameters of the A5/1 Registers.

Several attacks on the A5/1 have been published [46, 45, 76] and serious weaknesses have been identified.

In 2004, a project under the Information Societies Technology (IST) Program of the European Network of Excellence for Cryptology (ECRYPT), called "eStream" was tasked with seeking a strong stream cipher [216, 3].

Its goal was to give rise to a standardization of fast and secure stream ciphers. Thirty-four candidate ciphers were submitted. Only a few proposals were chosen to belong to the current official "eStream" project and the others were rejected because of security vulnerabilities or lower overall performance. Currently, none of these ciphers have been used in a widespread application, but all show advanced developments in the state of the art of stream cipher design. Two profiles of stream ciphers for software and hardware implementations were defined in the eStream project. The first profile is oriented to software-ciphers with high throughput and is faster than the 128-bits AES-CTR. The second profile is oriented to hardware ciphers that are suitable for highly constrained environments and are more compact than the 80-bits AES. The finalist ciphers for the two profiles are given in Table 2.3. These ciphers were found to be secure against known attacks. However, some tangible results have been reported by newer cryptanalysis attempts for some of these ciphers (Rabbit, Salsa12, SOSEMANUK, Grain, Trivium and MICKEY2.0)[162]. We briefly describe below two examples of the finalist stream ciphers for the first profile oriented to software implementation namely Rabbit and HC-128.

Profiles	The finalist stream ciphers
First profile	Rabbit, HC-128, Salsa20/12, SOSEMANUK
second profile	Grain, Trivium, MICKEY 2.0

Table 2.3 – The finalist stream ciphers for the eStream project.

The **Rabbit** is a synchronous stream cipher developed by Martin Boesgaard, Mette Vesterager, Thomas Pedersen, Jesper Christiansen, and Ove Scavenius [49]. The design of Rabbit was inspired by the complex

behaviour of chaotic maps and their heigh sensitivity to small change which lead to that chaotic systems can be used for cryptographic purposes (more explanations in next sections). Rabbit works internally on a finite precision N bits equal to 32 which makes it suitable for software implementation. The Rabbit algorithm is based on iterating a set of coupled non-linear functions. It takes as input a 128-bit secret key and produces for each iteration an output sequence of 128 pseudo random bits using a combination of the internal state bits. The internal state consists of 513 bits divided between eight 32-bit state variables $x_{j,t}$, eight 32-bit counters $c_{j,t}$ ($0 \leq j < 8$ and t denotes the number of iterations) and one counter carry bit $\phi_{7,t}$, which needs to be stored between iterations. Initially, $\phi_{7,t}$ is set to zero and the eight $x_{j,t}$ and $c_{j,t}$ are derived from the key at initialization using a next-state function.

The core of the Rabbit algorithm is the iteration of the system with the next-state function defined by the following equations:

$$\begin{aligned}
x_{0,t+1} &= g_{0,t} + (g_{7,t} <<< 16) + (g_{6,t} <<< 16) \\
x_{1,t+1} &= g_{1,t} + (g_{0,t} <<< 8) + g_{7,t} \\
x_{2,t+1} &= g_{2,t} + (g_{1,t} <<< 16) + (g_{0,t} <<< 16) \\
x_{3,t+1} &= g_{3,t} + (g_{2,t} <<< 8) + g_{1,t} \\
x_{4,t+1} &= g_{4,t} + (g_{3,t} <<< 16) + (g_{2,t} <<< 16) \\
x_{5,t+1} &= g_{5,t} + (g_{4,t} <<< 8) + g_{3,t} \\
x_{6,t+1} &= g_{6,t} + (g_{5,t} <<< 16) + (g_{4,t} <<< 16) \\
x_{7,t+1} &= g_{7,t} + (g_{6,t} <<< 8) + g_{5,t}
\end{aligned} \tag{2.5}$$

$$x_{j,t} = (x_{j,t} + c_{j,t+1})^2 \oplus [(x_{j,t} + c_{j,t+1})^2 >> 32] mod 2^{32} \tag{2.6}$$

Where $<<<$ and $>>$ denote left bit-wise rotation and right logical bit-wise shift respectively. All additions are modulo 2^{32}. Figure 2.9 illustrates schematically the next-state function.

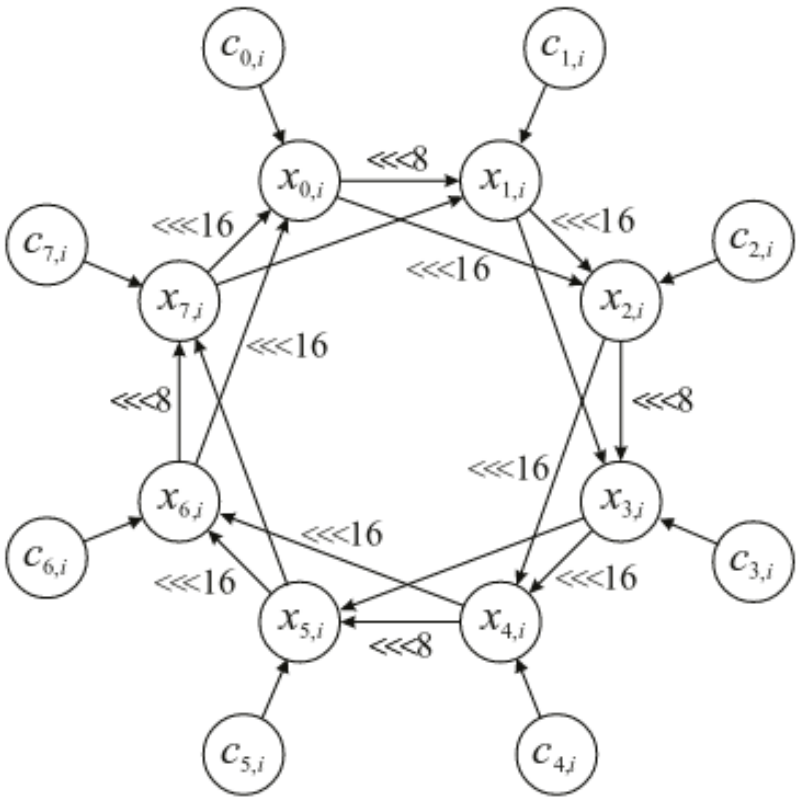


Figure 2.9 – The next-state function of the Rabbit stream cipher.

The counters are incremented as follows:

$$\begin{aligned} c_{0,t+1} &= c_{0,t} + a_0 + \phi_{7,t} mod\ 2^{32} \\ c_{1,t+1} &= c_{1,t} + a_1 + \phi_{0,t+1} mod\ 2^{32} \\ c_{2,t+1} &= c_{2,t} + a_2 + \phi_{1,t+1} mod\ 2^{32} \\ c_{3,t+1} &= c_{3,t} + a_3 + \phi_{2,t+1} mod\ 2^{32} \\ c_{4,t+1} &= c_{4,t} + a_4 + \phi_{3,t+1} mod\ 2^{32} \\ c_{5,t+1} &= c_{5,t} + a_5 + \phi_{4,t+1} mod\ 2^{32} \\ c_{6,t+1} &= c_{6,t} + a_6 + \phi_{5,t+1} mod\ 2^{32} \\ c_{7,t+1} &= c_{7,t} + a_7 + \phi_{6,t+1} mod\ 2^{32} \end{aligned} \tag{2.7}$$

where the carry $\phi_{7,t+1}$ is given by:

$$\phi_{j,t+1} = \begin{cases} 1 & \text{if} c_{0,t} + a_0 + \phi_{7,t} \geq 2^{32} \hat{} j = 0 \\ 1 & \text{if} c_{j,t} + a_j + \phi_{j-1,t+1} \geq 2^{32} \hat{} j = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

Furthermore, the a_j constants are equal to:

$$\begin{aligned} a_0 = a_3 = a_6 &= Ox4D34D34D, \\ a_1 = a_4 = a_7 &= 0xD34D34D3 \\ a_2 = a_5 &= 0x34D34D34 \end{aligned} \tag{2.9}$$

After each iteration, 128 bits of keystream are generated according to an extraction technique. Once the extraction function is completed, the extracted bits are xored with the plaintext/ciphertext for encryption/decryption.

Rabbit is a high speed stream cipher. Its simple design also helps in hardware implementation. The computing performance of Rabbit algorithm is much better than some modern stream ciphers but is far behind a stream cipher designed for hardware implementation such as Trivium. The cryptanalysis of Rabbit did not reveal any attacks against Rabbit. Only the existence of a non-null bias in the keystream generated by Rabbit is demonstrated [31]. The keystream bias is greater than $2^{-124.5}$ for certain bits, and this leads to a distinguisher requiring about 2^{247} 128-bit samples of keystream derived from random keys, which remains much higher than the cost of exhaustive key search.

The **HC-128** is a synchronous stream cipher designed by Hongjun Wu [4, 194] and is currently a member of the eSTREAM software portfolio. The HC-128 design is suitable for modern super-scalar processors. It makes use of a 128-bit key K and 128-bit initialization vector IV. Its state contains two tables P and Q, each with 512 registers of length equal to 32 bits. At each step, a non-linear feedback function is used to update one register of one of the tables. All the elements of the two tables get updated every 1024 steps. A non-linear output filtering function generates a 32-bit keystream output word. The cipher specification states that a keystream with length up to 2^{64} bits can be generated from 128-bit key K and a 128-bit IV.

HC-128 is the simplified version of HC-256 which uses a 256-bit key and 256-bit IV [280]. There are six functions being used in HC-128: f_1(x), f_2(x), g_1(x), g_2(x), h_1(x) and h_2(x). P is used as a S-box in h_2 and Q is used in the same purpose for h_1. The used functions are described in Table 2.4 and 2.5 where x is a 32-bit word and x = $x_3 \parallel x_2 \parallel x_1 \parallel x_0$, x_0, x_1, x_2, and x_3 are four bytes. The bytes x_3 and x_0 respectively denote the most and least significant byte of x.

The generation process starts with the initialization step i.e. with the Key and IV setup algorithms: K and IV are expanded into the two table P and Q, and the cipher runs 1024 steps.The Key and IV setup function is described in Algorithm 1. Once the initialization step completes, the algorithm is ready to generate keystream.

symbol	Meaning
+	addition mod 2^{32}
-	subtraction
$\oplus$	xor operation
$\|$	concatenation
$\ll$ / $\gg$	left / right shift
$\lll$ / $\ggg$	left / right rotate
$\square$	subtraction mod 512

Table 2.4 – Symbols and their meaning used in HC-128 stream cipher.

Function	Description
f_1(x)	(x $\ggg$ 7) $\oplus$ (x $\ggg$ 18) $\oplus$ (x $\gg$ 3)
f_2(x)	(x $\ggg$ 17) $\oplus$ (x $\ggg$ 19) $\oplus$ (x $\gg$ 10)
g_1(x)	((x $\ggg$ 10) $\oplus$(z $\ggg$ 23)) + (y $\ggg$8)
g_2(x)	((x $\lll$ 10) $\oplus$(z $\lll$ 23)) + (y $\lll$ 8)
h_1(x)	Q[x_0] + Q[256+x_2]
h_2(x)	P[x_0] + P[256+x_2]

Table 2.5 – Functions used in HC-128 stream cipher and their description.

Algorithm 1 KEY-IV-SETUP

Step-1: Expanding Key & IV into an array W_i (0 $\leq$ i $\leq$1279)...
for i =0 $\rightarrow$ 7 **do**
  $W_i \leftarrow K_i$
end for
for i =8 $\rightarrow$ 15 **do**
  $W_i \leftarrow IV_{i-8}$
end for
for i =16 $\rightarrow$ 1279 **do**
  $W_i = f_2(W_{i-2}) + W_{i-7} + f_1(W_{i-15}) + W_{i-16} + i$
end for
Step-2: Update the tables P and Q with the array W
for i =0 $\rightarrow$ 511 **do**
  $P[i] \leftarrow W_{i+256}$
  $Q[i] \leftarrow W_{i+768}$
end for
Step-3: Run the cipher for 1024 steps and use the outputs to replace the table elements...
for i =0 $\rightarrow$ 511 **do**
  $P[i] = (P[i] + g_1(P[i\square 3], P[i\square 10], P[i\square 511])) \oplus h_1(p[i\square 12])$
  $Q[i] = (Q[i] + g_2(P[i\square 3], P[i\square 10], P[i\square 511])) \oplus h_2(p[i\square 12])$
end for

At each step, one element of a table is updated and one 32-bit output is generated. Each S-box is used to generate only 512 outputs, then it is updated in the next 512 steps. The keystream generation algorithm of HC-128 is given in Figure 2 [181].

Algorithm 2 KEYSTREAM-GENERATION

Assume N bits are required...
for i =0 $\rightarrow N$ **do**
  $j = imod512$
  if (i mod 1024) $\leq$ 512 **then**
    $P[j] \leftarrow P[j] + g_1(P[j\Box 3], P[j\Box 10], P[j\Box 511])$
    $s_i = h_1(P[j] \oplus P[j\Box 12])$
  else
    $Q[j] \leftarrow Q[j] + g_1(Q[j\Box 3], Q[j\Box 10], Q[j\Box 511])$
    $s_i = h_1(Q[j] \oplus Q[j\Box 12])$
  end if
  i $\leftarrow$ i+1
end for

Many other stream ciphers have been proposed in software form, e.g., LEVIATHAN (Cisco), MUGI (Hitachi-K.U. Leuven), SNOW [75], SOBER (Qualcomm) and [217]. These stream ciphers have proven to be very weak and insecure. This has incited researchers to search for new methodologies that are immune to many attacks that can be applied.

2.3 Chaos-based cryptography

2.3.1 Chaos Theory

The word chaos is derived from the ancient Greek *'xαos'*, which means unpredictable behaviour or a state without order. Chaos theory has been established since 1970s. It is a branch of mathematics that focused on the behaviour of dynamical systems. In chaos theory, a chaotic system is a simple, non-linear dynamic process that reflects completely unpredictable behaviour, and hence randomness. Moreover, it is a deterministic system and high sensitive to initial conditions, such that, if two identical chaotic systems are in two slightly different initial conditions, they will evolve toward amazingly different results [36, 287]. A system is called a chaotic system if it is high sensitive to initial conditions and parameters and if periodic orbits are dense [43]. Chaos theory has many applications in several disciplines, including meteorology, physics, computer science, engineering, economics, philosophy, and biology [109].

Chaos-based cryptography is the use of chaos theory in cryptographic systems. Since 1980s, the idea of using chaotic systems to design crypto-systems has attracted more and more attention. It can be traced to Shanon's classical paper on theory of secrecy systems [236]. The good dynamical properties of chaotic systems implies good cryptographical properties of crypto-systems. And, the basic method to make crypto-systems have good and strong cryptographical properties implies quasi-chaos.

Chaos theory and nonlinear dynamic have been used in the design of cryptographic primitives including image encryption algorithms, hash functions, secure pseudo-random number generators, block ciphers, stream ciphers, watermarking and steganography [18].

The chaotic cryptographic primitives are generally made by combination of two operations called confusion and diffusion, which are modelled well by chaos theory [27]. Both operations are repeatedly performed till the sufficient security level is achieved. The quality of security is tested by its capability to defend different attacks including known plaintext attack, statistical attack, deferential attack, and brute-force attack, etc.

Most of the cryptographic algorithms are based on using uni-modal chaotic maps, their control parameters and their initial conditions as their keys [38]. Many chaotic maps are proposed in the literature that have been applying to cryptography in several ways. In the following sections, we will give a brief introduction to some chaotic maps, and their applications in cryptography particularly proposed chaotic pseudo-random number generators, chaos-based block ciphers and chaos-based stream ciphers of the literature.

2.3.2 Chaotic maps

In mathematics, a chaotic map is a function which exhibits some sort of chaotic behaviour. It often takes the form of iterated function and occurs in the study of dynamical systems. Chaotic maps may be parametrized by a continuous-time or a discrete-time parameter.

According to Alligood et al., [21] a chaotic map is a function of its domain onto itself, the starting point of the trajectory (the sate from which the system starts) is called the initial condition. Chaotic maps clearly illustrate statements of many characteristics of chaotic behaviour such us sensitivity to initial conditions, complex behaviour and the evolution of information in deterministic and unpredictable behaviour [32]. Several chaotic maps with one-dimension (1-D), two-dimensions (2-D) and three-dimensions (3-D) are proposed in the literature. In this subsection, we will give a brief description to some chaotic maps including Gauss map, Tent map, Hénon map, Lozi map, Lorenz attractor and Rössler attractor. Other maps such as Logistic map, Skew Tent and Piecewise Linear Chaotic Map (PWLCM) will be described in Chapter 3 since they will be used in this thesis as base of the proposed pseudo-chaotic number generators.

Gauss map The Gauss map (also known as Gaussian map), is a 1-D non-linear iterated map of the reals into a real interval given by the Gaussian function:

$$x_{n+1} = exp(-\alpha x_n^2) + \beta \tag{2.10}$$

where α and β are real parameters.

The Gauss map is also called the mouse map because its bifurcation diagram resembles a mouse (see Figure 2.10).

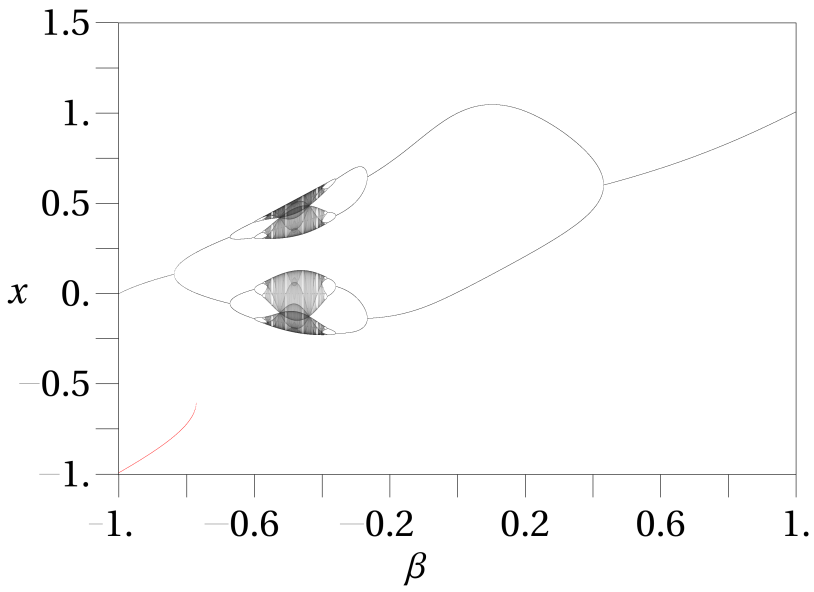


Figure 2.10 – Bifurcation diagram of the Gauss map with α= 4.90 and β in the range -1 to +1.

The discrete Gauss function is given by the following equation [160]:

$$X_{n+1} = G(X_n) = \begin{cases} 0 & \text{if } X_n = 0, \\ \frac{1}{X_n} - [\frac{1}{X_n}] & \text{otherwise} \end{cases} \tag{2.11}$$

where the notation [.] means to take the fractional part.

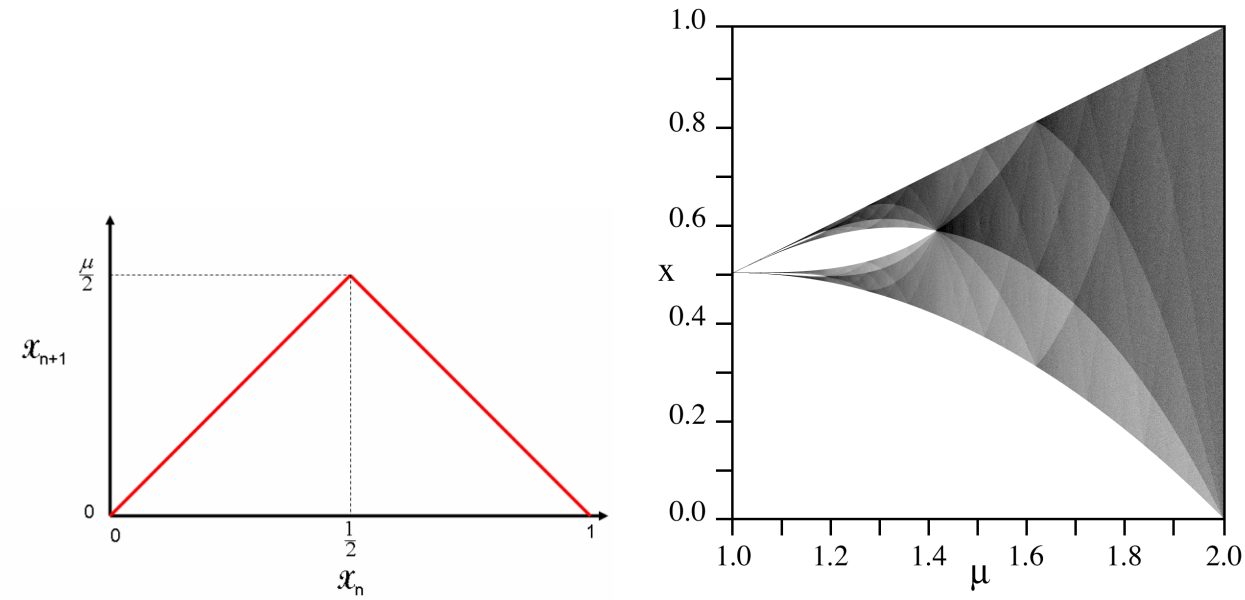
The researches have shown that the Gauss is a good example of a chaotic discrete dynamical system, in that it exhibits in an accessible fashion all the common features of such systems [67].

Tent map The Tent map is an iterated function of a dynamical system which exhibits chaotic behaviours. It is a 1-D simple map, on the unit interval J = [0, 1] into itself and governed by Eq.(2.12):

$$x_{n+1} = \begin{cases} ax_{n+1} & \text{if } x_n < \frac{1}{2} \\ a(1 - x_{n+1}) & \text{if} x_n \leq x_n \end{cases} \tag{2.12}$$

where a is a control parameter which is varied between 0 and 2.

The mapping of the Tent map function and its bifurcation diagram are given in Figure 2.11.



(a) Mapping (b) Bifurcation diagram

Figure 2.11 – Mapping and bifurcation digram of the Tent map [12].

The Tent map has been shown to have a uniform distribution. Therefore, the tent map is often used to design chaos-based crypto-systems [285].

Hénon map The Hénon map is a 2-D discrete-time dynamical map. It was introduced in 1976 by Michel Hénon as a simplified model of the Poincaré section of the Lorenz model [106]. The Hénon map takes one point (x, y) and maps this point to a new point in the plane. It is elaborated as follows:

$$\begin{cases} x_{n+1} = & 1 - \alpha x_n^2 + y_n \\ y_{n+1} = & \beta x_n \end{cases} \tag{2.13}$$

The Hénon map depends on two control parameters, α and β. It is known to display chaos for certain parameter values and initial conditions. The Hénon map is chaotic for of α = 1.4 and β = 0.3. For other values of α and β the map may be chaotic, intermittent, or converge to a periodic orbit.

The Hénon map tends toward a "strange attractor" (see Figure 2.12).

The Hénon map is an excellent system that bears all the classical chaotic characteristics, yet, it has its own disadvantages. Due its simplicity, it has become a benchmark system which is frequently used as an example to demonstrate scheme, analyse and control chaotic behaviour [210].

Lozi map Lozi Map is a 2-D map introduced by René Lozi in 1978 [154]. Lozi Map equations and attractors resemble the Hénon map, but with the term $-\alpha x_n^2$ replaced by $-\alpha|x_n|$. It is given by the following equation:

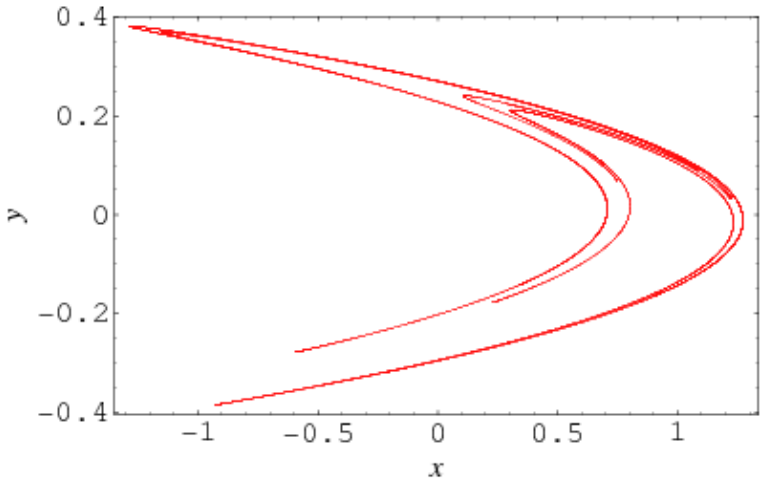


Figure 2.12 – Attractor of the Hénon map for α = 1.4 and β = 0.3. [272]

$$\begin{cases} x_{n+1} = & 1 - \alpha|x_n| + \beta y_n \\ y_{n+1} = & x_n \end{cases} \tag{2.14}$$

where α and β are bifurcation parameters

The strange attractor [178] illustrated in Figure 2.13 results from α=1.4, β=0.3.

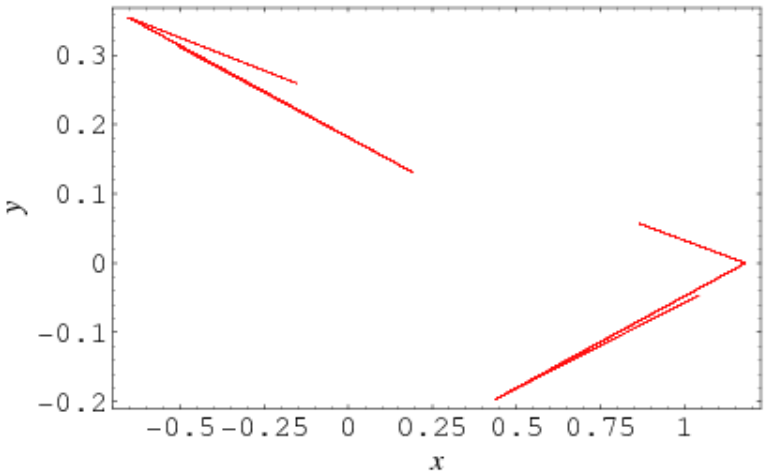


Figure 2.13 – Attractor of the Lozi map for α=1.4, β=0.3 [273].

This Lozi map is the subject of many works focused on its various properties [288].

Lorenz attractor The Lorenz attractor is one of the most know 3-D chaotic attractors. It was first studied and introduced by Edward Lorenz in 1963 [153, 246] as a simplified mathematical model for atmospheric convection. Edward Lorenz showed that a slight change in the initial conditions of a weather model would affect the whole system and could give large differences in the resulting weather. This is called sensitivity to the initial conditions. Lorenz's dynamic system is nonlinear, non-periodic, deterministic and very sensitive to the initial value. It presents a chaotic attractor which resembles a butterfly (see Figure 2.14).

The Lorenz attractor is a system of three ordinary differential equations now known as the Lorenz equations, defined as follows:

$$\begin{cases} \frac{dx}{dt} = & \sigma(y-x) \\ \frac{dx}{dt} = & x(\rho - z) - y \\ \frac{dz}{dt} = & xy - \beta z \end{cases} \tag{2.15}$$

where x, y, and z make up the system state, t is time, and σ, ρ and β are the system control parameters.

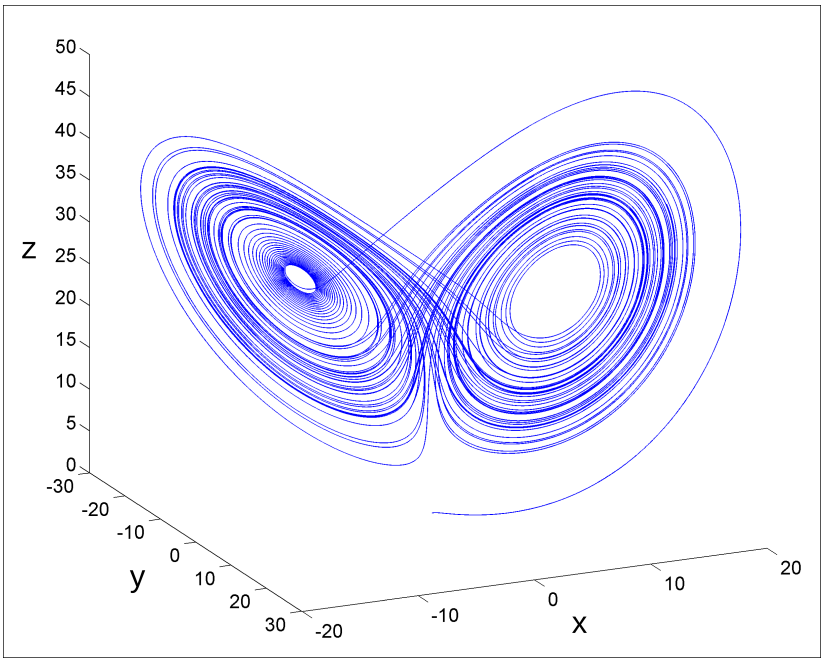


Figure 2.14 – Attractor of the Lorenz system for σ = 10, β = 8/3, ρ = 28.

The Lorenz equations have been the subject of several research articles in chaos-based cryptography. They also arise in simplified models for other applications such as lasers, dynamos,thermosyphons, electric circuits and chemical reactions.

Rössler attractor The Rössler attractor was created by Otto Rössler in 1976, with a system of three non-linear ordinary differential equations [221, 220]. These differential equations define a continuous-time dynamical system that exhibits chaotic dynamics. The defining equations of the Rössler system are:

$$\begin{cases} \frac{dx}{dt} = & -y - z \\ \frac{dx}{dt} = & x + ay \\ \frac{dz}{dt} = & b + z(x - c) \end{cases} \tag{2.16}$$

where a,b and c are constants.

O.Rössler studied the chaotic attractor with a=0.2, b=0.2 and c=5.7. The plotted attractor is a quite nice but is not famous attractor (see Figure 2.15).

2.3.3 Chaos Applications in Cryptography

Block ciphers based on Chaotic Systems As we discussed before, a block cipher is an algorithm that operates on fixed length of bits called block, with a transformation function which maps block of plaintext bits to ciphertext bits of equal bits size, specified by a symmetric key. The decryption algorithm is defined to be the inverse function of encryption: the ciphertext is divided into blocks of the same bit size and then the decryption function is applied to each block using the same shared secret key.

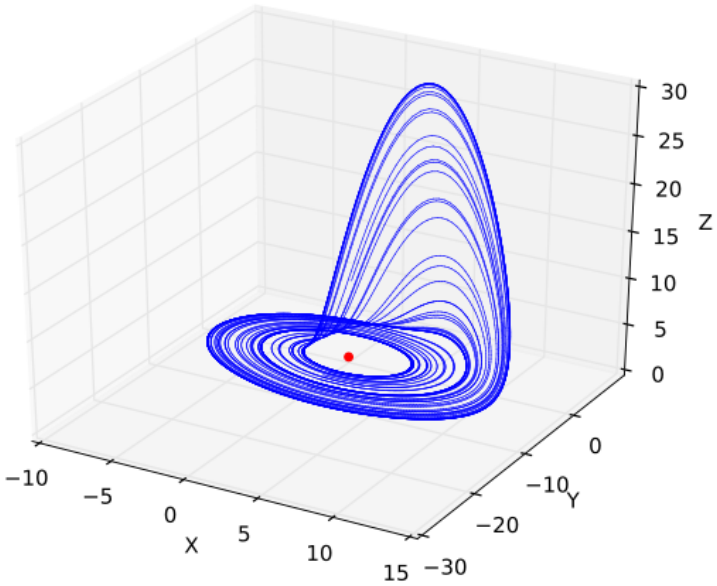


Figure 2.15 – Attractor of the Rössler system [11].

The security of a block cipher is evaluated with assuming that the attacker have access all transmitted ciphers and knows the encryption cipher details but he ignores the shared secret key. If the shared secret key is discovered, the block cipher is considered totally broken. Whereas, if part of the plaintext is retrieved, the block cipher is considered partially broken [249]. A well-designed block cipher should contains two layers: a confusion layer and a diffusion one. Confusion refers to how making each binary bit of the ciphertext should depend on several parts of the secret key, obscuring the connections between both. Diffusion means how a single bit change of the plaintext affects the ciphered bits. Several chaos-based block ciphers are based on the Fridrich structure in which the confusion and diffusion layers are separated.

A general structure of chaos-based block cipher is given in Figure 2.16, where the confusion and the diffusion layers are working separately. First, the confusion process is applied rc times on the block (or on the whole image), then the diffusion process is applied rd times on the output of the confusion process, and finally, the two processes are repeated r times. As we can see, both layers required image-scanning (for $rc = rd = r = 1$). The confusion process is usually done by substitution operation. The substitution can be achieved by any 2-D chaotic permutation map, such as: Cat map, Standard map, or Baker map [94], and also, by using any nonlinear chaotic function as the 1-D finite state Skew tent map. In the permutation case, the image pixels are relocated, but their values remain unchanged. The diffusion process changes the statistical properties of the plain-image by spreading the influence of each bit of the plain-image over all the ciphered ones. The diffusion process is essential for any secure cryptosystem, otherwise it is easy to break the system. The dynamic keys Kc and Kd are supplied by the chaotic generator(s) (keys generator(s)).

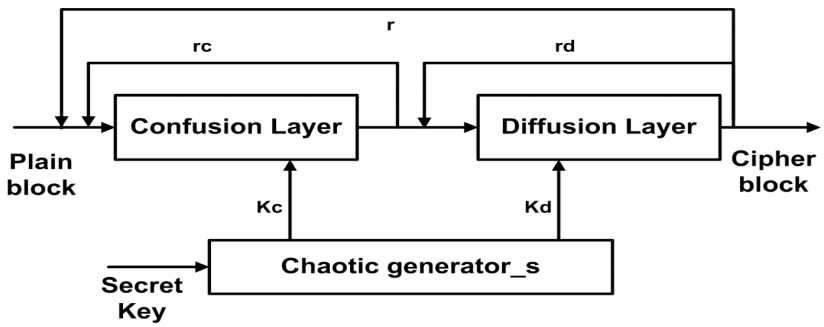


Figure 2.16 – General structure of chaos-based block cipher.

Over the past two decades, many researchers have used a chaotic system to design block cipher encryp-

tion algorithms in order to provide high security. These algorithms were studied and analyzed. Among these algorithms, there are those which are considered robust and secure and can be used in data transmission applications. But, unfortunately, some of them are described as insecure and/or slow algorithms. Therefore, further research is still needed to design fast and secure chaos-based block ciphers. In this section, we will review some chaos-based block ciphers and give brief details of their algorithms.

In 1997, Fridrich introduced a symmetric block encryption technique based on two-dimensional chaotic map [93, 94]. The general architecture of this crypto-system is shown in Figure 2.17. Fridrich crypto-system became the main structure of the most proposed chaos-based crypto-systems and it has been widely referenced since 1997.

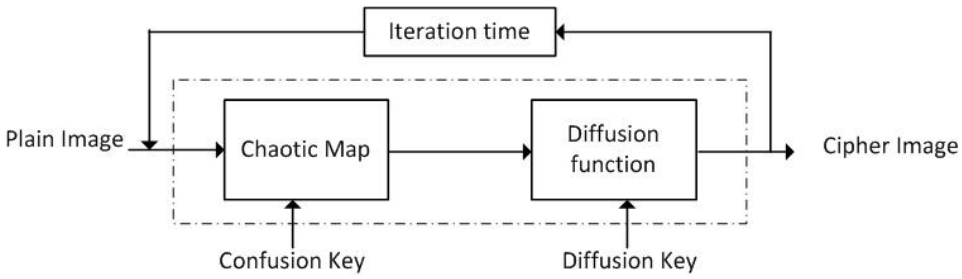


Figure 2.17 – Image encryption scheme of Fridrich [148].

Fridrich's crypto-system consists of two parts: chaotic confusion and pixel diffusion. The former process is achieved by permuting all the pixels of a plain-image as a whole, using one of the three types of 2-D chaotic maps, namely, Standard map, Cat map, and generalized Baker map defined. The parameters of the chaotic map serve as the confusion key. The diffusion process changes sequentially the value of each pixel one by one, in such a manner that the change to a particular pixel depends on the accumulated effect of all previous pixel values. The parameters of the chaotic map as the initial value or control parameter of the diffusion function serve as the diffusion key (see Eq (1.18), (1.19) and (1.20)) [148].

$$x_{j+1} = (x_j + y_j)\ mod\ N\ y_{j+1} = (y_j + k\ sin\ \tfrac{x_{j+1}N}{2\pi})\ mod\ N \qquad (1.18)$$

$$\begin{bmatrix} x_{j+1} \\ y_{j+1} \end{bmatrix} = \begin{bmatrix} 1 & u \\ v & uv+1 \end{bmatrix} \begin{bmatrix} x_j \\ y_j \end{bmatrix} (modN) \qquad (1.19)$$

$$\begin{aligned} x_{j+1} &= \tfrac{N}{k_i}\ (x_j - N_i) + y_j\ mod\ \tfrac{N}{K_i}, \\ y_{j+1} &= \tfrac{k_i}{N}\ (y_j - y_j\ mod\ \tfrac{N}{K_i}) + N_i. \end{aligned} \quad \text{with} \begin{cases} k_1 & +k_2 + ... + k_t = N \\ N_i & = k_1 + ... + k_{i-1} \\ N_i & \leq x_j < N_i + k_i \\ 0 & \leq y_j < N \end{cases} \qquad (1.20)$$

Lian et al., [148] studied the performance of Fridrich's crypto-system and its security against brute-force attack, statistical attack, known-plaintext attack, select-plaintext attack and so on, by investigating the properties of the involved chaotic maps and diffusion functions. Furthermore, They found some weaknesses and proposed some enhancement means to strengthen the overall performance of the focused crypto-system, and some advices to select suitable chaotic map, diffusion function and iteration time. In 2010, Solak et al., [245] cryptanalyzed Fridrich's chaotic crypto-system and showed that the later could be broken using chosen-ciphertext attacks.

In 1998, M.S. Baptista [34] published a new crypto-system based on ergodic property of the simple low-dimensional and chaotic logistic map. In such crypto-system, the logistic map is used as a chaotic source and its output range is divided into intervals [Xmin, Xmax). The number of intervals are S (where S is the number of symbols can be used in plaintext). Consider a plain text having S different characters set $Ca_1, Ca_2...Ca_S$, use a one to one onto mapping $f_S : X_t = X_1, X_2, ..., X_S \rightarrow A_t = Ca_1, Ca_2, ..., Ca_S$ to associate S different intervals with S different characters. After Baptista's system publication, there have

been several attacks [144, 111, 24, 25] on it and several modified versions have been proposed [208, 145, 279, 277, 278, 187].

According to Alvarez et al., [24], Baptista's encryption scheme has several weaknesses. They found that three types of cryptanalysis attacks: the chosen plain text attack, the entropy attack and the key recovery attack. Wong et al., [279] found that Baptista's approach has two main drawbacks. First, the resultant ciphertext is usually concentrated at few number of iterations. The second drawback is that the algorithm has low encryption speed and random numbers are repeated early. After these drawbacks, Baptista's crypsto-system is not competitive for standard algorithms of secure applications.

In 2004, Mao et al., [166] extended the two-dimensional chaotic baker map to be three-dimensional and proposed a new Symmetric block encryption scheme based on this map. The 3-D baker map was used to speed up image encryption while retaining its high degree of security. The proposed algorithm contains confusion and diffusion stage, and aims to obey traditional block cipher's principles (see Figure 2.18).

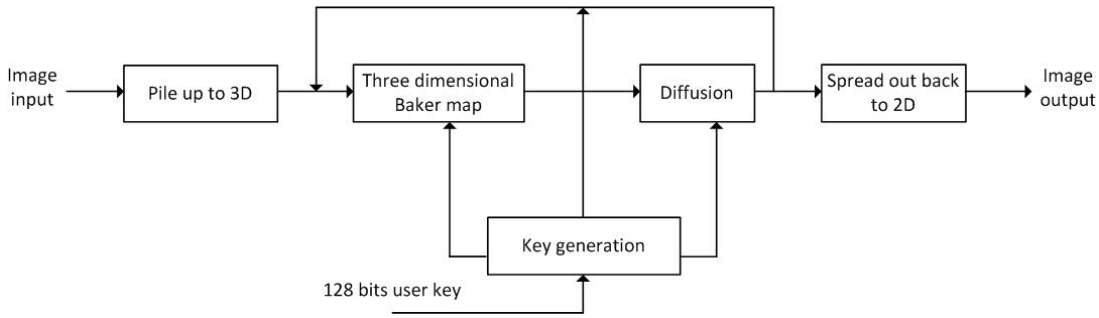


Figure 2.18 – Block cipher encryption scheme proposed in [166].

Compared to other existing similar schemes that were designed on the 2-D baker map, Mao *et al.,* scheme [166] has higher security and faster encryption/decryption speeds, which makes it a potential candidate for real-time image encryption applications.

In 2005, Lian et al., [147] designed a block cipher based on the discretized chaotic standard map, which can be presented for encrypting large-volume data sets. It is composed of three parts: a confusion process based on chaotic standard map which consists of the random-scan process, a diffusion function realized by a logistic map, and a key generator based on the chaotic Skew Tent. Some cryptanalysis on the security of the designed cipher is carried out. The cipher has satisfactory security with a low cost: it is of high key-sensitivity, and high security against brute-force attack, statistical attack and differential attack. Thus, it may provide a choice for multimedia encryption applications such as images, audios and even videos.

In 2011, Wang et al., [268] introduced the idea of combining the permutation and diffusion layers into one single layer. As a result, one image scanning is required and the algorithm may win at least two-time on image-scanning (see Figure 2.19).

In Wang's algorithm, the image is first partitioned into a number of blocks $Nb_{blocks} = \frac{L \times P}{64}$, where L and P are the height and the width of the image, respectively. Then, the pseudo-random numbers, generated from the nearest-neighboring coupled-map lattices (NCML) given in Eq.(2.17) [118], are used to modify the pixel values in the blocks. Meanwhile, the blocks are moved to new positions according to the lattice values of the NCML and some lattice values are exchanged. These steps are repeated R rounds until the required security level is reached.

$$x_{n+1}(i) = (1-\epsilon)f(x_n(i)) + \epsilon f(x_n(i+1)) \tag{2.17}$$

Using multiplication and conversion from floating points to integers operations when generating pseudo-random numbers from NCML can avoid time-consuming, which greatly increases the encryption speed. Also, the mixing of the permutation and diffusion layers makes the image scan required only once in each encryption round, which also improves the encryption speed. In addition, the new algorithm has high security level, it can well resist brute-force attack, statistical attack, differential attack, known/chosen-

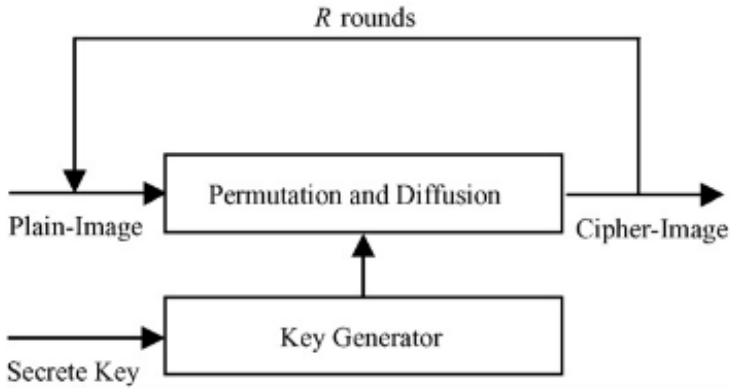


Figure 2.19 – Image crypto-system combining the permutation-diffusion architecture [268].

plaintext attacks. Therefore, the algorithm indeed has excellent potential for practical image encryption applications.

In 2016, Farajallah et al., [85] proposed an efficient crypto-system that overcome the weaknesses of Zhang crypto-system [289] while keeping a very high speed compared to the main chaos-based crypto-system of the literature. The encryption side of the proposed cryptosystem is given in Figure 2.20, for the first block.

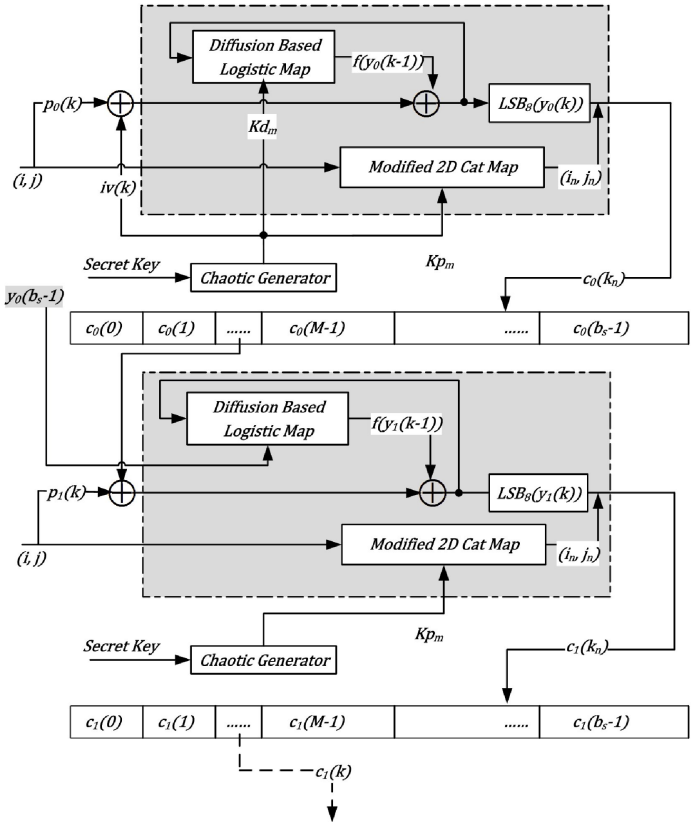


Figure 2.20 – Encryption structure of the proposed crypto-system by Farajallah et al., [85].

Each pixel from the plain block $p_0(k)$ is XOR-ed with the initial byte iv(k) from the initial vector IV, then the output is XOR-ed with the discrete logistic map output to carry out the diffusion process. Then, the 8 least significant bits resulting from the diffusion process $LSB_8\ (y_0(k))$ are relocated using the modified 2-D cat map to obtain the ciphered pixel at the new position $c_0(k_n)$. It is important to note that the input

of the discrete logistic map is based on the previous ciphered pixel (since $c_0(k_n) = LSB_8(y_0(k))$ and the input of the discrete logistic map is 32 bits and the ciphered pixel is 8 bits. That is why the crypto-system takes $y_0(k-1)$ before the LSB_8 function and not after. For the first encrypted byte, the input of the discrete logistic map is Kd_m and this value is re-initialized every new encryption round. Because the $c_0(k_n)$ is only a part of the logistic map input, it is impossible to recover $y_0(k-1)$ from $c_0(k_n)$ only. The encryption of the next blocks is almost the same. Each pixel from the plain block $p_l(k)$ is XOR-ed with ciphered byte from the previous block at the same position(i.e., $c_(l-1)(k)$ to achieve the CBC mode). Then the rest of the operations are the same as in the first encryption block.

The security level of the proposed cryptosystem is verified by testing different kinds of known mathematical attacks and statistical analysis. The proposed crypto-system in proved to be more resistant against known attacks and faster than Zhang crypto-systems; the dynamic key space is much larger. All results prove that such crypto-system is suitable for securing real-time applications.

Pseudo-random number generators and stream ciphers based on chaotic maps

Noise like appearance and unpredictable behavior of the results generated by chaotic systems have attracted the researchers' interest in applying such systems for designing Pseudo-random number generators (PRNGs) [168, 290, 238, 22, 126, 259]. PRNGs are the main element on stream cipher algorithms as keystreams are combined, using an XOR operation, to the plaintext to generate the corresponding ciphertext [239]. The keystream must be random enough and different at each new execution to ensure that if an attacker knows at instant t the keystream, he cannot recover the secret key or derive the internal state. Thus, the security of any stream cipher depends on the randomness of the keystream, therefore on the robustness of the used PRNG. In addition, it is very important to use RNG and PRNGs when generating the secret keys and initialization variables [256]. Many stream cipher algorithms have been proposed in the literature, based on pseudo chaotic number generators (PCNGs). In the design of PCNGs, many chaotic maps have been utilized including Logistic map, Tent map, Piecewise non-linear chaotic map and Hénon map. Some researchers have used multiple chaotic maps to enhance the PCNG security [276].

Over the past two decades, many researchers have utilized chaotic maps for the design of PRNGs and stream ciphers to obtain high security performance [168, 290, 238, 22, 126, 259]. Unfortunately, some of the proposed PCNGs are considered as insecure and/or slow algorithms. Therefore, further research is still needed to design fast and secure pseudo-random number generators. In this section, we will review some of pseudo-chaotic number generators and stream ciphers, and we will give brief details of insecure and slow algorithms.

In 1985, Wolfram published the first paper on a dynamical system; this was a stream cipher based on a simple one-dimensional cellular automation [276]. The cellular automation which consists of a circular register with N cells, is used to generate a random binary keystream sequence that is XORed with the plaintext to produce the correspondence ciphertext.

In 1989, Matthews et al., [173] used discrete chaotic dynamical systems for the first time, to design chaos-based stream cipher algorithm. This work has attracted the attention of many researchers. Matthews suggested using a one-dimensional chaotic map, which exhibits chaotic behavior for a range of parameter and initial values. This map is used for generating a random sequence as system keys, which serves as a one time pad for encrypting plaintext. Matthews's algorithm has been criticized later by Wheeler [275], who demonstrated that this map, when implemented on digital computer systems, produces repeating cycles of values, which are unpredictable and often have short length. Therefore, the map is not suitable for cryptographic use in the manner proposed by Matthews.

In 2001, Shujuna et al., [240] presented a novel pseudo-random binary sequence generator based on a couple of chaotic systems called CCS-PRBG. The general structure of CCS-PRBG is given in Figure 2.21. Authors used two different chaotic maps instead of one in order to provide higher security.

In the same year, another stream cipher based on the logistic map was presented [200]. Two nearby logistic map trajectories were used to generate the pseudo-random sequences with high complexity. The

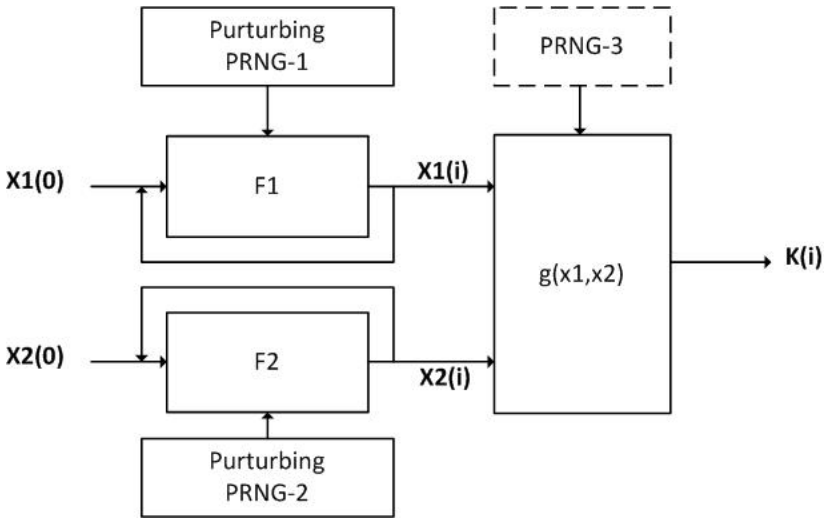


Figure 2.21 – Structure of the digital CCS-PRBG proposed in [240].

plaintext is Xored with the generated sequence to obtain the ciphertext. The simplified block diagram of this algorithm is shown on Figure 2.22 [244] where: y_n is the plaintext stream, C_n is a ciphertext, $\oplus$ means a bitwise xor operation, x_0, x'_0 and λ are the cipher's key. f is the logistic map function.

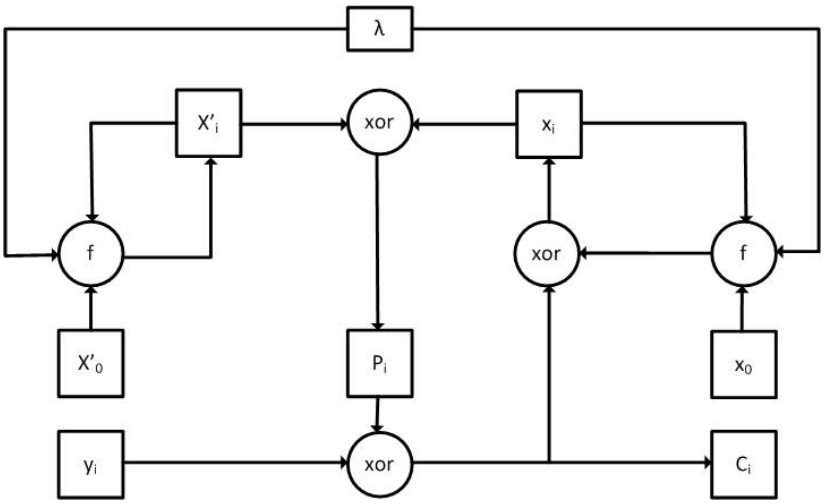


Figure 2.22 – Structure of the chaos-based stream cipher proposed by [200].

Skrobek [244] presented an efficient attack on the values of the key of this algorithm. Other weaknesses of this cipher are presented, and proposals of algorithm's improvement as well.

In 2003, Lee et al., [138] proposed a new scheme for generating good pseudo-rando mnumbers, based on the composition of multiple chaotic maps. The proposed algorithm generates first a sequence of pseudo-random bytes by using a known chaotic dynamical system, then applies certain permutations to them, using the discretized version of another two-dimensional chaotic map. The proposed cipher can generate a high percentage of usable pseudo-random numbers, while maintaining a large key space for potential use in encryption. Thus far, there have been no successful attacks on this cipher. In the same year, a new chaos-based PRNG were proposed for cryptographic applications [126]. Its construction is based on the fact that the inverse of a function is not a well-defined function, and has a large number of branches, although the inverse can be easily computed on a particular branch. The proposed generator uses only binary operations.

In 2005, Addabbo et al., [16] proposed a pseudo random number generator, based on a family of digital maps derived from the discretized chaotic Sawtooth map, as a source of long-period pseudo random bits. Several statistical parameters and tests showed that these PRNG have a good performance in terms of period length and statistical properties of the generated sequences, while requiring a moderate increase in silicon

area in comparison with LFSRs, that are the reference for low hardware complexity PRNGs.

In 2006, Wang et al., [266] proposed a new chaotic pseudo-random bit generator (PRBG), named NDF-PRBG, using n-dimensional non-linear digital filter (NDF) and chaotic maps. They used a coupling method followed by a quantization function to overcome the effects of finite wordlength to NDF and to hide its dynamic behaviour (see Figure 2.23). Proposed PRBG is confirmed to have perfect cryptographic properties, and can be used to construct stream ciphers with high level security. Moreover, it is much faster than other chaotic pseudo random number generators due to the inherent parallel structure of NDF.

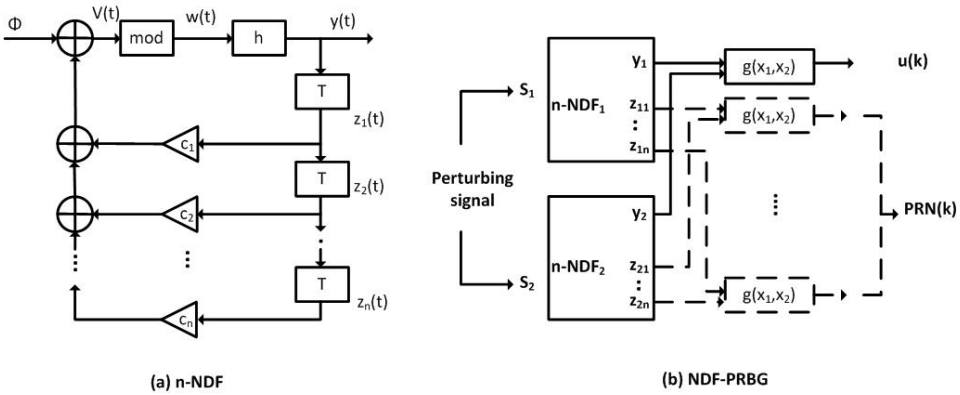


Figure 2.23 – Architecture of NDF-PRBG [266].

Yu and Cao [286] proposed a novel approach of encryption based on chaotic Hopfield neural networks with time varying delay. They used the chaotic neural network to generate a binary pseudo-random sequence, which will be used for masking plaintext. The plaintext is masked by switching of chaotic neural network maps and permutation of generated binary sequences. Li et al., [140] studied the performance of this chaos-based encryption algorithm. They proved that the generated pseudo-random sequence does not have uniform distribution and sufficient randomness. In addition, this scheme is insecure against the differential known-plaintext attack and the chosen-plaintext attack, in which only two known/chosen plaintexts are required to achieve a perfect breaking performance.

Kwok et al., [134] proposed a fast chaos-based image encryption system with stream cipher structure. A 32-bit precision representation with fixed point arithmetic is assumed. The major core of the encryption system is a pseudo-random keystream generator. It consists of two parts, serving for the generation of initial keystream and mixing, based on a cascade of chaotic high-dimensional cat map and tent map, respectively. It is found that such a design not only achieves a very fast throughput, but also enhance the randomness, even under finite precision implementation. Thus far, there have been no successful attacks on this algorithm.

Chong Fu et al. [95], proposed an improved chaos-based key stream generator to enlarge the key space, extend the period and improve the linear complexity of the key stream under precision restricted condition so as to enhance the security of a chaos-based image encryption system. The generator is constructed by three Logistic maps and a nonlinear transform. The balance and correlation properties of the generated sequence are analyzed in this work.

Ahmed et al., [17] published a chaos-based feedback stream cipher (ECBFSC) for image cryptosystems. The proposed stream cipher is based on the use of a logistic map and an external secret key of 256-bit. The initial conditions for the logistic map are derived using the external secret key by providing weight to its bits corresponding to their position in the key. Furthermore, new features of the proposed stream cipher include the heavy use of data-dependent iterations, data-dependent inputs, and the inclusion of three independent feedback mechanisms.

In 2008, Kurian et al., [133] proposed a new chaotic stream cipher for digital communication. It uses one-dimensional chaotic systems such as Logistic map and Tent map. The algorithm utilized the Symbolic dynamics (SD) of chaotic system based synchronisation to produce a pseudo-random sequence as a keystream. The plaintext is then encrypted using the SD of the Tent map or the Logistic map with certain values of its initial conditions and parameters. Statistical tests reveal that the proposed system qualifies as

random binary source. In [29], the authors studied the proposed stream cipher encryption scheme. Analysis of the keystream reveals the existence of some security problems, since a chosen-plaintext attack makes possible to estimate the control parameter of the underlying chaotic map, based on a "noisy" version of the keystream.

Patidar et al., [191] presented an image encryption scheme based on chaotic standard and logistic maps with simple mixing operation. Rhouma et al., proposed an equivalent description of the Patidar et al.'s cryptosystem which facilitated them in the cryptanalysis of the original cipher in terms of chosen plainext and known plaintext attacks. They found that the scheme can be broken with only one known/chosen-plaintext and the corresponding ciphertext. [209]. Later, Patidar et al., proposed modifications in their image cipher to make it robust against these two cryptanalytic attacks [190]. In [141], Li et al., pointed out that the modified scheme is still insecure against the same known/chosen-plaintext attack. In addition, some other security defects existing in both the original and the modified schemes are also reported.

In 2010, Liu et al., [151] designed a stream-cipher algorithm based on one-time keys and robust chaotic maps, in order to obtain high security and improve the dynamic degradation. They used the piecewise linear chaotic map as the generator of a pseudo-random key stream sequence. The initial conditions were generated by the true random number generators, the Message-Digest algorithm 5 (MD5) of the mouse positions.

In 2013, Goumidi et al. [101], two schemes are combined to enhance the encryption process complexity, the key space and the robustness of the cryptosystem. First, the image is divided into two sub-images. Next, these two sub-images are encrypted using respectively stream and block cipher schemes. After that, the two sub-images are merged to create enciphered image.

In 2014, cheng et al., proposed in [63] an efficient image encryption scheme. Logistic chaos-based stream cipher is utilized to permute the color image. The MD5 hash function and the ZUC stream cipher algorithm are combined to diffuse the color image. ZUC is a new stream cipher due for possible inclusion in the Long Term Evolution standards for mobile devices. The ZUC algorithms are the new crypto- graphic algorithms recommended by CCSA to be used in 3GPP LTE (Long Term Evolution).

Vidal et al., [265] proposed a new fast and light stream cipher based on a hyper-chaotic dynamic system, a codifying method with a whitening technique and a non linear transformation. This stream cipher has been implemented in video-conference applications for smart phones.

In 2016, Jallouli et al., [112] presented two pseudo-chaotic number generators (PCNGs). The first PCNG is based on two nonlinear recursive filters of order one using a Skew Tent map (STmap) and a Piece-Wise Linear Chaotic map (PWLCmap) as non linear functions. Whereas the second one consists of four coupled chaotic maps, namely: PWLCmaps, STmap, Logistic map by means a binary diffusion matrix [D]. The structure of the two PCNGs are shown in Figures 2.24 and 2.25 respectively.

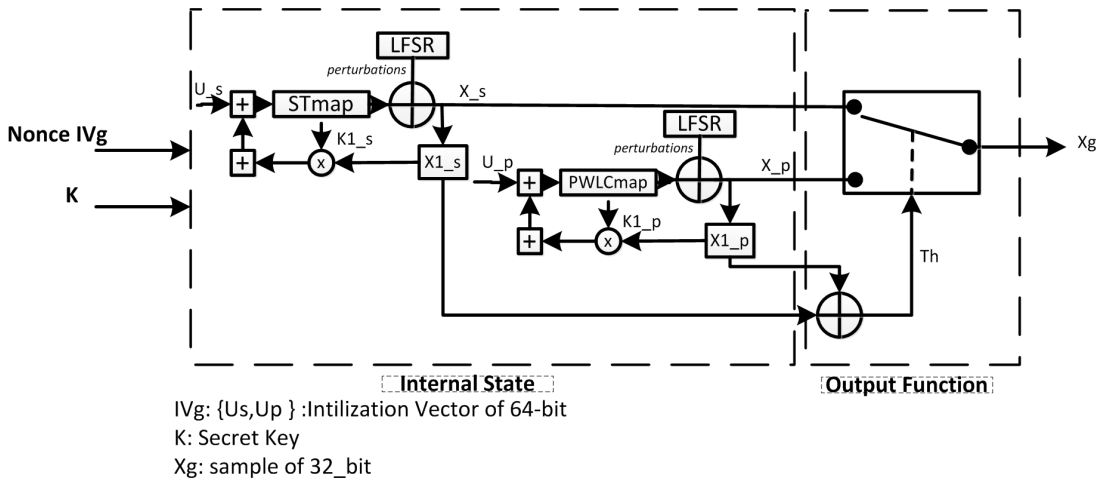


Figure 2.24 – Structure of the first proposed PCNG proposed in [112].

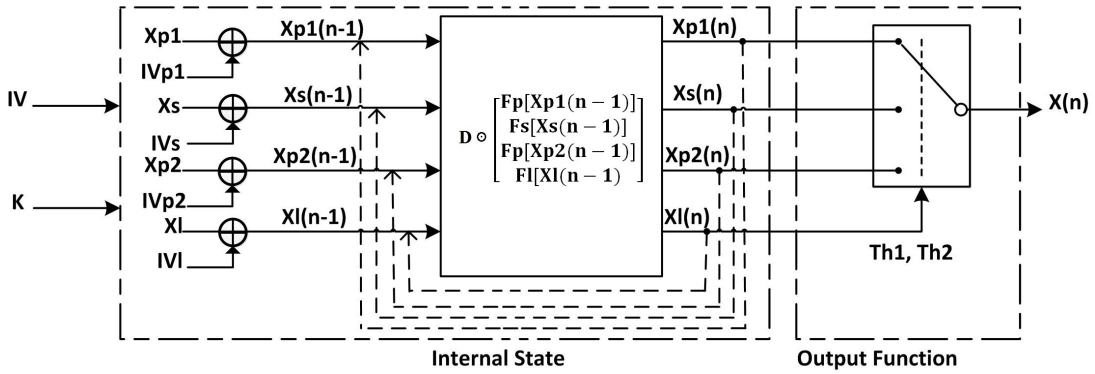


Figure 2.25 – Structure of the second proposed PCNG proposed in [112].

A comparative analysis of the performance in terms of computation time and security of the two PCNGs is carried out. The analysis study and the obtained results of the two PCNGs show that the two proposed PCNGs have strong cryptographic properties. Security performance of the first proposed PCNG is better than the second one but it is slightly slower.

2.4 Conclusion

Researchers have been attracted by chaos theory in the cryptography field due to its interesting such as deterministic nature, sensitivity to initial conditions, unpredictability, and complex structure. Over the past two decades, several cryptographic systems based on chaotic systems / maps have been proposed such as pseudo random number generators and cipher encryption algorithms. Unfortunately, there are those which suffer from security problems or slow performance. Also, some of them are not suitable to smart devices that have very limited resources in terms of memory, computing power, and battery supply. For such cipher algorithms that are particularly suited for this purpose, the main challenge is to design *lightweight* cryptographic ciphers that cope with the trade-offs between security, cost, and performance. In this chapter, we provided the fundamental concepts of cryptography primitives and the two major categories of modern cryptographic primitives, namely symmetric and asymmetric algorithms. We gave an overview of block ciphers and stream ciphers. Also, we introduce chaos theory and some chaotic maps are briefed. Finally, we presented a review of block ciphers, pseudo-random number generators and stream ciphers based on chaotic maps.

3

Performance evaluation of some chaotic maps as base of proposed chaos-based stream ciphers

3.1 Introduction

Random number sequences are used for a variety of purposes in various contexts like statistical mechanics, gaming industry, cryptography and communications, etc. The generation of such sequences may be carried out by a generator based on an algorithmic process or on a physical process. There are two basic types of random number generators: True Random Number Generators (TRNGs) and Pseudo-Random Number Generators (PRNGs). TRNGs produce a random bit stream from a non-deterministic natural source. They extract randomness from certain physical phenomena such as thermal and atmospheric noises. TRNGs are characterized by a higher security. However, their implementation requires additional devices, which make them more tedious (cost and slow) [258]. A PRNG is a deterministic algorithm that produces numbers whose distribution is uniform, by inputting an initial seed (often generated by a TRNG). PRNGs are important in practice for their rapidity in number generation, reproducibility of the pseudo-random sequences and use of less memory for storage [119].

Over the past years, Pseudo Chaotic Number Generators (PCNGs) have been one of the most important elements of chaos-based crypto-systems. Indeed, chaotic signals have very interesting characteristics for security and for digital communications such as: ergodicity, high sensitivity to initial conditions and parameters, good cryptographic properties, identical reproducibility (deterministic), broadband spectrum, auto and cross-correlation similar to pseudo-random signals [127]. The chaotic maps present potential elements in the design of a PCNG. Several generators based on continuous-time chaotic maps have been studied and proposed in the literature. However, using chaotic maps in continuous-time cannot avoid the floating data operations, which make the encryption and decryption depend on the computer's resolution. Indeed, it may be difficult to realize synchronization between the sender and the receiver if they use computers with different resolutions, because of chaos properties of high initial-value sensitivity and parameter sensitivity.

To solve this problem, discrete chaotic maps are proposed [94, 131, 171], which permits to discrete chaotic maps and make them run in integer domain. Based on these chaotic maps, the security of the corresponding crypto-system [23] is determined by the property of the discrete chaotic map and the crypto-system architecture.

In this chapter, we study the security performance of some discrete chaotic maps including: Logistic, Skew Tent and PWLCM maps, as base of proposed chaos-based stream ciphers during this thesis. We present the cryptographic properties of studied maps as well as their time computing performance. First, we define in Section 3.2 a collection of common and standard security tools useful for that assessment. Second, the chaotic maps are discretized making them run over a finite precision N=32, and their cryptographic properties and speed are analyzed in Section 3.3. As we know that the discretizing process degrades the original chaotic map's properties according to the high initial-value sensitivity, we introduce in Section 3.4.1 a perturbation technique that permits the decrease of the degradation. The security analysis of chaotic maps using the perturbation technique are performed (presented) in Section 3.4. In order to improve the cryptographic performance of chaotic maps, we propose a recursive structure described in Section 3.5.1. Then, we give in Section 3.4.4 the security and speed performance of chaotic maps using the perturbation technique and the recursive structure. Finally, some conclusions are presented in Section 3.6.

3.2 Common and standard security performance evaluation tools

In order to quantify and compare the cryptographic properties of the generated pseudo-chaotic sequences, a series of statistical security measurements and evaluation tools must be performed. These security tests check the randomness degree of the produced sequences. This is done by measuring different characteristics such as the uniformity degree of the sequence distribution. In this section, we describe in details the well-known statistical security measurements and evaluation tools. These tools are used in this chapter to study the cryptographic properties of some chaotic maps and also in Chapter 4 to evaluate the security performance of the proposed PCNGs. These security tests include phase space or mapping,

histogram, Chi-square test, auto and cross-correlation and the NIST test. Also, we present the used metrics to evaluate the speed performance of PCNGs and stream ciphers.

3.2.1 Phase space

The phase space or mapping is a diagram that reflects the signature of a chaotic map. It is a tool to visualize the behaviour of a dynamical system and does not provide any additional information about the system. The phase space refers to the graphic presentation of the corresponding differential equation or chaotic function [122]. In this thesis, for all phase space analysis, we plot the mapping of a decimal random sequence formed by 31250 samples.

3.2.2 Histogram analysis

The histogram is a graphical representation of the numerical data distribution. It is an estimation of the probability distribution of a random or pseudo-random variable. It was first introduced by Karl Pearson [195]. We use the histogram test to study the distribution uniformity of the generated random sequences.

3.2.3 Chi-square test analysis

A random sequence that has a good cryptographic properties must provide a uniform distribution. The histogram is a visual test of uniformity. It is necessary, but it is not sufficient. To ensure the uniformity, the Chi-square test is applied to statistically confirm the uniformity of the histogram [282, 184].

The experimental Chi-square χ^2 value is given by:

$$\chi^2_{\text{exp}} = \sum_{i=0}^{K-1} \frac{(O_i - E_i)^2}{E_i}. \tag{3.1}$$

where K is the number of classes (sub-intervals) chosen in our experiment equal to 1000, O_i is the number of observed (calculated) samples in the i-th class and E_i is the expected number of samples of a uniform distribution, $E_i = 10^7/K$.

To prove the uniformity of a sequence, the experimental value of Chi-Square must be lower than the theoretical one. Also, the smaller the experimental value of Chi-Square is than the theoretical one, the better the uniformity of the histogram.

For the histogram and Chi-square experiments, we generate 320 different decimal sequences, each one with a different secret key formed by 31250 samples.

3.2.4 Correlation analysis

Correlation analysis is also one of the statistical test that are used to evaluate the security performance of a generated random sequence. Correlation reflects the intensity of connection which may exist between two random variables. For cryptographic application, the values in a random sequence must not be repeated nor correlated. To evaluate the statistical analysis, three metrics can be used: the cross correlation which is a measure of similarity of two series, the auto-correlation, which is the cross-correlation of a signal with itself, and the correlation coefficient. The cross and auto-correlation give information about how much the sequence of a random numbers as a whole depends on the other sequence or on the value of the preceding members in the sequence itself. If two sequences X and Y are not correlated, then the correlation coefficient ρ_{XY} between X and Y should be close to zero. Else if sequences X and Y are highly correlated, then ρ_{XY} should be close to one. ρ_{XY} is given by the following equation [207]:

$$\rho_{XY} = \frac{\sum_{i=1}^{N}(x_i - \bar{X})(y_i - \bar{Y})}{[\sum_{i=1}^{N}(X_i - \bar{X})^2]^{1/2} \times [\sum_{i=1}^{N}(Y_i - \bar{y})^2]^{1/2}}. \tag{3.2}$$

where $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} x_i$ and $\bar{Y} = \frac{1}{N}\sum_{i=1}^{N} Y_i$ are the mean values of two sequences X and Y respectively.

We produce two sequences of a random numbers computed with nearby initial conditions, each composed of 31250 samples, to analyse its correlation properties.

3.2.5 NIST test analysis

We apply the NIST statistical test, which presents one of the most popular standard test for analysing randomness of binary data [224]. The STS 2.1.2 version statistical test suite published in [81] is used. It consists of a battery of 188 tests (globally 15 different tests) to conclude regarding the randomness or non-randomness of binary sequences. For each test, a set of m P-values are expected to indicate failure. Indeed, an α = 0.01, indicates that 1% of the sequences are expected to fail.

- A $P - value \geq \alpha = 0.01$ would mean that the sequence would be random with a confidence of $(1 - \alpha) = 99\%$.
- A $P - value < \alpha = 0.01$ would mean that the conclusion was that the sequence is non-random with a confidence of $(1 - \alpha) = 99\%$.

To apply the NIST test, we generate 100 different binary sequences, each one with a different secret key (size of each sequence equal to 31250 samples = 10^6 bits) and $\alpha = 0.01$.

3.2.6 Computing performance analysis

Computing performance is an important factor for evaluating the performance of a chaotic maps and PCNGs. For that, we calculate the Bit Rate (in Mega bits per second) and the number of needed cycles to generate one byte (NCpB). The later permits to compare the speed performance of different systems working on different platforms. The Bit Rate and NCpB are calculated respectively as follows:

$$Bit\ Rate(Mbps) = \frac{Generated\ data\ size(Mbits)}{Average\ generation\ time(s)} \tag{3.3}$$

$$NCpB = \frac{CPU speed(Hz)}{Bit\ Rate(Byte/s)} \tag{3.4}$$

In this thesis, all experiments are performed on a personal computer with Intel(R) Core(TM) i5-4300M CPU @2.60GHz and memory 15,6 GB and the operating system is Ubuntu 14.04 Trusty Linux distribution, using GNU GCC Compiler. For all speed performance evaluations, we give, over 100 different secret keys, the average Bit Rate in Mbps and the average number of needed cycles to generate one byte (NCpB).

3.3 Performance Evaluation of some chaotic maps

Chaotic maps are dynamic systems defined in real by recurrence relations, given by the following equation:

$$x_i(n) = f(x_1(n-1), x_2(n-1), ..., x_m(n-1)), i = 1, 2..m \tag{3.5}$$

where $x \in S, f : S \rightarrow S$ is a function with m variables, $S \subset [0, 1] ou [-1, 1]$.

Some one-dimensional chaotic maps such as the Logistic map, the Piecewise Linear Chaotic Maps (PWLCM), and the Skew Tent [126, 199, 239], and two-dimensional chaotic maps such as: the Cat map, the baker map, the standard map [94] And the Lozi map [155, 157] are studied in the literature and widely used for the design of random number generators and in chaos based crypto-systems.

The chaotic generators proposed in this thesis are based on the following discrete chaotic maps: Logistic map, Skew Tent and PWLCM map, using finite precision N = 32 bits.

Fig. 3.1 shows the general scheme of generating a pseudo-random sequence using a chaotic map.

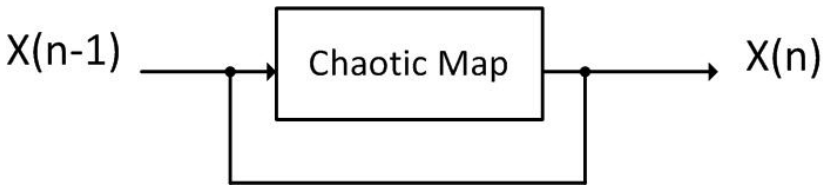


Figure 3.1 – Scheme for generating a pseudo-random sequence by a chaotic map.

In the following sections, we discuss the performance of each discrete chaotic map namely Logistic map, Skew Tent and PWLCM, by presenting the results of the various security performance tests performed.

3.3.1 Performance Evaluation of the Logistic map

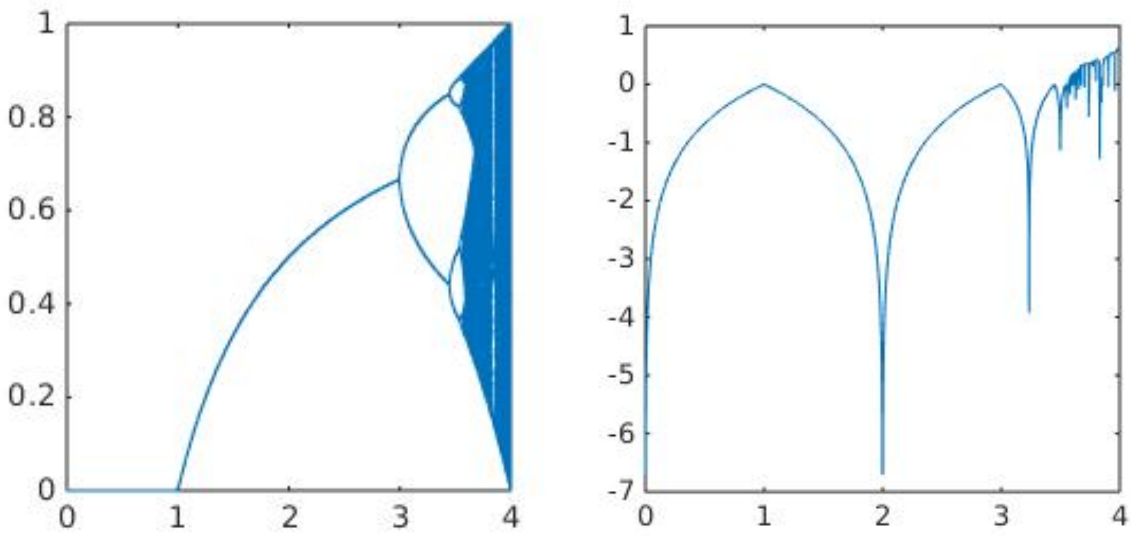
The logistic map is a one-dimensional map displaying a singularity, that is characterized by the simplicity of its recurrence equation depending on a single parameter λ. The logistic equation first created by Pierre Francois Verhulst [264] as a discrete-time demographic model and it was popularized by Robert May who used it as a pseudo random generator [174]. Since then, it is one of the most used maps in cryptographic applications. The basic form of the logistic map is given by the following equation 3.6.

$$F_L(x_{n-1}) = x_n = \lambda x_{n-1}(1 - x_{n-1}) \tag{3.6}$$

with $F_L : S \to S =]0, 1]$, $x_n \in S$.

The values of interest for the growth rate parameter λ are those in the interval [0,4]. The Logistic map exhibits an astonishing range of behavior as the growth rate λ is varied [189, 257].

In Fig. 3.2, we present the known curves of the bifurcation diagram and the Lyapunov exponent of the Logistic map, with a growth rate λ values between 0 and 4.



(a) Bifurcation Diagram (b) Lyapunov Exponent

Figure 3.2 – Bifurcation Diagram and Lyapunov Exponent of the Logistic map.

Fig. 3.2a shows that for growth rates λ less than one, the system always eventually collapses to zero. For growth rates λ between 1 and 3, the system always settles into an exact stable population level. At a growth rate between 3 and 3.4, the system oscillates between two population values. Just beyond a growth rate of 3.4, the diagram bifurcates into 4 paths and with r increasing beyond a growth rate of 3.5, beyond a growth rate of 3.5, the system oscillates over 8 population values, then 16, 32, etc.

Beyond a growth rate of 3.75, the bifurcations ramp up until the system is able to land on any population value. The length of the parameter intervals that yield oscillations decreases rapidly; the ratio between the lengths of two successive bifurcation intervals tends to the Feigenbaum constant $\delta \simeq 4.66920$. This behavior gives a concrete example of a period-doubling cascade. However, for a growth rate value equal to 4, the population value covers the whole interval]0,1]. For that, in the discrete version of the map, we set the value of the growth rate (control parameter) to the optimal value corresponding to 4 ($\lambda = 4$) [48].

The discrete Logistic map equation for the control parameter set to 4 is given by the following equation [196]:

$$F_L[X(n-1)] = X(n) = \begin{cases} \left\lfloor \frac{X(n-1)\times[2^N - X(n-1)]}{2^{N-2}} \right\rfloor & \text{if } X(n-1) \neq [3 \times 2^{N-2}; 2^N] \\ 2^N - 1 & \text{if } X(n-1) = [3 \times 2^{N-2}; 2^N] \end{cases} \tag{3.7}$$

where $\lfloor Z \rfloor (functionFloor)$ is the greatest integer less than or equal to Z.
$X(n)$ takes an integer value $\in$ [0, 2^N-1] and N = 32 is the used precision.

Phase space trajectory analysis We draw in Fig. 3.3, the phase space (mapping) of a sequence $X_L(n)$ produced by the discrete chaotic Logistic map and formed of 31250 samples, the attractor and the discrete variation. The chosen initial condition $X_L(0)$ is equal to 1488169157.

The produced phase space trajectory clearly shows the signature relating to the Logistic map.

Histogram and Chi-square test analysis We plotted in Fig. 3.4 the histogram of the generated sequence $X_L(n)$. Visually, the generated sequence $X_L(n)$ is not uniform over all values [1, 2^{32}-1]. This is intuitively confirmed since the invariant measure of $X_L(n)$ is given by $P(X_L(n)) = \frac{1}{\pi \times \sqrt{X_L(n)(1-X_L(n))}}$.

We confirm the non-uniformity of the sequence using the Chi-square test.

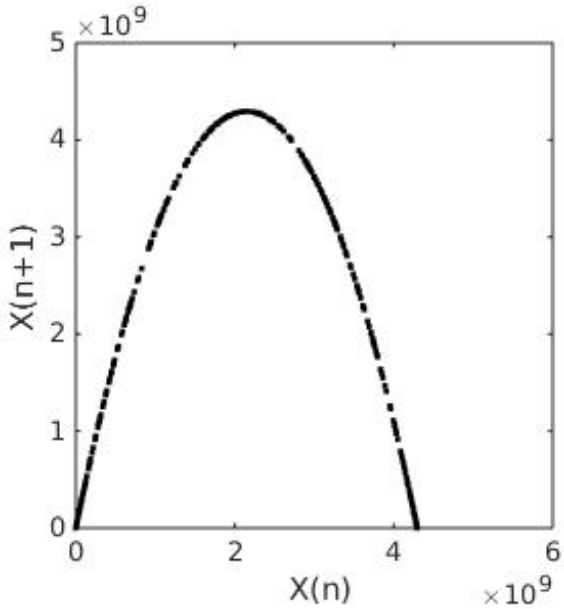
The calculated experimental value χ^2_{exp} is equal to 12546727 which is significantly higher than the theoretical one χ^2_{th}, equal to 1073,64. This asserts the non-uniformity of the sequence. However, as the Logistic map is the most used chaotic map in cryptographic applications, only the following interval [$\lfloor 0.2 \times 2^{32} \rfloor$, $\lfloor 0.8 \times 2^{32} \rfloor$] is useful and in which the values follow a uniform distribution.

Correlation analysis To evaluate the correlation between two sequences X_L and $X_{L'}$ generated using slightly different keys, we calculate the correlation coefficient $\rho_{X_L, X_{L'}}$. The obtained value is equal to -0,0019. Also, we draw the auto-correlation of sequence X_L and a zoom of this autocorrelation on 200 samples in Fig. 3.5. And we plot in Fig. 3.6, the cross-correlation between the two sequences X_L and $X_{L'}$, a zoom on it and a zoom of the auto and cross-correlation. Obtained results show that the different generated sequences X_L and $X_{L'}$ have good auto and cross-correlation properties. This asserts the pseudo-randomness of the generated sequences.

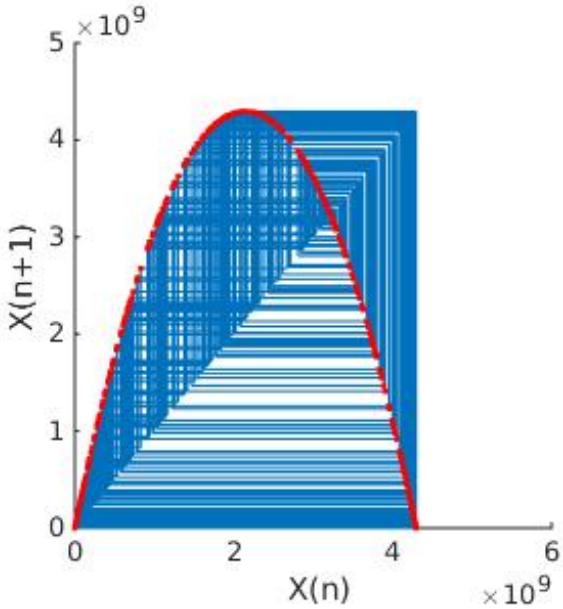
NIST test analysis We performed the NIST test by generating 100 different sequences, each of size equal to 31250 samples and using different secret key for each sequence (the total size of sequences is equal to 10^7 bits). In Table 3.1 and Fig. 3.7, we present the obtained results of the NIST test.

The obtained results show that the sequences do not pass all the NIST tests and that some tests are far from the acceptance threshold of a test materialized by the red line. This shows that the Logistic map does not have good cryptographic statistical properties for all values [1, $2^N - 1$] or]0, 1].

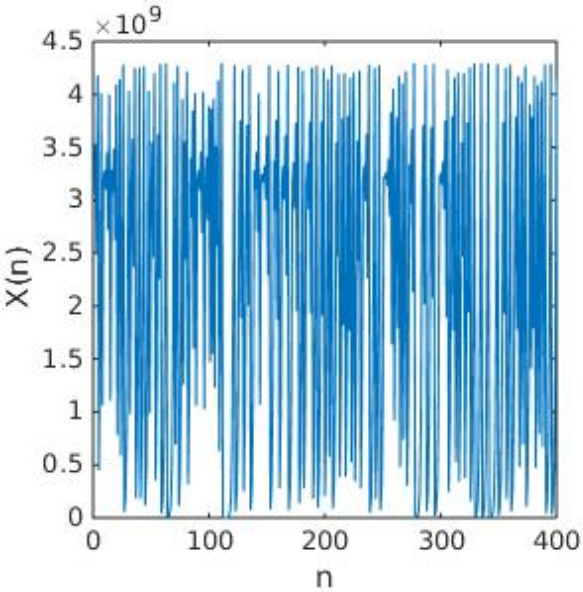
Computing performance of the Logistic map We evaluate the computing performance of the Logistic map and we report in Table 3.2, the obtained results in average for the Generation Time in μs, the Bit Rate in Mbits/s and the number of cycle needed to generate one byte (NCpB).



(a) Phase space



(b) Attractor



(c) Discrete variation

Figure 3.3 – Phase space trajectory, attractor and discrete variation of sequence $X_L(n)$ generated by the discrete Logistic map.

Obtained values show that the Logistic map is characterized by a high bit rate compared to the bit rates of the other chaotic maps presented in the following sections.

3.3.2 Performance Evaluation of the Skew Tent map

The Skew Tent map is a one dimensional piecewise map, exhibiting chaotic dynamics. It is a non invertible transformation of the unit interval onto itself. It depends on the one parameter p. The Skew Tent map is given by:

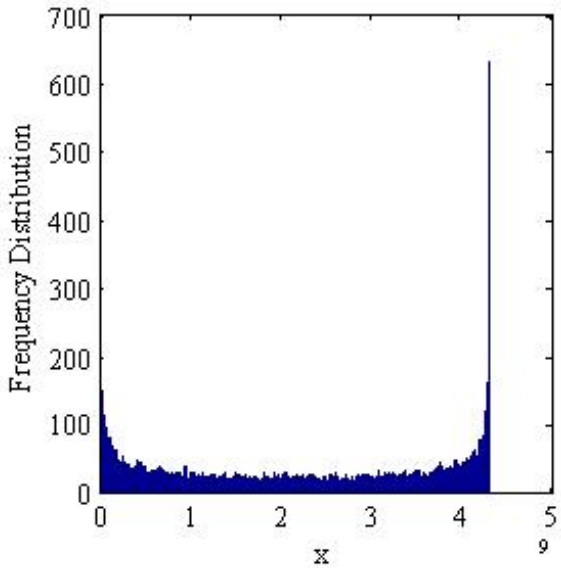
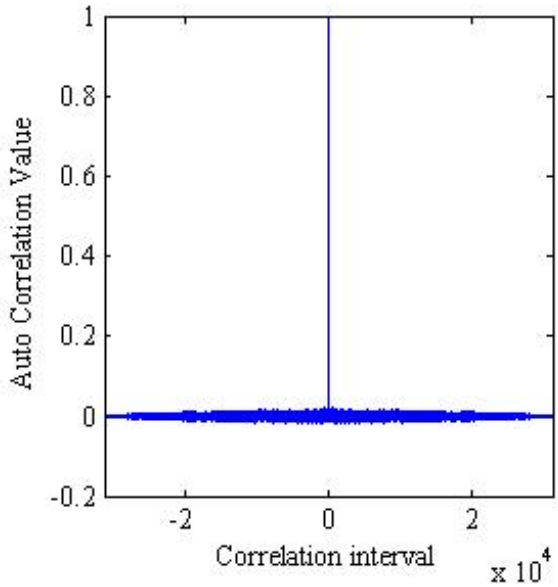
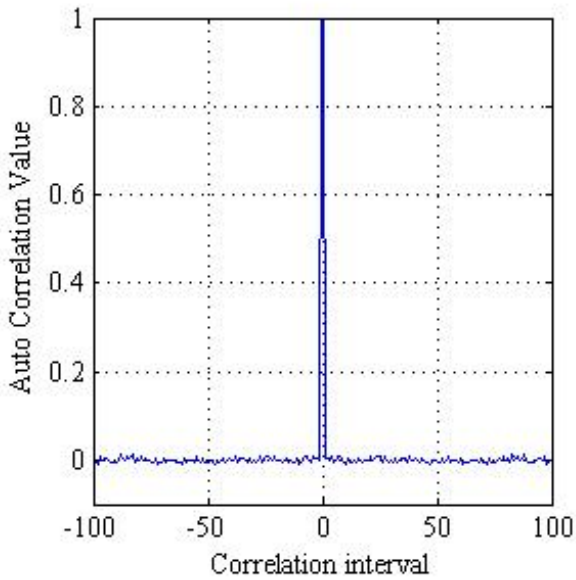


Figure 3.4 – Histogram of sequence X_L generated by the discrete Logistic map.



(a) Auto-correlation of sequence X_L



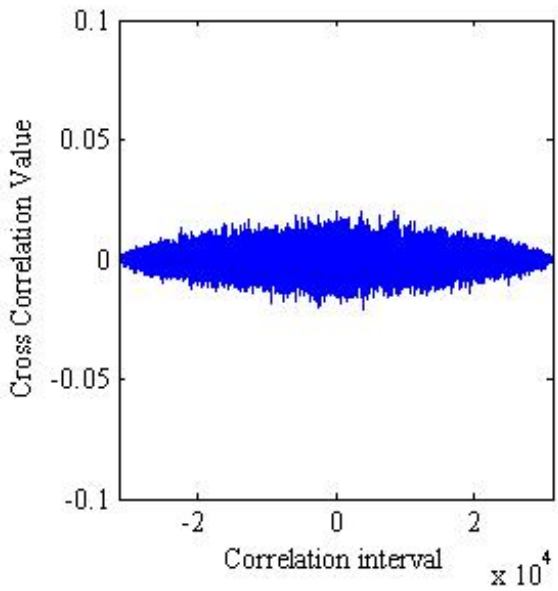
(b) A zoom of the autocorrelation on 200 samples of sequence X_L

Figure 3.5 – Auto-correlation of sequence X_L generated by Logistic map.

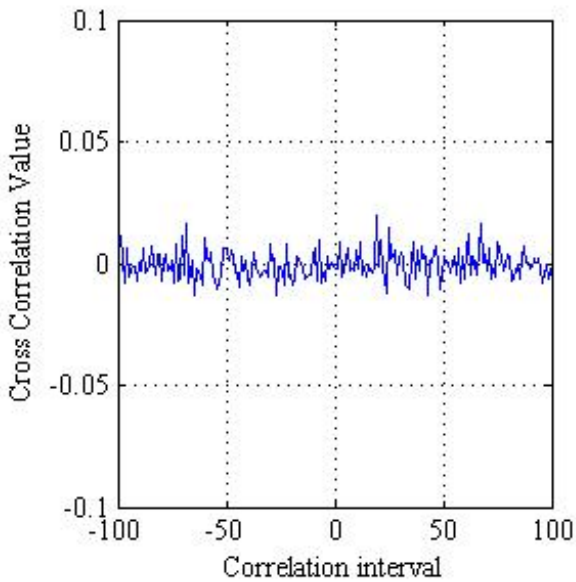
$$F_S(x_{n-1}) = x_n = \begin{cases} \frac{x_{n-1}}{p} & \text{if } 0 \leq x_{n-1} \leq p \\ \\ \frac{x_{n-1}-1}{p-1} & \text{if } p < x_{n-1} \leq 1 \end{cases} \tag{3.8}$$

where: $F_S : S \rightarrow S, S =]0, 1]$ and p is the control parameter with p $\in$]0,1[, $x_n \in S$.

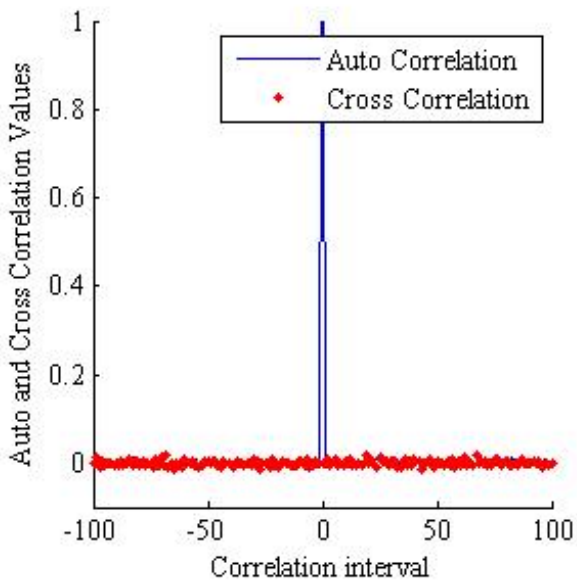
If p is equal to 0.5, F_S becomes the regular tent map.



(a) Cross-correlation of sequences X_L and $X_{L'}$



(b) A zoom of the cross-correlation on 200 samples of sequences X_L and $X_{L'}$



(c) A zoom of the cross-correlation of sequences X_L and $X_{L'}$ and of the auto-correlation of sequence $X_{L'}$

Figure 3.6 – Cross-correlation functions of sequences X_L and $X_{L'}$ generated by the Logistic map

The equation of the discretized Skew Tent function is defined by Eq(3.9):

$$F_S[X(n-1)] = X(n) = \begin{cases} \left\lceil 2^N \times \frac{X(n-1)}{P} \right\rceil & \text{if } 0 < X(n-1) < P \\ 2^N - 1 & \text{if } X(n-1) = P \\ \left\lceil 2^N \times \frac{2^N - X(n-1)}{2^N - P} \right\rceil & \text{if } P < X(n-1) < 2^N \end{cases} \tag{3.9}$$

where $\lceil Z \rceil$ (ceiling function) is the least integer greater than or equal to Z, $X(n)$ takes an integer value that belongs to the interval [0, $2^N - 1$], and P is the control parameter with: $0 < P \leq 2^N - 1$.

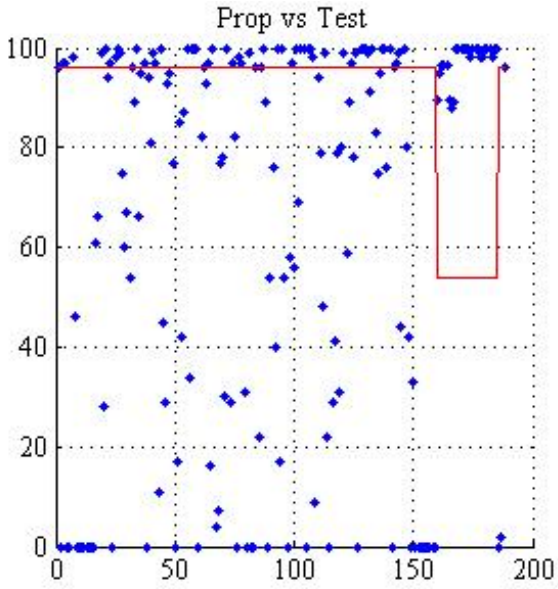


Figure 3.7 – NIST test results of the Logistic map.

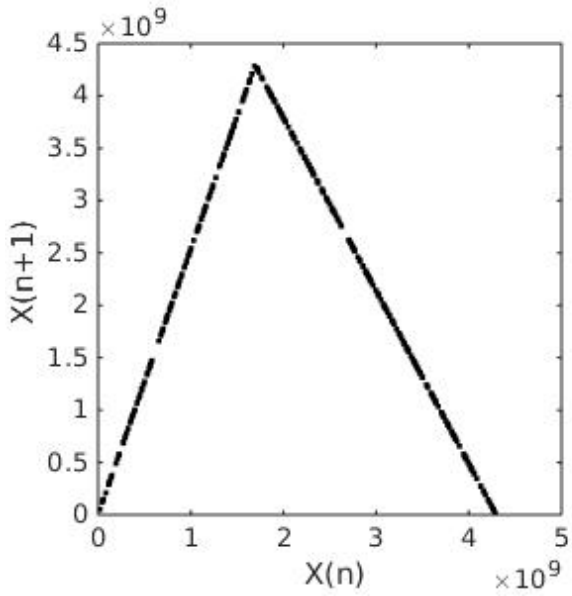
Test	**P-value**	**Proportion**
Frequency test	0.000	92.000
Block-frequency test	0.000	0.000
Cumulative-sums test	0.000	93.500
Runs test	0.000	0.000
Longest-run test	0.000	0.000
Rank test	0.419	98.000
FFT test	0.000	41.000
Non-periodic-templates	0.036	61.304
Overlapping-templates	0.000	0.000
Universal	0.000	0.000
Approximty entropie	0.000	0.000
Random-excursions:	0.002	90.367
Random-excursions-variant	0.400	99.145
Serial test	0.000	0.500
Linear-complexity	0.081	97.000

Table 3.1 – P-values and Proportion results of NIST test for the Logistic map.

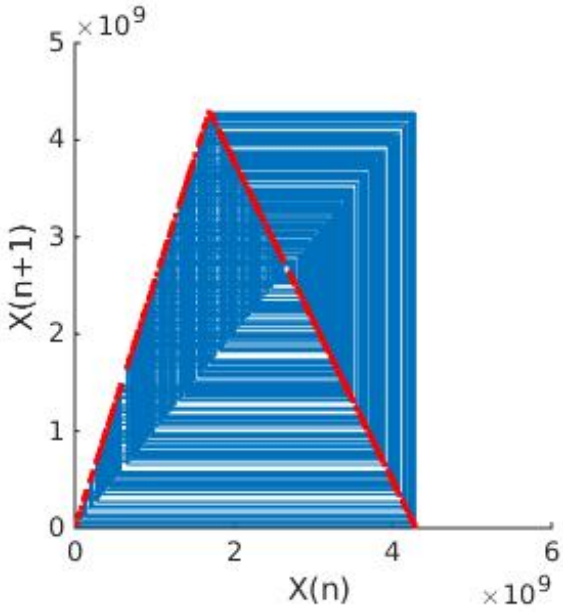
Generation Time (μs)	317.75
Bit Rate (Mbits/s)	3147.13
NCpB	3

Table 3.2 – Computing performance of the Logistic map.

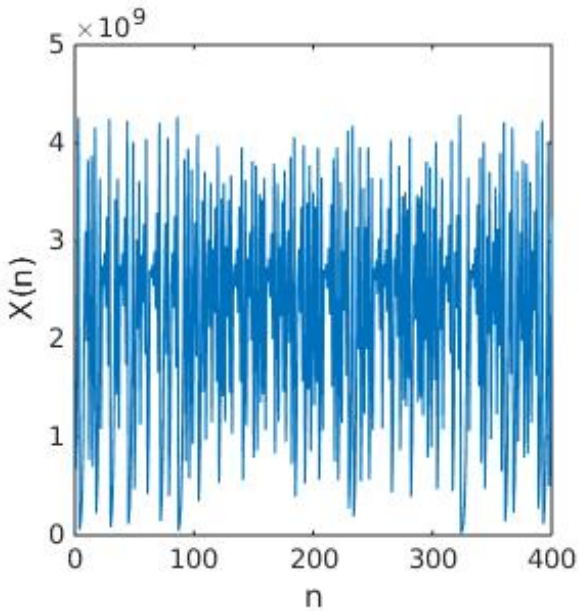
Phase space trajectory analysis We draw in Fig. 3.8 the phase space trajectory, the attractor and the discrete variation of a sequence $X_S(n)$ generated by the Skew Tent map. This figure shows the signature for the Skew Tent map.



(a) Phase space



(b) Attractor



(c) Discrete variation

Figure 3.8 – Phase space trajectory, attractor and discrete variation of sequence $X_S(n)$ generated by the discrete Skew Tent map.

Histogram and Chi-square test analysis Fig. 3.9 presents the histogram of a sequence $X_S(n)$ which is visually more uniform than the histogram of a sequence $X_L(n)$ generated by the Logistic map.

However, the Skew Tent map is not uniform. This is proved by the Chi-square test. The experimental value of the Chi-square test is bigger than the theoretical one (χ^2_{th} = 1073.64, χ^2_{exp} = 1111.62).

Correlation analysis The Skew tent map has good auto and cross-correlation properties as shown in Fig. 3.10 and Fig. 3.11 respectively. This result is confirmed by the value of the coefficient of correlation of two sequences X_S and $X_{S'}$ generated with nearby initial, which is very low ($\rho_{X_S,X_{S'}}$ = -0.0013).

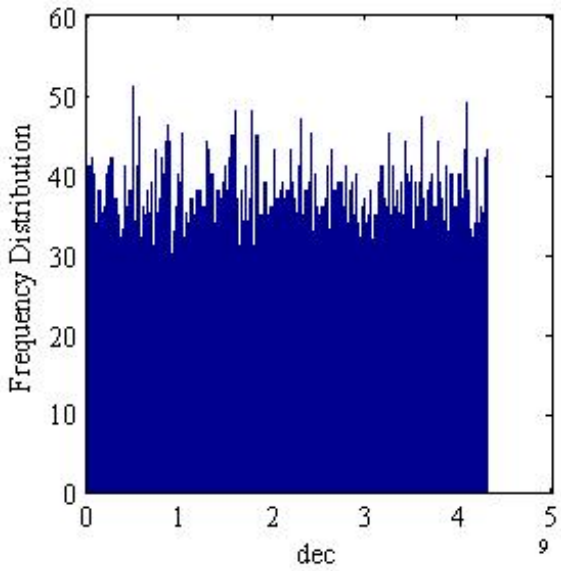
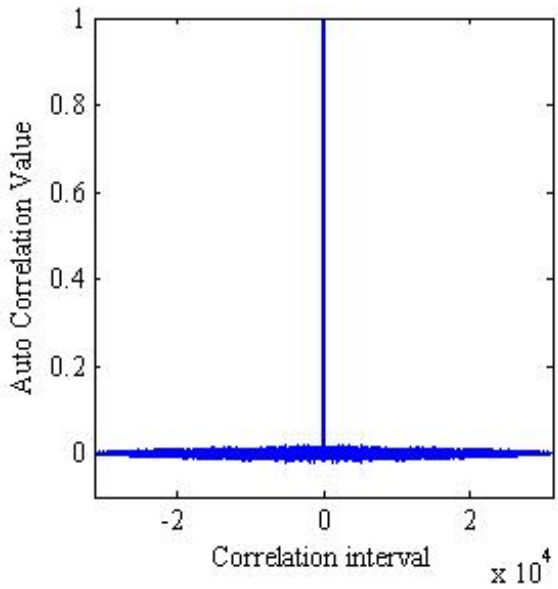
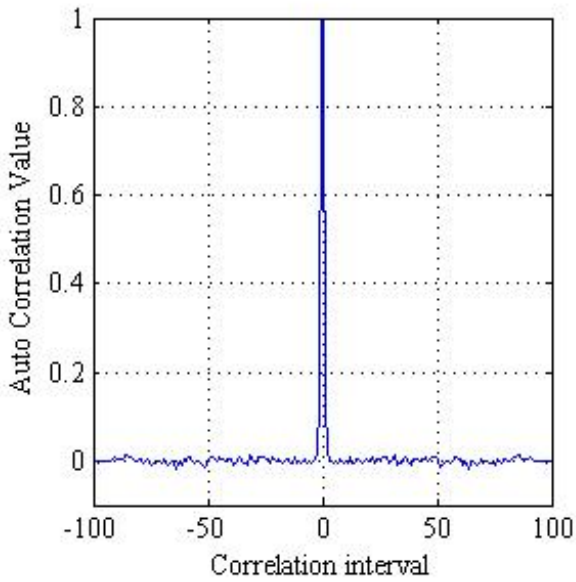


Figure 3.9 – Histogram of sequence X_S generated by the discrete Skew Tent map.



(a) Auto-correlation of sequence X_S

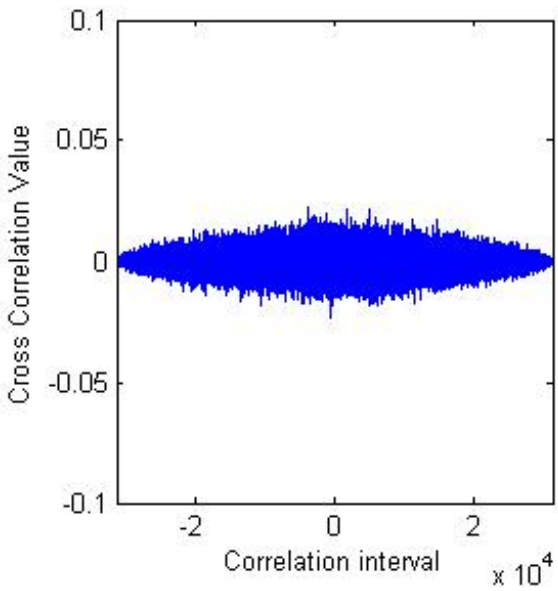


(b) A zoom of the autocorrelation on 200 samples of sequence X_S

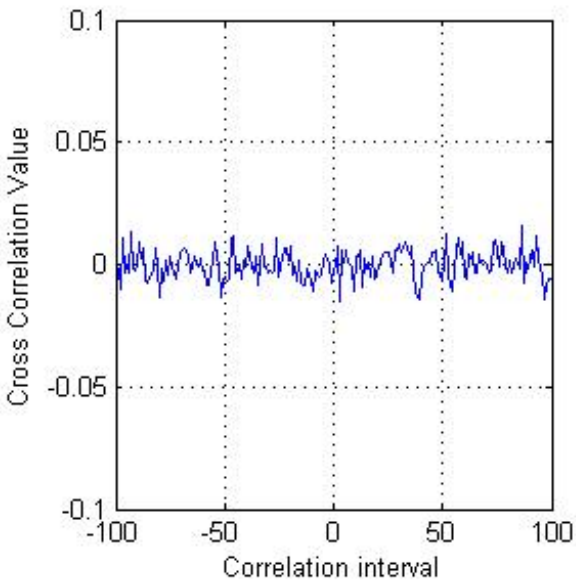
Figure 3.10 – Auto-correlation of sequence X_S generated by a Skew Tent map.

NIST test analysis Table 3.3 and Fig. 3.12 give the results of NIST test applied on a sequence $X_S(n)$. Obtained results show that sequences $X_S(n)$ does not pass all the NIST tests but they remains better than the results obtained for a sequence $X_L(n)$ generated by the Logistic map (the number of passed tests is higher and the passed tests are closer to the threshold).

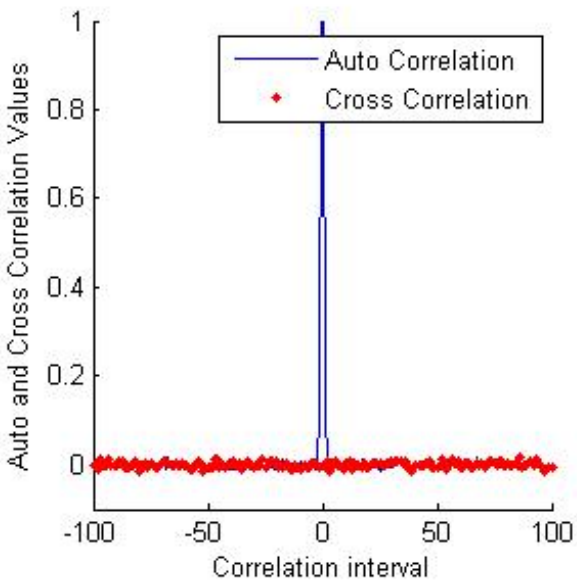
Computing performance of the Skew Tent map Table 3.4 shows the computing performance of the Skew Tent map. We note that the Skewtent map is slower than the Logistic map.



(a) Cross-correlation of sequences X_S and $X_{S'}$



(b) A zoom of the cross-correlation on 200 samples of sequences X_S and $X_{S'}$



(c) A zoom of the cross-correlation of sequences X_S and $X_{S'}$ and of the auto-correlation of sequence $X_{S'}$

Figure 3.11 – Cross-correlation functions of sequences X_S and $X_{S'}$ generated by the Skew tent map.

3.3.3 Performance Evaluation of the PWLCM map

The Piecewise Linear Chaotic Maps (PWLCM) map is another piecewise linear chaotic map, described by the following equation (3.10):

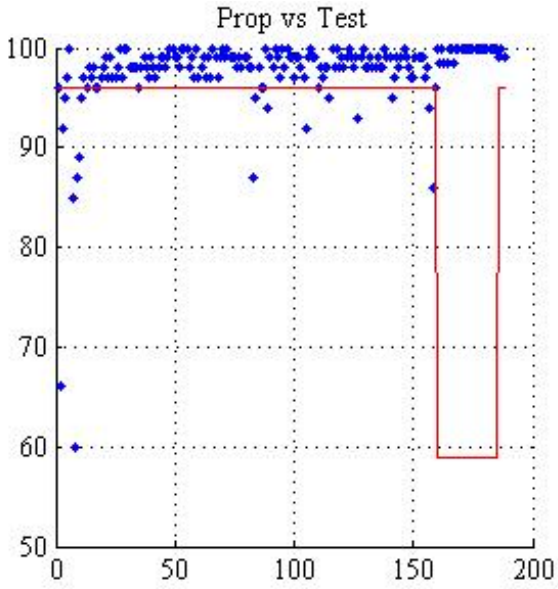


Figure 3.12 – NIST test results of the Skew Tent map.

Test	**P-value**	**Proportion**
Frequency test	0.262	93.000
Block-frequency test	0.000	59.000
Cumulative-sums test	0.198	91.000
Runs test	0.081	93.000
Longest-run test	0.575	99.000
Rank test	0.000	79.000
FFT test	0.000	55.000
Non-periodic-templates	0.531	98.108
Overlapping-templates	0.760	98.000
Universal	0.000	88.000
Approximty entropie	0.575	97.000
Random-excursions:	0.468	97.645
Random-excursions-variant	0.369	98.631
Serial test	0.402	100.000
Linear-complexity	0.103	98.000

Table 3.3 – P-values and Proportion results of NIST test for the Skew Tent map.

Generation Time (μs)	422.20
Bit Rate (Mbits/s)	2368.54
NCpB	8

Table 3.4 – Computing performance of the Skew Tent map.

$$F_P(x_{n-1}) = x_n = \begin{cases} \frac{x_{n-1}}{p} & \text{if } 0 \leq x_{n-1} < p \\ \frac{x_{n-1}-p}{0.5-p} & \text{if } p \leq x_{n-1} < 0.5 \\ F_P(1-x_{n-1}) & otherwise \end{cases} \tag{3.10}$$

where: $F_S : S \to S, S =]0, 1]$, $x_n \in S$ and p is the control parameter with p $\in$ [0, 0.5].

The PWLCM is known to be chaotic when its control parameter p is within [0, 0.5] and its initial condition is chosen within the interval]0, 1] [291]. The PWLCM has been used in data encryption due to its features of parameter-controllable and good randomness [188].

Eq(3.11) gives the discrete PWLCM function [146]

$$X(n) = F[X(n-1)] = \begin{cases} \left\lceil 2^N \times \frac{X(n-1)}{P} \right\rceil & \text{if } 0 < X(n-1) \leq P \\ \left\lceil 2^N \times \frac{X(n-1)-P}{2^{N-1}-P} \right\rceil & \text{if } P < X(n-1) \leq 2^{N-1} \\ \left\lceil 2^N \times \frac{2^N-P-X(n-1)}{2^{N-1}-P} \right\rceil & \text{if } 2^{N-1} < X(n-1) \leq 2^N - P \\ \left\lceil 2^N \times \frac{2^N-X(n-1)}{P} \right\rceil & \text{if } 2^N - P < X(n-1) \leq 2^N - 1 \end{cases} \tag{3.11}$$

where $X(n-1) \in$ [1, 2^N-1] and P is the discrete control parameter and satisfies 0 < P < 2^{N-1}.

Phase space trajectory analysis We give in the Fig. 3.13 the phase space, attractor and discrete variation of a sequence X_P, formed by 31250 samples and generated by the discrete PWLCM map, with P = 1210290246, and an initial state X(0) = 830235384. Resulted mapping shows the signature of the PWLCM map.

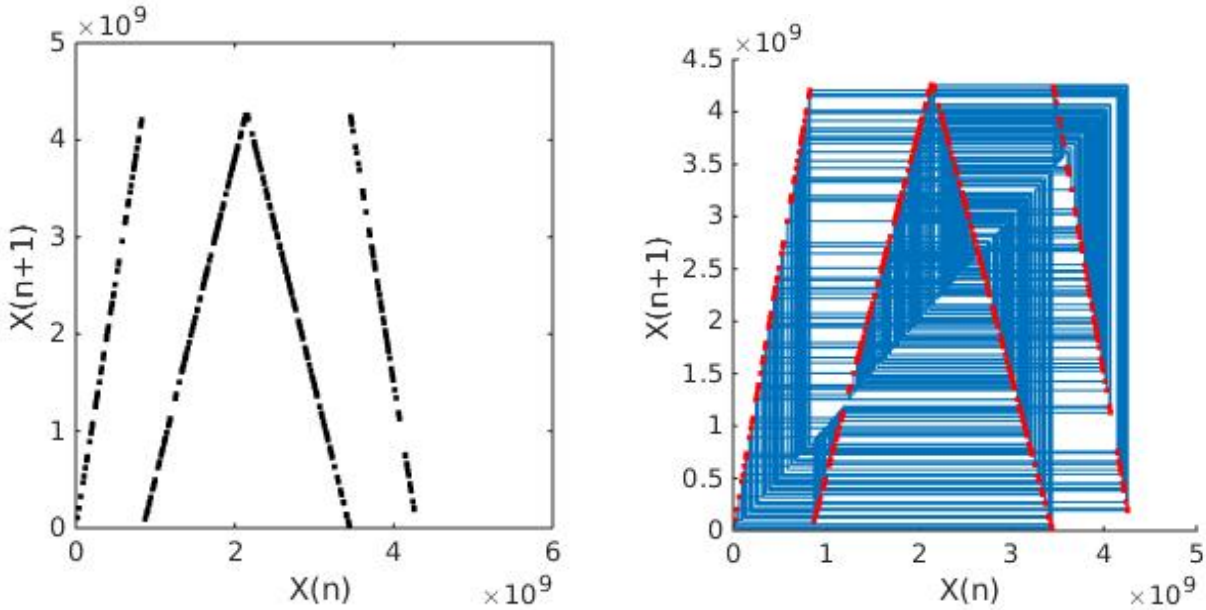
Histogram and Chi-square test analysis Fig. 3.14 represents the histogram of a sequence X_P generated by the PWLCM. Visually, the distribution of the sequence X_P looks uniform.

The experimental value of the Chi-square test χ^2_{exp} which is equal to 1219.97, is greater than the theoretical value χ^2_{th}, equals 1073.64. Therefore, the distribution of the discrete PWLCM map is non-uniform. Also, χ^2_{exp} is higher than that of the Skew Tent map. Thus, the Skew Tent map has better uniform distribution than the PWLCM map. In fact, studies have shown that the periodicity of the Skew Tent map is greater than the periodicity of the PWLCM, this may explain the fact that the Skew Tent map has a better uniformity.

Correlation analysis We give in Fig. 3.15, the auto-correlation function of sequence X_P and a zoom of the autocorrelation on 200 samples of this sequence. Fig. 3.16 presents the Cross-correlation functions of sequences X_P and $X_{P'}$ generated by the PWLCM map, and a zoom of the auto and cross-correlation. We note that the cross-correlation function is very low (maximum value = 0.025). Consequently, there is no correlation between the generated sequences, that are produced using slightly different seeds. The correlation coefficient $\rho_{X_P,X_{P'}}$ which is equal 0.0028 confirms this result.

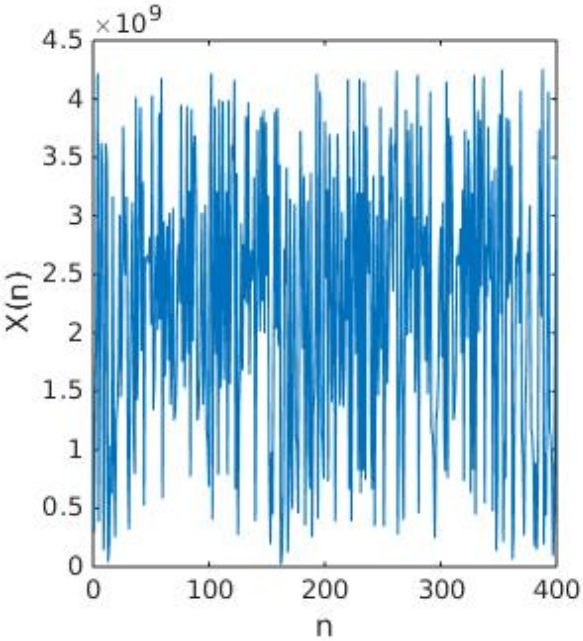
NIST test analysis We give in Fig. 3.17 and Table 3.5 the obtained NIST test results of a sequence X_P generated by the discrete PWLCM map. Some sub-tests have not passed but they are close to the acceptance threshold. Also, we remark that the PWLCM map has better security performance than the other studied chaotic maps Skew Tent and Logistic map.

Computing performance of the PWLCM map Table 3.6 shows the PWLCM map's computing performance. Compared to the other chaotic maps, the PWLCM is slower than the Skew Tent and the Logistic map.



(a) Phase space

(b) Attractor



(c) Discrete variation

Figure 3.13 – Phase space trajectory, attractor and discrete variation of sequence $X_P(n)$ generated by the discrete PWLCM map.

3.4 Performance Evaluation of some disturbed chaotic maps

3.4.1 Description of the used perturbation technique

The discretizing process and the usage of a finite precision N bits cause a degradation of the chaotic signals and may cause some state circles [143], which degrades the chaotic properties. Indeed, for a system of N bits, the maximum number of different chaotic levels is smaller than 2^N. The limited space values (assumed to be infinite for analogue chaos) causes periodic cycles of the different chaotic orbits, each having a maximum length necessarily less than 2^N [239, 78]. Moreover, for each initial condition, we have a chaotic orbit formed generally of two parts: a transient branch of length l and a cycle of period c. The

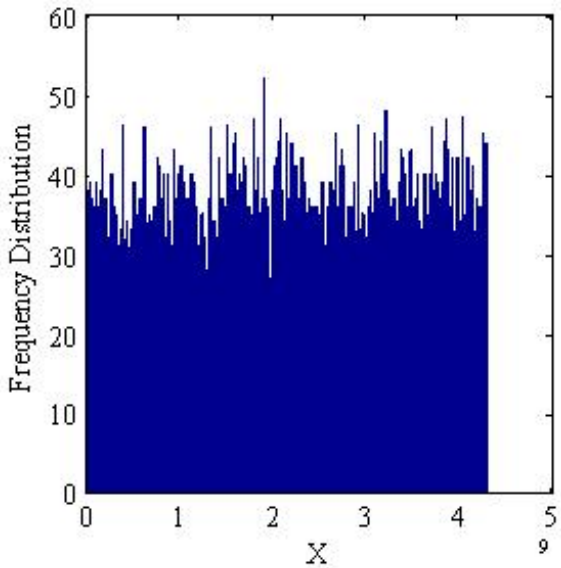
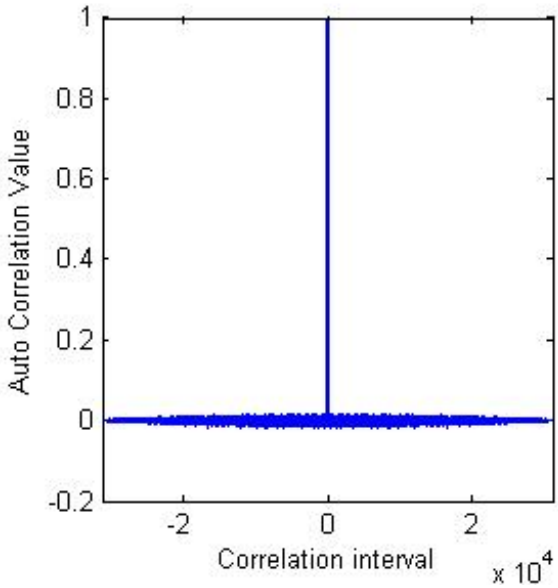
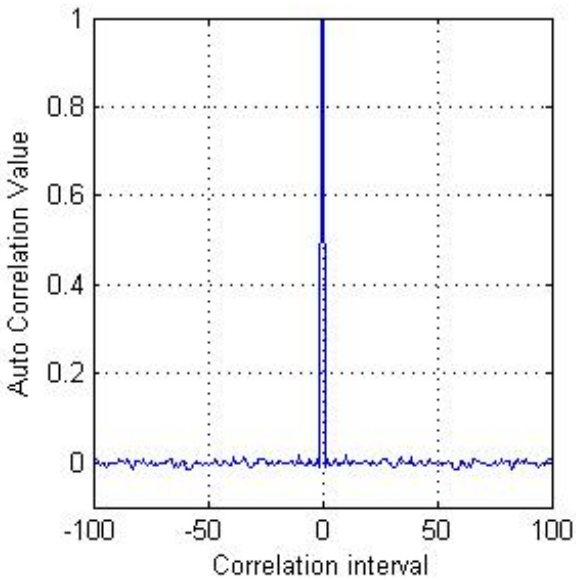


Figure 3.14 – Histogram of sequence X_P generated by the discrete PWLCM map.



(a) Auto-correlation of sequence X_P

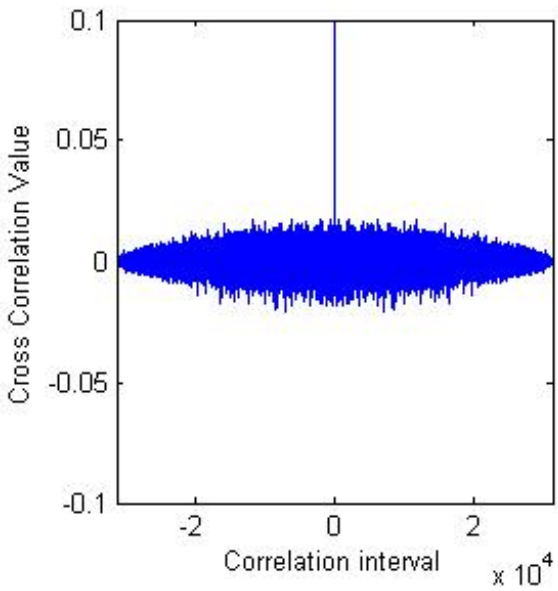


(b) A zoom of the autocorrelation on 200 samples of sequence X_P

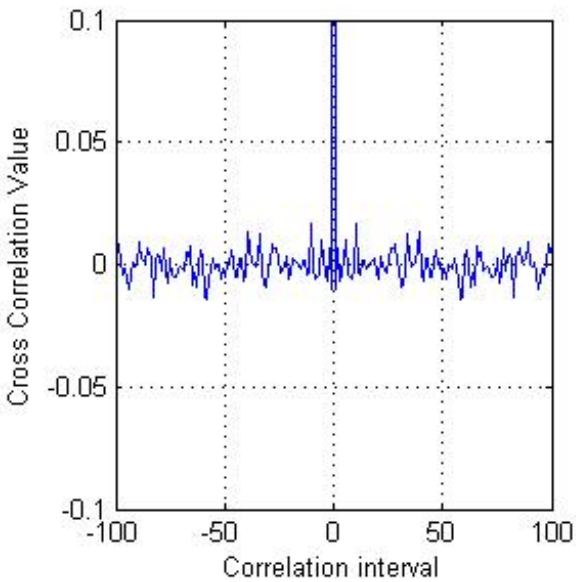
Figure 3.15 – Auto-correlation of sequence X_P generated by the PWLCM map.

general scheme of a chaotic orbit of length $o = l + c$ is presented in Fig. 3.18.

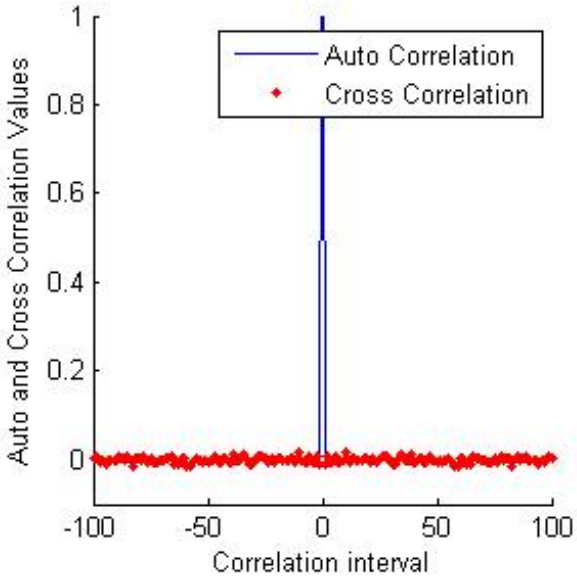
In order to decrease the degradation and to circumvent the effect of finite precision on chaotic signal, we use a perturbation technique. This technique permits to increase the length of the cycles and to impose a minimum length of cycle, depending directly on the disturbing signal. The perturbation technique results in the fact that no stable cycle exists, ie, if the chaotic system describes a given cycle at a given time, it can, by application of a perturbation, leave this cycle immediately to go to another cycle. Fig. 3.19 shows the principle of the perturbation technique. The disturbed structure is composed of a chaotic map, an XOR function and a disturbing generator. A good candidate for the generation of disturbing sequences is the Linear Feedback Shift Register (LFSR), whose role is to disrupt the chaotic orbit, thus allowing it to reach a new orbit.



(a) Cross-correlation of sequences X_P and $X_{P'}$



(b) A zoom of the cross-correlation on 200 samples of sequences X_P and $X_{P'}$



(c) A zoom of the cross-correlation of sequences X_P and $X_{P'}$ and of the auto-correlation of sequence X_P

Figure 3.16 – Cross-correlation functions of sequences X_P and $X_{P'}$ generated by the PWLCM map.

The choice of the disturbing sequence is made according to the following rules: it should have a long controllable cycle length and a uniform distribution; It should not degrade the good statistical properties of the chaotic dynamics, so the amplitude of the disturbing signal must be much smaller than that of the chaotic signal [78, 80]. We choose the disturbing sequence by choosing a polynomial of perturbations. We list in Appendix B the list of disturbance polynomials.

In the next sections, we study the security performance of disturbed chaotic maps: Logistic map, Skew Tent and PWLCM which include a perturbation technique. Also, we present their computing performance.

3.4.2 Performance Evaluation of the disturbed Logistic map

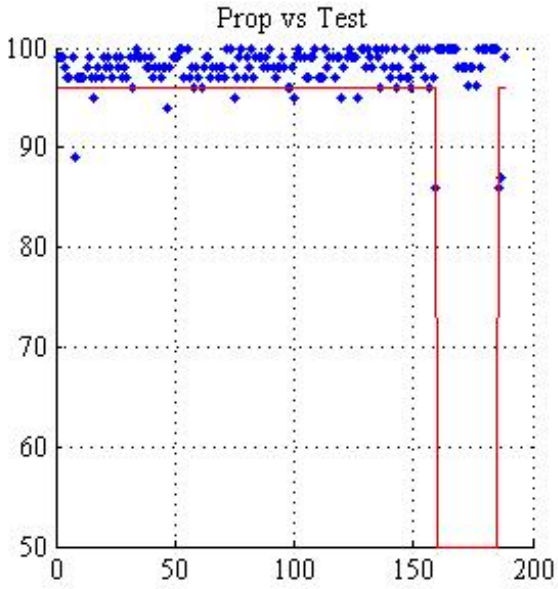


Figure 3.17 – NIST test results of the PWLCM map.

Test	P-value	Proportion
Frequency test	0.994	98.000
Block-frequency test	0.456	100.000
Cumulative-sums test	0.856	97.000
Runs test	0.924	100.000
Longest-run test	0.720	98.000
Rank test	0.616	97.000
FFT test	0.040	93.000
Non-periodic-templates	0.482	98.838
Overlapping-templates	0.964	98.000
Universal	0.868	100.000
Approximty entropie	0.868	93.000
Random-excursions:	0.266	97.955
Random-excursions-variant	0.308	99.495
Serial test	0.269	93.500
Linear-complexity	0.046	99.000

Table 3.5 – P-values and Proportion results of NIST test for the PWLCM map.

Generation Time (μs)	514.98
Bit Rate (Mbits/s)	1941.82
NCpB	10

Table 3.6 – Computing performance of the PWLCM map.

Security Performance Evaluation In order to carry out the various cryptographic tests, we choose the polynomial number 24 to generate different sequences $X_L(n)$. We recall that for the different tests, the size of a generated sequence, apart from the NIST test (which requires 100 sequences) and the Chi-square test (320 sequences), is equal to 31250 samples. We give in Fig. 3.20 and Table 3.7 the mapping, histogram, correlation and NIST test results of a sequence $X_L(n)$ generated by the disturbed Logistic map.

We notice that the phase space, given in Fig. 3.20a, looks like that of a sequence generated by the

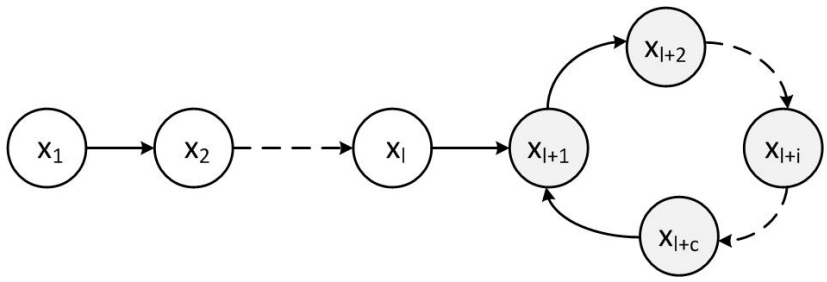


Figure 3.18 – General pseudo chaotic orbit of length $o = l + c$.

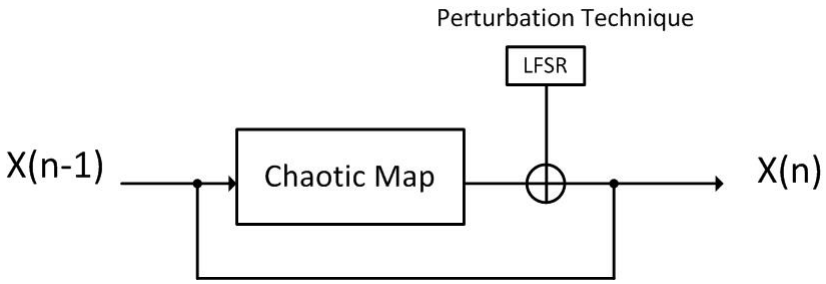


Figure 3.19 – Scheme of generating a pseudo-random sequence by a chaotic map using a perturbation technique.

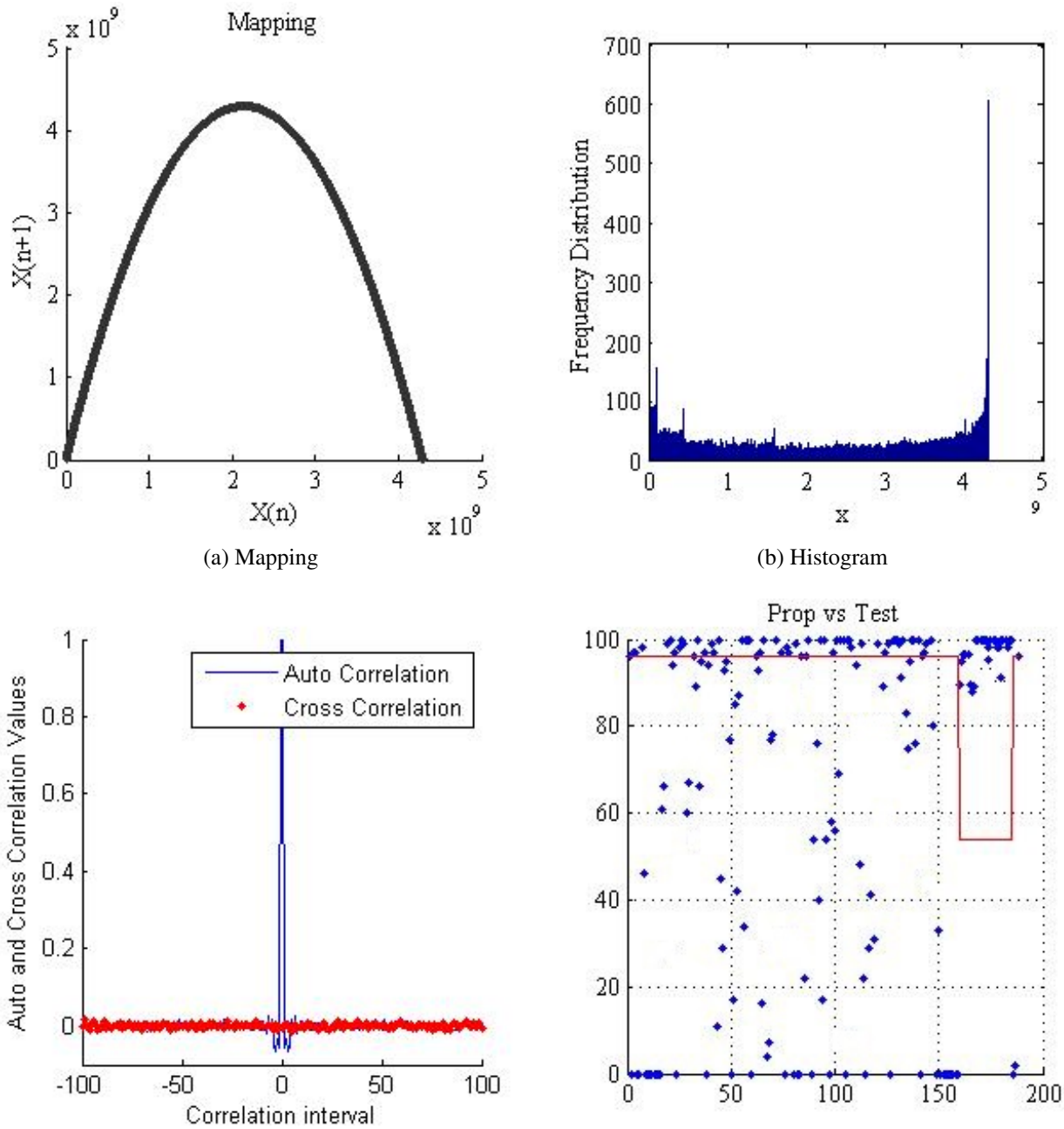
Test	**P-value**	**Proportion**
Frequency test	0.000	98.000
Block-frequency test	0.000	0.000
Cumulative-sums test	0.000	97.500
Runs test	0.000	0.000
Longest-run test	0.000	0.000
Rank test	0.898	100.000
FFT test	0.000	43.000
Non-periodic-templates	0.041	61.304
Overlapping-templates	0.000	0.000
Universal	0.000	0.000
Approximty entropie	0.000	0.000
Random-excursions:	0.002	90.367
Random-excursions-variant	0.400	99.901
Serial test	0.000	2.000
Linear-complexity	0.103	100.000

Table 3.7 – P-values and Proportion results of NIST test for the disturbed Logistic map.

non-disturbed Logistic map and clearly shows the signature relating to the map. This indicates that the perturbation technique has no effect on the phase space. This remark remains true for the phase space of the other disturbed Skew Tent and PWLCM maps.

Visually, the histogram is non-uniform (see Fig. 3.20b). This non-uniformity is confirmed by the Chi-square test. Indeed, the experimental value is equal to 925222 and it still higher than the theoretical one. But, it is less than the experimental value of an undisturbed sequence, i.e the uniformity is improved when using a perturbation technique.

Fig. 3.20c gives a zoom of the cross-correlation of sequences X_L and $X_{L'}$ and a zoom of the auto-correlation of sequence X_L. Obtained results shows that the generated sequences X_L and $X_{L'}$ possess good auto and cross correlation properties. This is confirmed by $\rho_{X_L X_{L'}}$ with is equal to 0.002.



(a) Mapping

(b) Histogram

(c) Correlation

(d) NIST Test

Figure 3.20 – Mapping, histogram, correlation and NIST test results of sequence generated by the disturbed Logistic map.

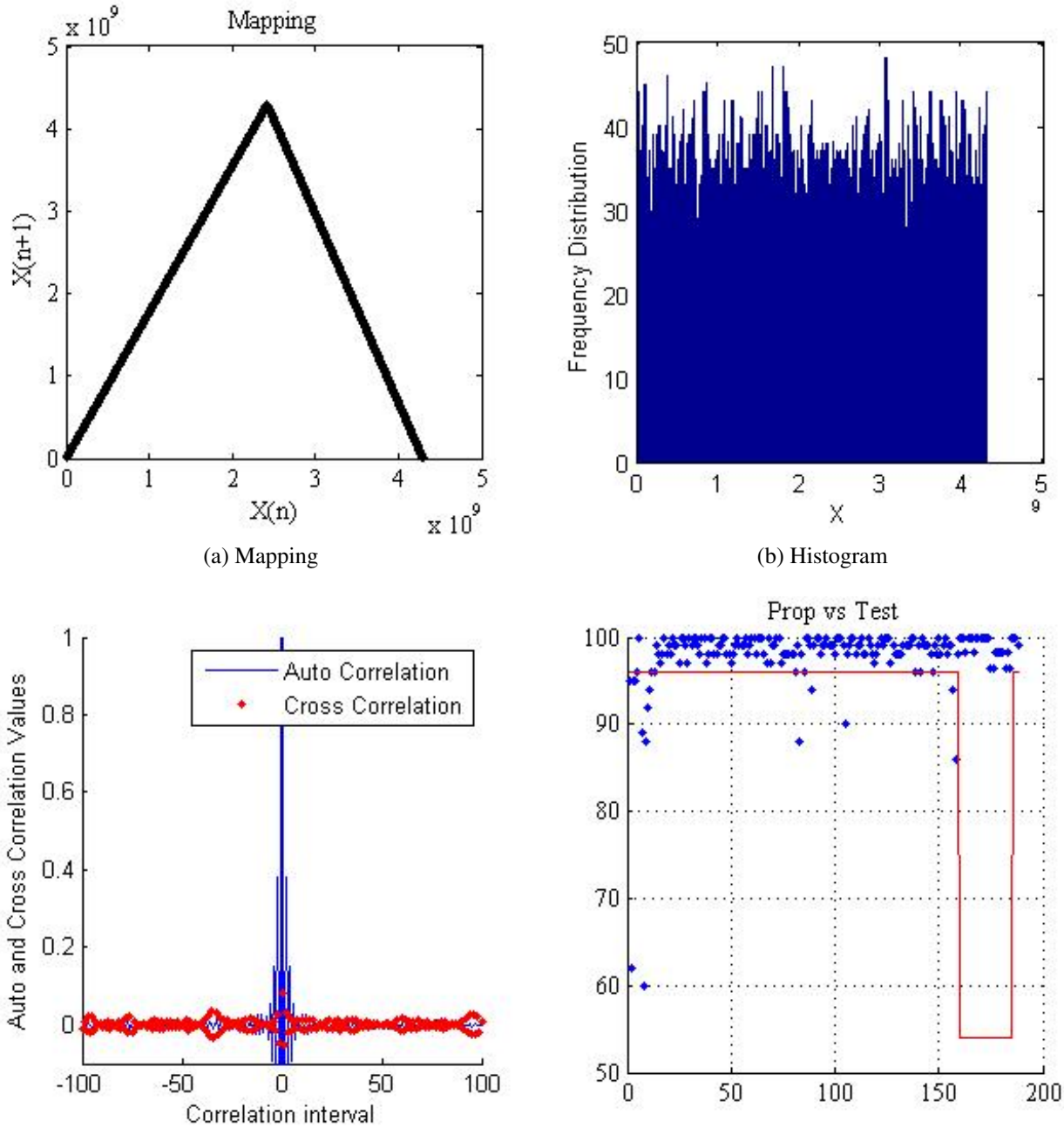
Computing Performance Evaluation In Table 3.8, we present the computing performance measures of the disturbed Logistic map. The additional perturbation technique decreases the computing performance of the map.

Generation Time (μs)	319.66
Bit Rate (Mbits/s)	3128.32
NCpB	7

Table 3.8 – Computing performance of the disturbed Logistic map.

3.4.3 Performance Evaluation of the disturbed Skew Tent map

Security Performance Evaluation We choose the polynomial P = 12 to integrate the perturbation technique in the generation of the different sequences. We show in Fig. 3.21 and Table 3.9, an example of the results obtained: (a) Mapping, (b) Histogram, (c) Correlation and (d) NIST Test.



(a) Mapping (b) Histogram

(c) Correlation (d) NIST Test

Figure 3.21 – Mapping, histogram, correlation and NIST test results of a sequence generated by the disturbed Skew Tent map.

All of these results show that the disturbed Skew Tent map using the perturbation technique has better cryptographic properties than the non-disturbed Skew Tent map. Indeed, the generated sequences are uniform (The experimental value of chi-square test equal to 1020.97 is less than the theoretical one which is equal to 1073.64. There are no correlation between sequences generated with slightly different secret keys ($\rho_{X_S X_{S'}}$= 0.0015). The number of passed NIST tests increases as compared to the number of tests passed from the ordinary map. This intensifies the importance of the perturbation technique in the generation of

Test	P-value	Proportion
Frequency test	0.616	95.000
Block-frequency test	0.000	60.000
Cumulative-sums test	0.252	94.000
Runs test	0.154	94.000
Longest-run test	0.456	98.000
Rank test	0.000	85.000
FFT test	0.000	54.000
Non-periodic-templates	0.484	98.108
Overlapping-templates	0.506	96.000
Universal	0.000	83.000
Approximty entropie	0.883	97.000
Random-excursions:	0.196	98.707
Random-excursions-variant	0.286	99.713
Serial test	0.380	99.000
Linear-complexity	0.554	100.000

Table 3.9 – P-values and Proportion results of NIST test for the disturbed Skew Tent map.

chaotic sequences.

Computing Performance Evaluation Table 3.10 shows the computing performance of the disturbed Skew Tent map. We note that the bit rate of a disturbed Skew Tent map is a little less than the bit rate of the Skew Tent map used alone.

Generation Time (μs)	448.94
Bit Rate (Mbits/s)	2227.46
NCpB	10

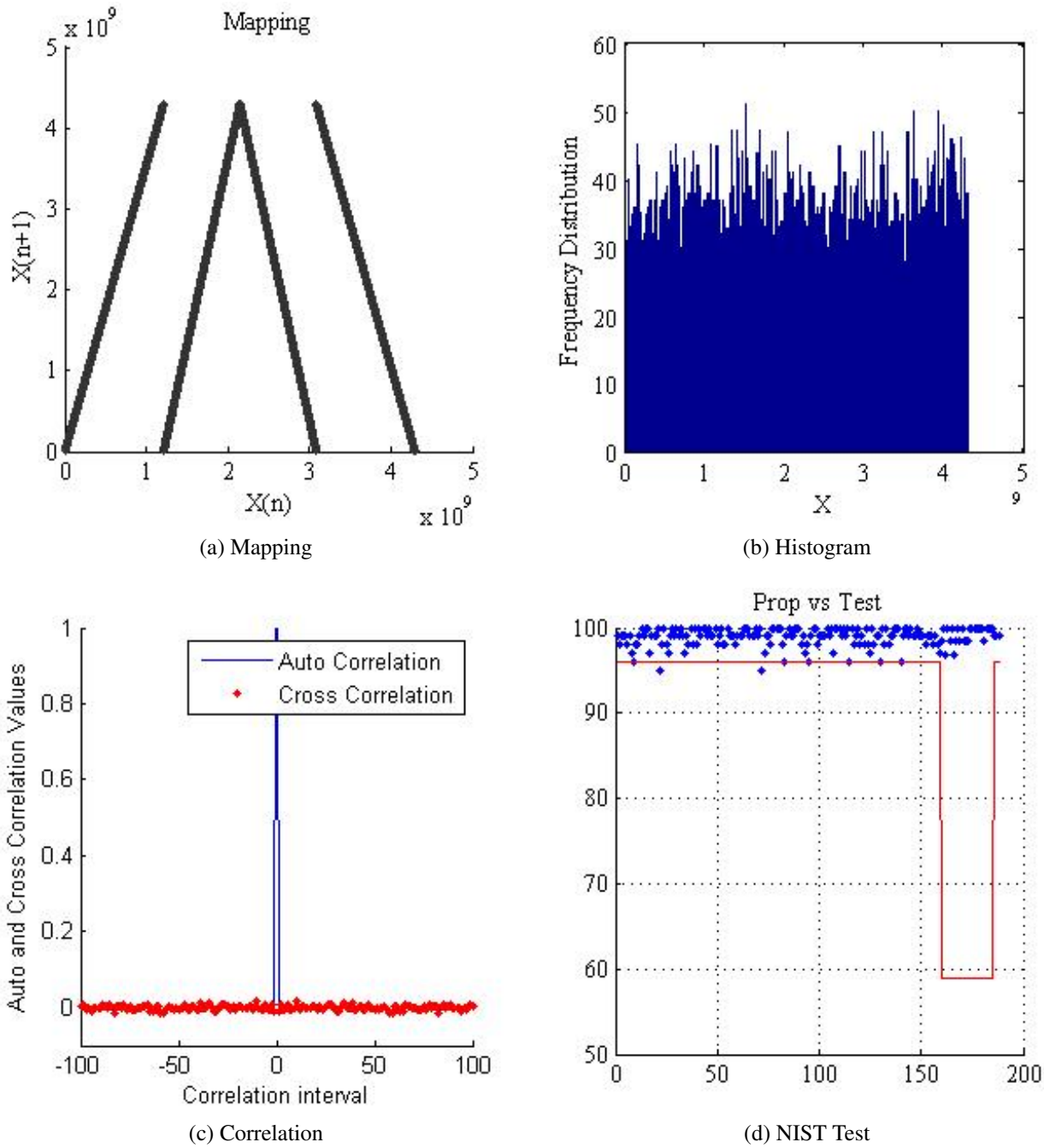
Table 3.10 – Computing performance of the disturbed Skew Tent map.

3.4.4 Performance Evaluation of the disturbed PWLCM map

Security Performance Evaluation We use the polynomial P number 7 to integrate the perturbation technique in the generation of the different sequences. As for the Logistic and Skew Tent maps, the perturbation technique improves the cryptographic performance of the produced sequences. This result is shown in Fig. 3.22 and Table 3.11.

Despite the used perturbation technique, according to Fig. 3.22b, the histogram and the Chi-square test demonstrated that the generated sequences are not uniform (Experimental value of Chi-square test equal to 1697.81 is higher than the theoretical value). However, the disturbed PWLCM map has a better NIST test and cross-correlation results ($\rho_{X_P X_{P'}}$ = -0.001).

Computing Performance Evaluation Table 3.12 gives the computing performance of the PWLCM map incorporating a perturbation technique. The PWLCM becomes slower than the Skew Tent map.



(a) Mapping

(b) Histogram

(c) Correlation

(d) NIST Test

Figure 3.22 – Mapping, histogram, correlation and NIST test results of a sequence generated by the disturbed PWLCM map.

3.5 Performance Evaluation of some disturbed chaotic maps including recursive technique

3.5.1 Description of the proposed non linear recursive structure

Another technique used to decrease the degradation caused by the finite precision usage and to improve the cryptographic performance of generated pseudo-chaotic sequences is the recursive structure.

Fig. 3.23 shows the principle scheme of a chaotic map inserted in a recursive structure. The produced sample $X(n)$ depends not only on the previous sample $X(n-1)$ but also on other previous samples. The number of samples to be dependent is chosen by the user, called "*delay*". In Fig. 3.23, we present a recursive

Test	P-value	Proportion
Frequency test	0.182	100.000
Block-frequency test	0.163	99.000
Cumulative-sums test	0.378	100.000
Runs test	0.154	98.000
Longest-run test	0.276	98.000
Rank test	0.225	100.000
FFT test	0.494	96.000
Non-periodic-templates	0.504	98.682
Overlapping-templates	0.401	99.000
Universal	0.122	99.000
Approximty entropie	0.033	94.000
Random-excursions:	0.359	97.461
Random-excursions-variant	0.422	99.132
Serial test	0.387	93.000
Linear-complexity	0.834	98.000

Table 3.11 – P-values and Proportion results of NIST test for the disturbed PWLCM map.

Generation Time (μs)	523.030
Bit Rate (Mbits/s)	1960.66
NCpB	11

Table 3.12 – Computing performance of the disturbed PWLCM map.

structure with number of delays equal to three.

To produce a sample $X(n)$, the system uses the previous samples, calculated as follows:

$$X(n-1) = U = \sum_{i=1}^{3} X_i \times K_i \tag{3.12}$$

where K_i are integers, $0 < K_i < 2^N$ and X_1 = X(n-1), X_2 = X(n-2), X_3 = X(n-3).

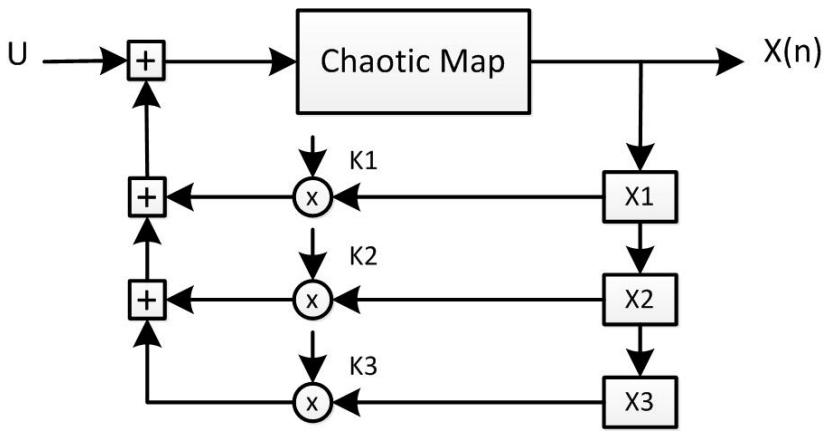


Figure 3.23 – Non linear recursive structure.

We study the performance of the non linear recursive structure with the Logistic, Skew Tent and PWLCM disturbed maps as non linear functions. Fig. 3.24 presents the general scheme of the non linear recursive structure based on disturbed chaotic map. We generate different chaotic sequences, using the

same previously chosen perturbation polynomials for each map and random coefficients K_i .

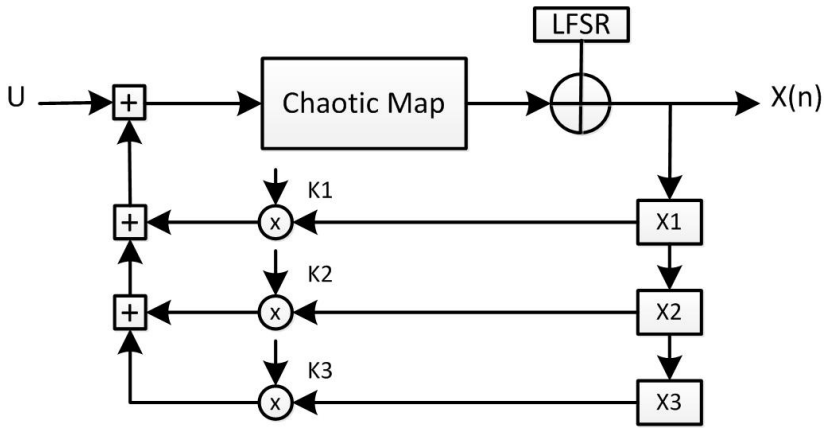
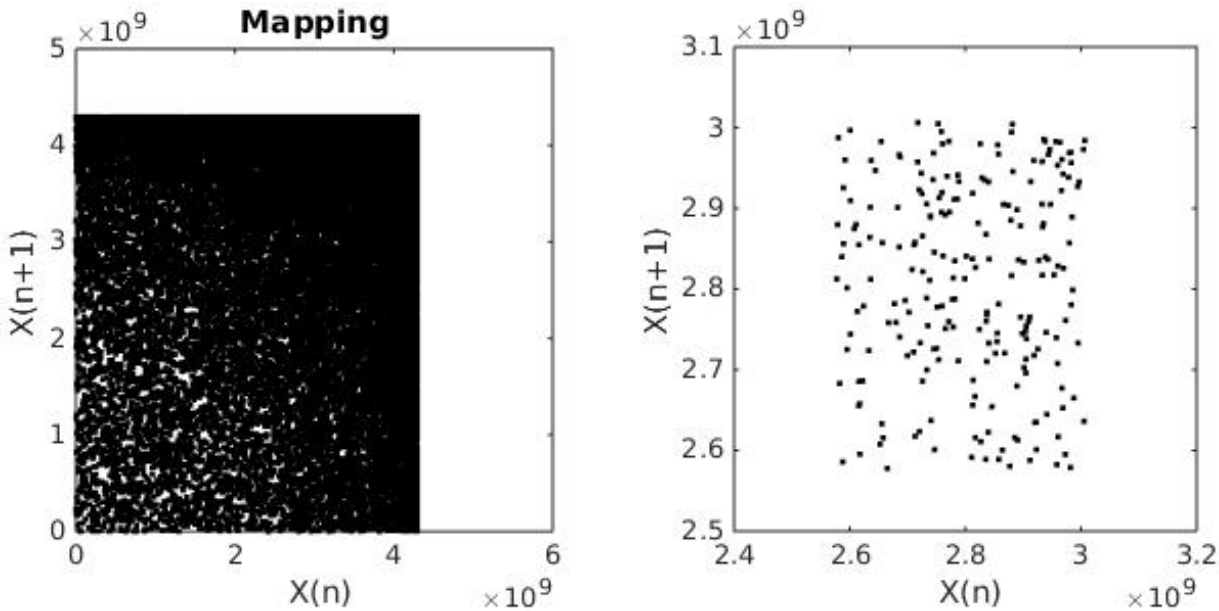


Figure 3.24 – Non linear recursive structure based on disturbed chaotic map.

3.5.2 Performance evaluation of the non linear recursive structure using the disturbed Logistic map

Security Performance Evaluation We draw in Fig. 3.25 the mapping and a zoom on the mapping of a generated sequence $X_L(n)$ by the structure of Fig. 3.24 with delay equal to 1 and using the disturbed Logistic map. The resulted mapping is random. Similar results of mapping are obtained for sequences generated with delays equal to 2 and 3. This shows that with the recursive structure, the mapping loses the signature of the chaotic map. Consequently, it is impossible to recognize the used map through the tracing of the phase space or mapping.



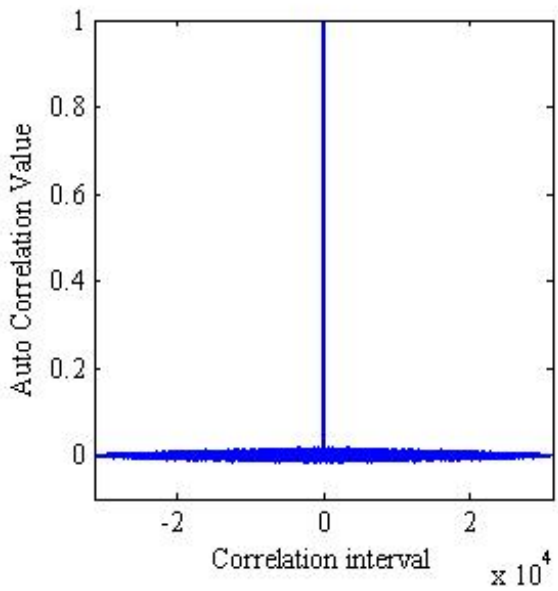
(a) Mapping

(b) Zoom on the mapping

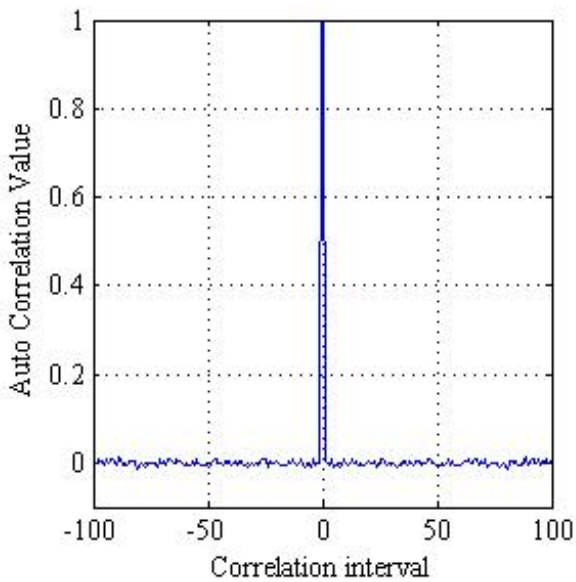
Figure 3.25 – Mapping and zoom on the mapping of a sequence $X_L(n)$ generated by the recursive structure with delay equal to 1, using the disturbed Logistic map.

Fig. 3.26 presents the autocorrelation and cross-correlation of two sequences X_L and $X_{L'}$ generated by the same structure with slightly different secret keys. Table 3.13 gives the correlation coefficients between

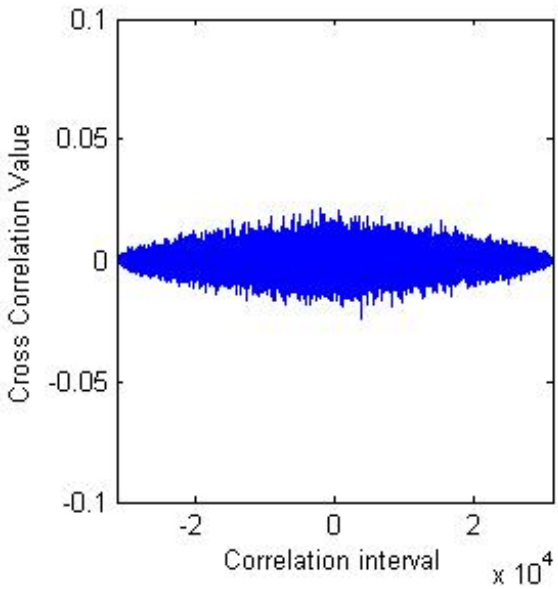
sequences X_L and $X_{L'}$. Obtained results show good correlation properties of the generated sequences X_L and $X_{L'}$.



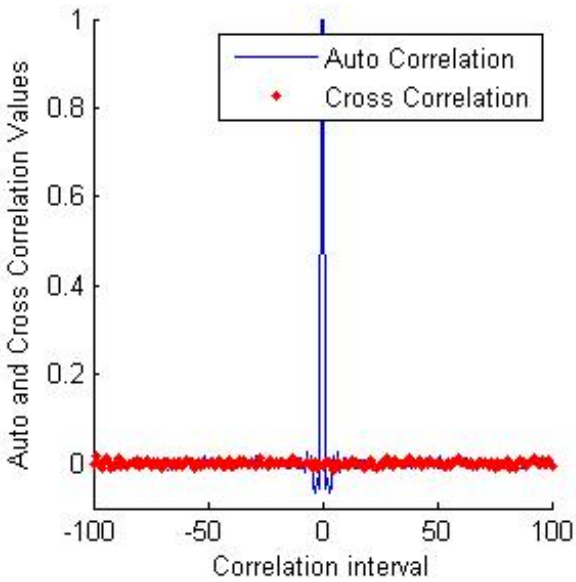
(a) Auto-correlation of sequence X_L



(b) A zoom of the autocorrelation on 200 samples of sequence X_L



(c) Cross-correlation of sequences X_L and $X_{L'}$



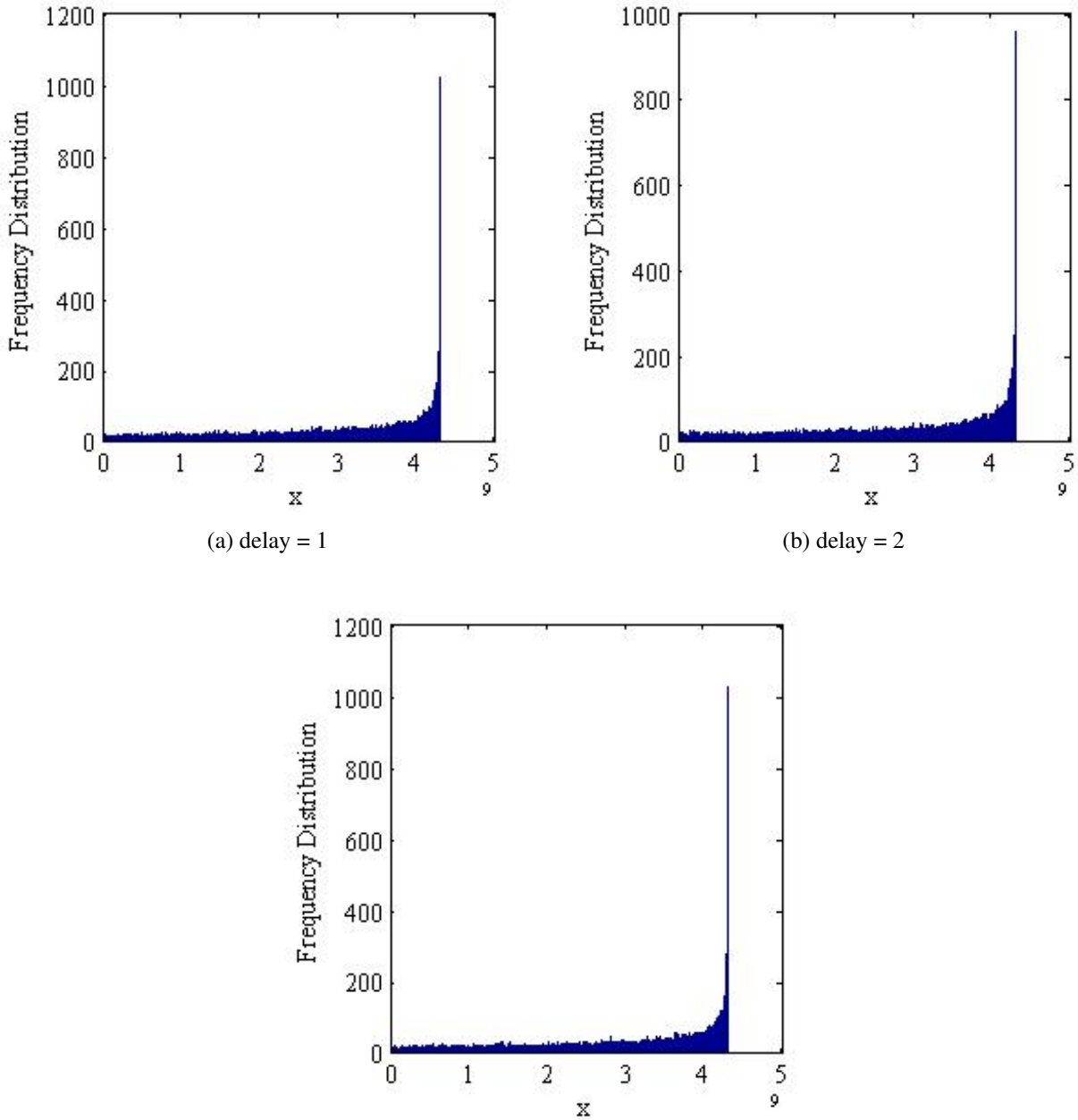
(d) A zoom of the cross-correlation of sequences X_L and $X_{L'}$ and of the auto-correlation of sequence $X_{L'}$

Figure 3.26 – Cross-correlation functions of sequences X_L and $X_{L'}$ generated by the structure of Figure 3.5.2 with delay equal to 1 and using the disturbed Logistic map.

Correlation coefficient	delay = 1	delay = 2	delay = 3
$\rho_{X_L,X_{L'}}$	0.0014	-0.0017	0.0013

Table 3.13 – Correlation coefficient values for sequences generated by the structure of Figure 3.5.2 using the disturbed Logistic map with delays equal to 1, 2 and 3.

We give in Fig. 3.27 the histograms of sequences generated with the perturbation technique in recursive structure with delays equal to 1, 2 and 3 respectively. The obtained histograms are visually non-uniform. This result is confirmed by the Chi-square test (see Table 3.14).



(a) delay = 1

(b) delay = 2

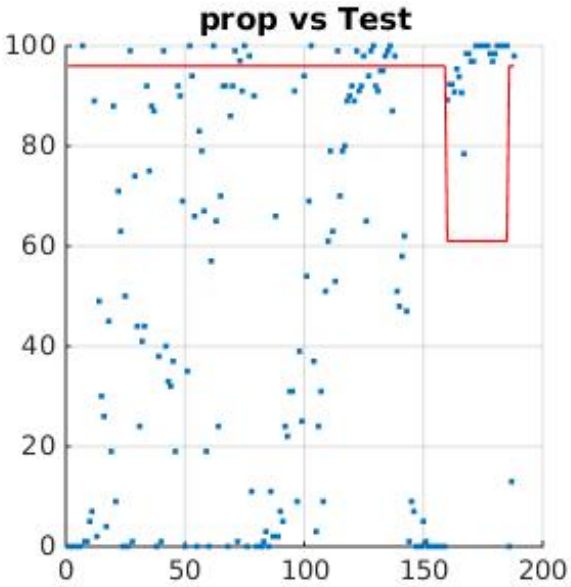
(c) delay = 3

Figure 3.27 – Histograms of sequences X_L generated by a disturbed Logistic map in a recursive structure with delays equal to 1, 2 and 3 respectively.

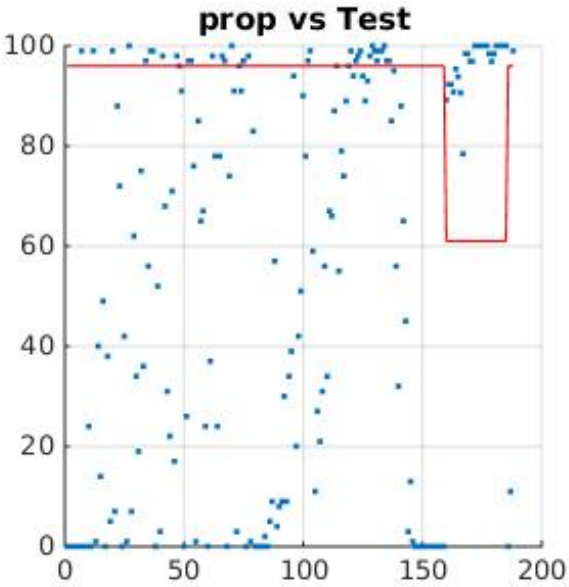
Chi-square value	delay = 1	delay = 2	delay = 3
χ^2_{th}	1073.64	1073.64	1073.64
$\chi^2_{\exp}$	17017171.47	17214833.41	17201737.77

Table 3.14 – Theoretical and experimental values of Chi-square test for sequences generated by the structure of Fig. 3.5.2 with delay equal to 1 and using the disturbed Logistic map.

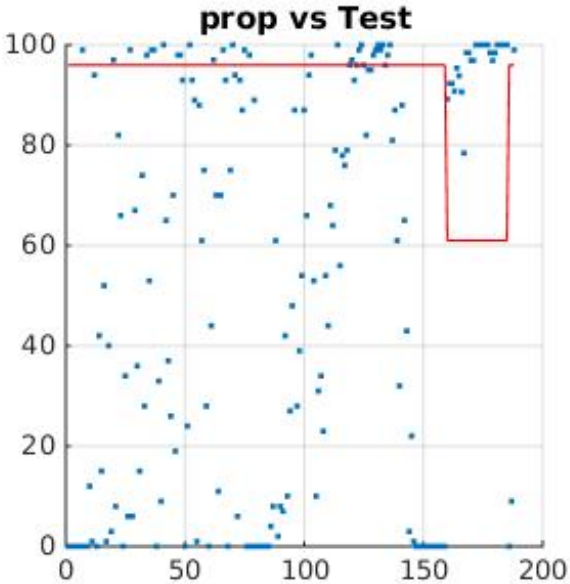
Fig. 3.28 and Table 3.15 presents the results of the NIST for sequences generated by the disturbed Logistic map using a perturbation technique in a recursive structure, with delay equal to 1, 2 and 3 respectively. Obtained results show that the perturbation technique and the recursion structure have improved cryptographic properties of the Logistic map.



(a) NIST test for delay = 1



(b) NIST test for delay = 2



(c) NIST test for delay = 3

Figure 3.28 – NIST test results for sequences generated by the structure of Fig. 3.5.2 with delays equal to 1, 2 and 3 and using the disturbed Logistic map.

Computing Performance Evaluation We study the speed of the disturbed Logistic map in recursive structure (see Table 3.16). The bit rate decreases as the delay increases. This is expected since the number of operations increases as the delay increases.

	delay = 1		**delay = 2**		**delay = 3**	
Test	**P-value**	**Proportion**	**P-value**	**Proportion**	**P-value**	**Proportion**
Frequency test	0.000	0.000	0.000	0.000	0.000	0.000
Block-frequency test	0.000	0.000	0.000	0.000	0.000	0.000
Cumulative-sums test	0.000	0.000	0.000	0.000	0.000	0.000
Runs test:	0.000	0.000	0.000	0.000	0.000	0.000
Runs test	0.000	0.000	0.000	0.000	0.000	0.000
Longest-run test	0.000	0.000	0.000	0.000	0.437	99.000
Rank test	0.596	100.000	0.786	99.000	0.952	100.000
FFT test	0.000	0.000	0.000	0.000	0.000	0.000
Non-periodic-templates	0.026	49.000	0.063	51.007	0.065	53.676
Overlapping-templates	0.000	0.000	0.000	0.000	0.000	0.000
Universal	0.000	0.000	0.000	0.000	0.000	0.000
Approximty entropie	0.000	0.000	0.000	0.000	0.000	0.000
Random-excursions:	0.002	90.367	0.180	92.000	0.502	100.000
Random-excursions-variant	0.400	96.145	0.600	99.000	0.681	100.000
Serial test	0.000	5.500	0.000	6.500	0.002	52.500
Linear-complexity	0.798	98.000	0.898	99.000	0.974	99.000

Table 3.15 – P-values and Proportion results of NIST test for the disturbed Logistic map in a recursive structure.

	delay = 1	delay = 2	delay = 3
Generation Time (μs)	418.830	420.30	421.93
Bit Rate (Mbits/s)	2387.6036	2379.2529	2370.0614
NCpB	7.71	7.73	7.76

Table 3.16 – Computing performance of the structure of Fig. 3.5.2 with delays equal to 1, 2 and 3 and using the disturbed Logistic map.

3.5.3 Performance evaluation of the non linear recursive structure using the disturbed Skew Tent map

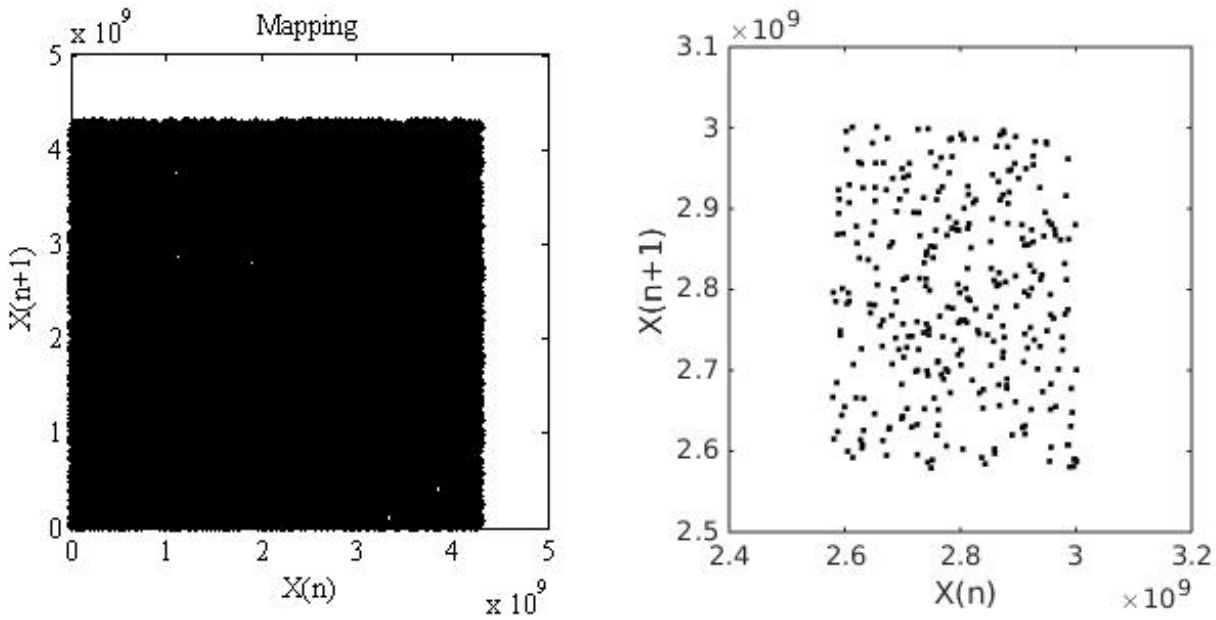
Security Performance Evaluation Fig. 3.29 gives the mapping and a zoom on this mapping of sequence $X_S(n)$ generated by the disturbed Skew Tent map used in a recursive structure with a delay equal to 1. The mapping seems random. This is due to the used recursive structure when generating sequences. It should also be noted that the mapping of the sequences generated with a delay equal to 2 and 3 is practically identical to the phase space of Fig. 3.29.

We show in Fig. 3.30 the auto and cross-correlation of two sequences X_S and $X_{S'}$ generated by the cited recursive structure with a delay equal to 1. The sequences possess good auto and cross-correlation properties (see also Table 3.17).

Correlation coefficient	delay = 1	delay = 2	delay = 3
$\rho_{X_S,X_{S'}}$	-0.0015	-0.002	0.0013

Table 3.17 – Correlation coefficient values for sequences generated by the structure of Fig. 3.5.2 using the disturbed Skew Tent map with delays equal to 1, 2 and 3.

We give in Fig. 3.31 the histograms of generated sequences produced with the same structure with



(a) Mapping (b) Zoom on the mapping

Figure 3.29 – Mapping and zoom on the mapping of a sequence $X_S(n)$ generated by the disturbed Skew Tent map used in a recursive structure with a delay equal to 1.

delays equal to 1, 2 and 3 respectively. Visually, generated sequences have uniform distribution.

Table 3.18 presents the theoretical and experimental values of the Chi-square test. We note that the sequences generated with a delay equal to 2 and 3 are uniform. Also, the uniformity of the sequence with a delay of 3 is the best. This demonstrate that, the more the number of delay increase, the more the uniformity is better.

Chi-square value	delay = 1	delay = 2	delay = 3
χ^2_{th}	1073.64	1073.64	1073.64
$\chi^2_{\exp}$	93533.54	1020.92	992.57

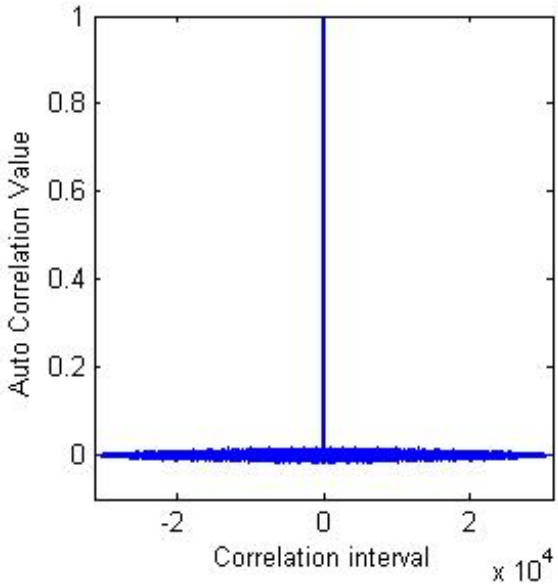
Table 3.18 – Theoretical and experimental values of Chi-square test for sequences generated by the structure of Fig. 3.5.2 with delay equal to 1 and using the disturbed Skew Tent map.

The results of the NIST test given in Fig. 3.32 and Table 3.19 show that, by integrating the perturbation technique and the recursion structure, the success rate of the various tests increases. Also, the more the number of delay increases, the more the number of tests that pass increases. Hence the interest of the perturbation technique and the recursive structure.

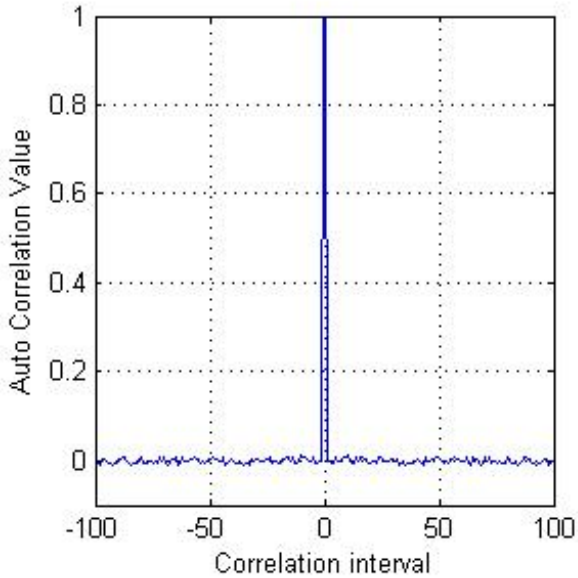
Computing Performance Evaluation We present in Table 3.20 the computing performance measures of a disturbed Skew Tent map in a recursive structure.

3.5.4 Performance evaluation of the non linear recursive structure using the disturbed PWLCM map

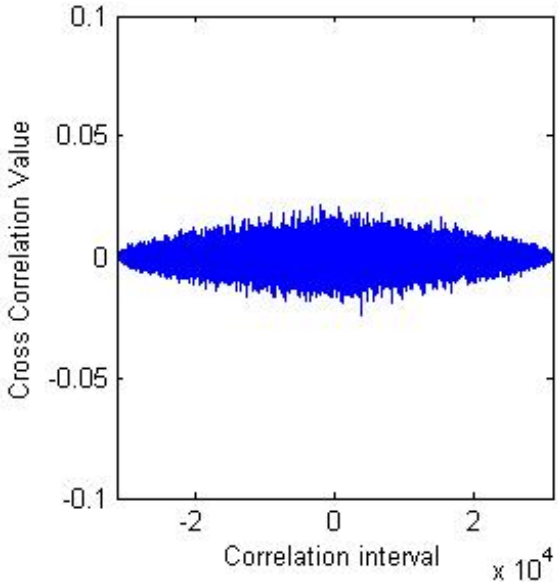
Security Performance Evaluation We present the mapping of a sequence $X_P(n)$ generated by a disturbed PWLCM map used in a recursive structure with a delay equal to 1 and a zoom on this mapping in Fig. 3.33. The obtained mapping is random.



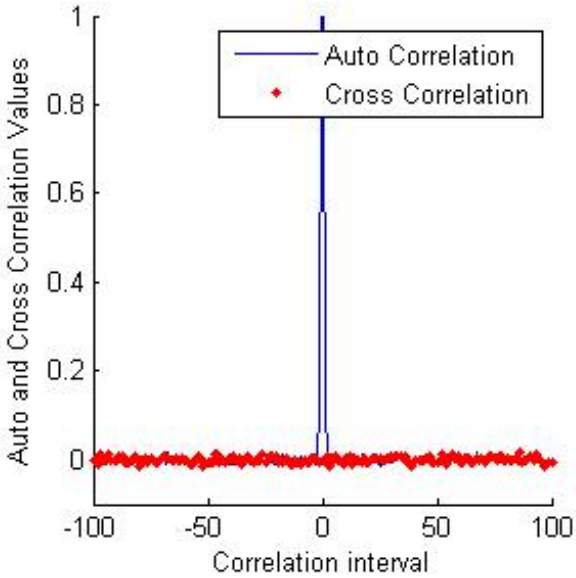
(a) Auto-correlation of sequence X_S



(b) A zoom of the autocorrelation on 200 samples of sequence X_S



(c) Cross-correlation of sequences X_S and $X_{S'}$



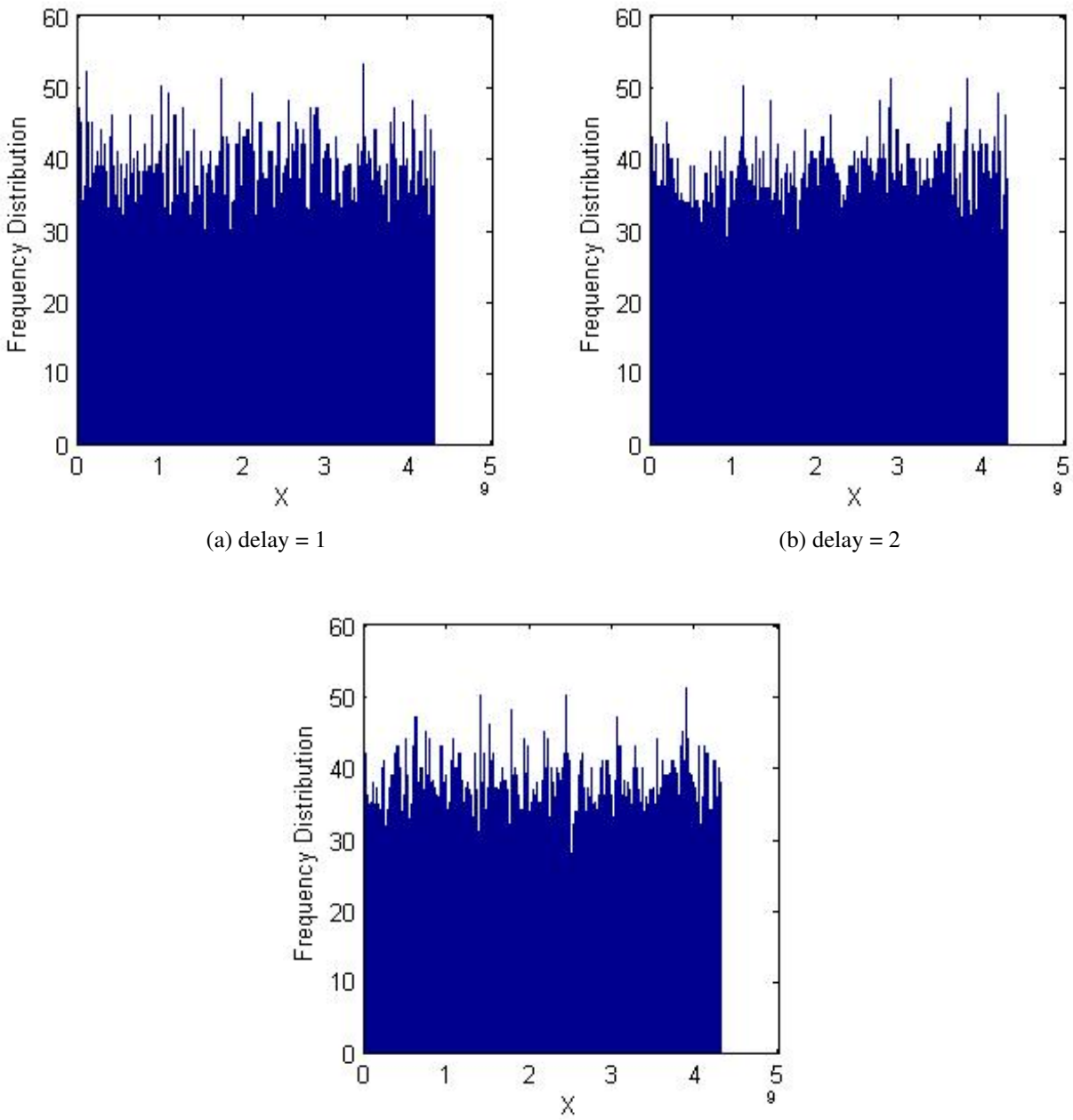
(d) A zoom of the cross-correlation of sequences X_S and $X_{S'}$ and of the auto-correlation of sequence $X_{S'}$

Figure 3.30 – Cross-correlation functions of sequences X_S and $X_{S'}$ generated by the structure of Fig. 3.5.2 with delay equal to 1 and using the disturbed Skew Tent map.

We draw in Fig. 3.34 the autocorrelation and cross-correlation of sequences X_P and $X_{P'}$ generated by the disturbed PWLCM map used in a recursive structure with a delay equal to 1. And we give in Table 3.21 the correlation coefficients values of sequences X_P and $X_{P'}$. Good correlation properties are obtained for the generated sequences X_P and $X_{P'}$.

We plot in Fig. 3.35 the histograms of sequences X_P generated by the disturbed PWLCM map used in a recursive structure with a delay equal to 1, 2 and 3 respectively. The obtained histograms seems to be uniform.

We calculate the theoretical and experimental values of the Chi-square test to assert the uniformity of the generated sequences. Obtained measures are given in Table 3.22.



(a) delay = 1

(b) delay = 2

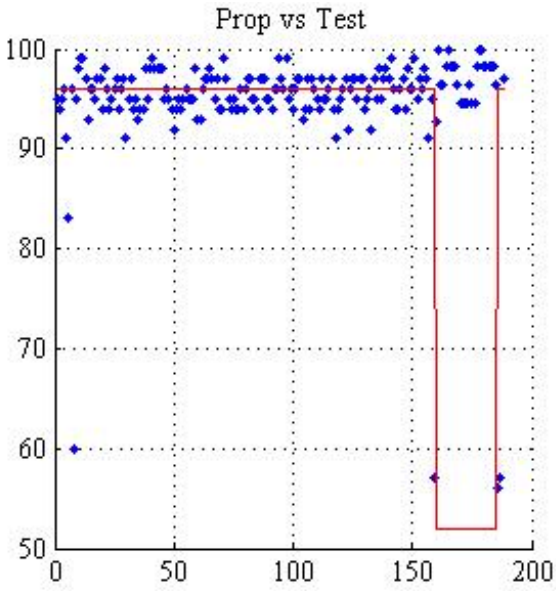
(c) delay = 3

Figure 3.31 – Histograms of sequences X_S generated by a disturbed Skew Tent map in a recursive structure with delays equal to 1, 2 and 3 respectively.

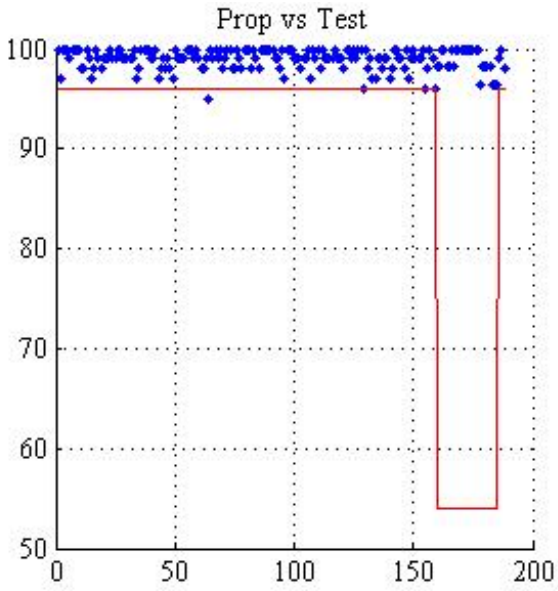
The generated sequences with a delay equal to 2 and 3 are uniform. Also, the uniformity is better for a delay of 3. This confirms that the recursion structure improves the uniformity of the chaotic sequences.

We present in Fig. 3.36 and Table 3.23 the results of NIST test applied for sequences generated by the same structure with different delays (1, 2 and 3). The sequence generated with a delay of 3 passes all NIST tests. This asserts the utility of perturbation perturbation and recursive structure in terms of cryptographic robustness.

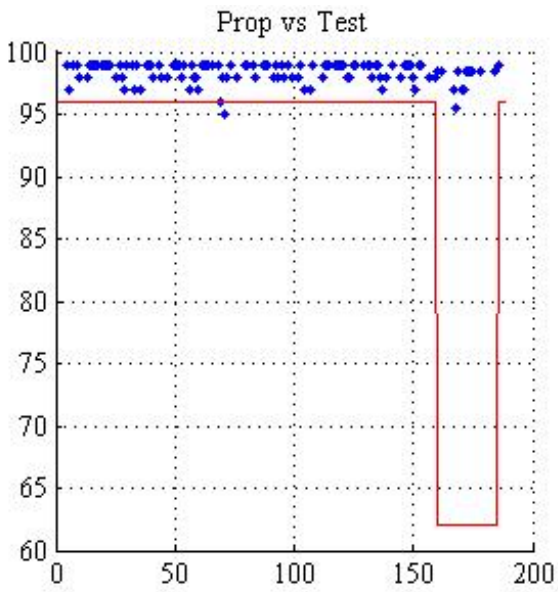
Computing Performance Evaluation Table 3.24 gives the computing performance measures performed to study the speed performance of a disturbed PWLCM map using a perturbation technique in a recursive structure.



(a) NIST test for delay = 1



(b) NIST test for delay = 2



(c) NIST test for delay = 3

Figure 3.32 – NIST test results for sequences generated by the structure of Fig. 3.5.2 with delays equal to 1, 2 and 3 and using the disturbed Skew Tent map.

3.6 Conclusion

In this chapter, we study the security and computing performance of some chaotic maps in particular Logistic map, Skew Tent and PWLCM map which present the basic elements of the proposed chaotic generator presented in Chapter 4. We first presented the common and standard tools for measuring the performance of chaotic sequences, in order to quantify and compare the cryptographic properties of the generated chaotic sequences including: phase space or mapping, auto and cross-correlation, histogram, Chi-square test and NIST test.

Then, we have presented the equations of different maps in real and discrete domain. We have also performed a series of statistical tests and we have presented the results obtained of these tests. We note that the Logistic map is the fastest one but it has the weakest cryptographic properties compared to the Skew

	delay = 1		**delay = 2**		**delay = 3**	
Test	**P-value**	**Proportion**	**P-value**	**Proportion**	**P-value**	**Proportion**
Frequency test	0.616	93.000	0.494	99.000	0.616	98.000
Block-frequency test	0.946	96.000	0.262	100.000	0.249	100.000
Cumulative-sums test	0.617	94.000	0.351	99.000	0.689	98.000
Runs test	0.172	95.000	0.122	100.000	0.350	98.000
Longest-run test	0.290	94.000	0.658	98.000	0.596	100.000
Rank test	0.637	99.000	0.851	99.000	0.596	100.000
FFT test	0.000	79.000	0.740	100.000	0.401	99.000
Non-periodic-templates	0.459	97.581	0.530	98.966	0.474	98.831
Overlapping-templates	0.384	90.000	0.456	99.000	0.898	100.000
Universal	0.213	96.000	0.071	98.000	0.437	99.000
Approximty entropie	0.000	81.000	0.145	99.000	0.419	97.000
Random-excursions:	0.458	99.091	0.454	99.254	0.619	99.180
Random-excursions-variant	0.437	99.495	0.257	98.259	0.512	99.180
Serial test	0.000	78.000	0.464	99.500	0.461	97.500
Linear-complexity	0.437	96.000	0.637	100.000	0.456	98.000

Table 3.19 – P-values and Proportion results of NIST test for the disturbed Skew Tent map in a recursive structure.

	delay = 1	delay = 2	delay = 3
Generation Time (μs)	507.23	508.94	529.34
Bit Rate (Mbits/s)	1971.49	1964.86	1889.14
NCpB	10.55	10.59	11.01

Table 3.20 – Computing performance of the structure of Fig. 3.5.2 with delays equal to 1, 2 and 3 and using the disturbed Skew Tent map.

Correlation coefficient	delay = 1	delay = 2	delay = 3
$\rho_{X_S,X_{S'}}$	0.0013	-0.001	0.0009

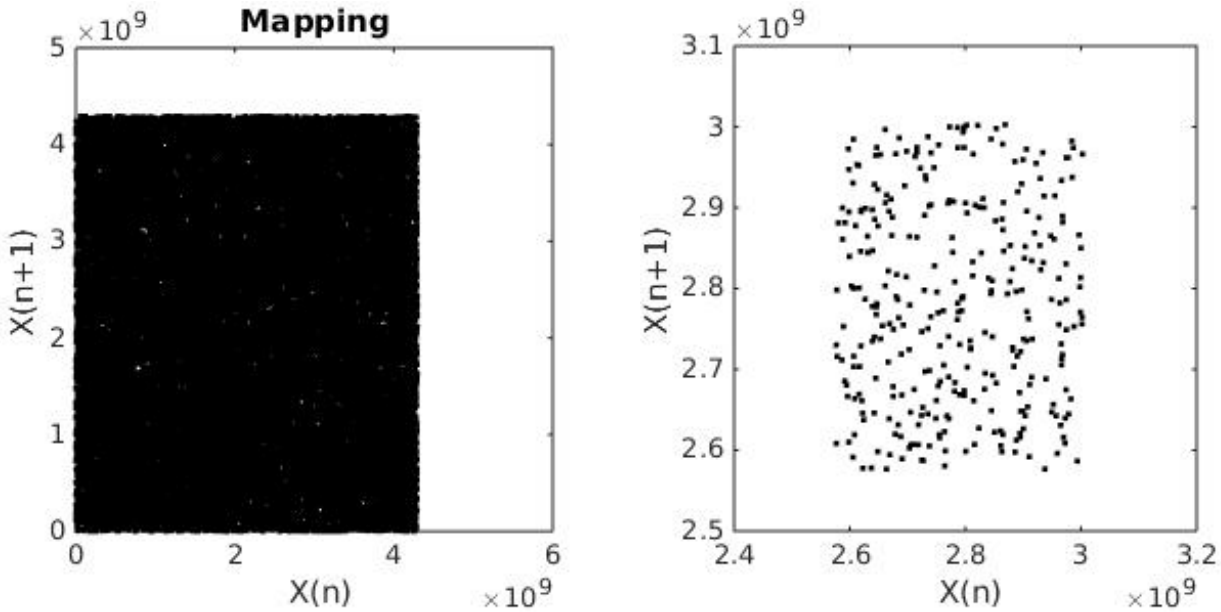
Table 3.21 – Correlation coefficient values for sequences generated by the structure of Fig. 3.5.2 using the disturbed PWLCM map with delays equal to 1, 2 and 3.

Chi-square value	delay = 1	delay = 2	delay = 3
χ^2_{th}	1073.64	1073.64	1073.64
χ^2_{exp}	4313.57	947.47	925.11

Table 3.22 – Theoretical and experimental values of Chi-square test for sequences generated by the structure of Fig. 3.5.2 with delay equal to 1 and using the disturbed PWLCM map.

Tent and PWLCM maps. The PWLCM map is characterized by its good cryptographic properties that are better than those of the Skew Tent map but it is less fast than the later.

Also, we have showed the importance of integrating a perturbation technique and a recursive structure when generating chaotic sequences. We presented a description for each technique and its effect on the robustness and time performance of each chaotic map. The two proposed techniques improve the cryptographic performance of the chaotic maps, but they increase the computation time of the sequences.



(a) Mapping

(b) Zoom on the mapping

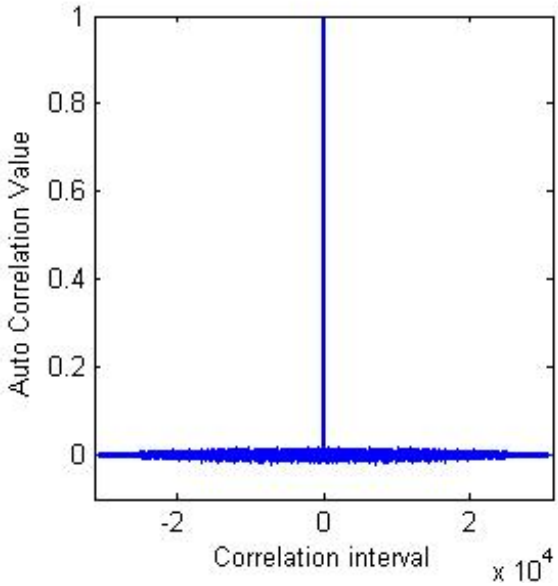
Figure 3.33 – Mapping and zoom on the mapping of a sequence $X_P(n)$ generated by the disturbed PWLCM map used in a recursive structure with a delay equal to 1.

	delay = 1		**delay = 2**		**delay = 3**	
Test	**P-value**	**Proportion**	**P-value**	**Proportion**	**P-value**	**Proportion**
Frequency test	0.182	98.000	0.779	99.000	0.996	98.000
Block-frequency test	0.419	99.000	0.554	99.000	0.081	97.000
Cumulative-sums test	0.378	100.000	0.593	99.000	0.720	97.500
Runs test	0.367	94.000	0.163	99.000	0.213	99.000
Longest-run test	0.494	92.000	0.012	100.000	0.616	100.000
Rank test	0.898	99.000	0.637	99.000	0.419	100.000
FFT test	0.000	78.000	0.779	99.000	0.163	100.000
Non-periodic-templates	0.457	97.824	0.487	99.027	0.500	98.865
Overlapping-templates	0.817	94.000	0.834	97.000	0.456	99.000
Universal	0.868	99.000	0.798	99.000	0.163	98.000
Approximty entropie	0.000	80.000	0.554	100.000	0.475	99.000
Random-excursions:	0.579	98.750	0.433	98.694	0.338	98.387
Random-excursions-variant	0.467	98.778	0.438	98.093	0.355	98.925
Serial test	0.000	78.000	0.748	98.000	0.647	100.000
Linear-complexity	0.983	100.000	0.276	100.000	0.679	98.000

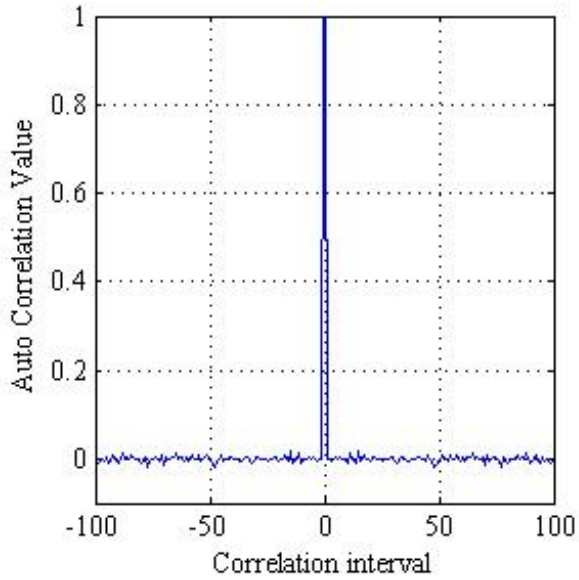
Table 3.23 – P-values and Proportion results of NIST test for the disturbed PWLCM map in a recursive structure.

	delay = 1	delay = 2	delay = 3
Generation Time (μs)	627.55	654.69	694
Bit Rate (Mbits/s)	1593.49	1527.44	1440.92
NCpB	13.05	13.62	14.44

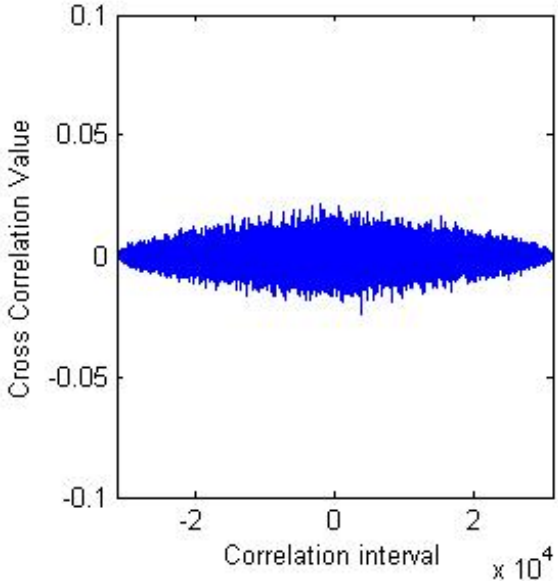
Table 3.24 – Computing performance of the structure of Fig. 3.5.2 with delays equal to 1, 2 and 3 and using the disturbed PWLCM map.



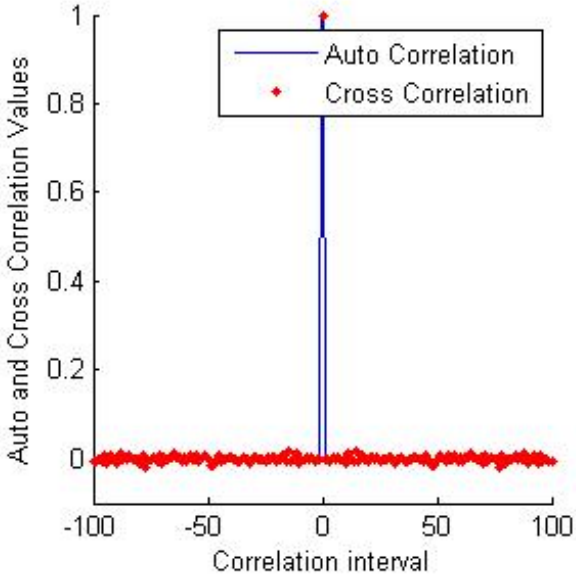
(a) Auto-correlation of sequence X_P



(b) A zoom of the autocorrelation on 200 samples of sequence X_P

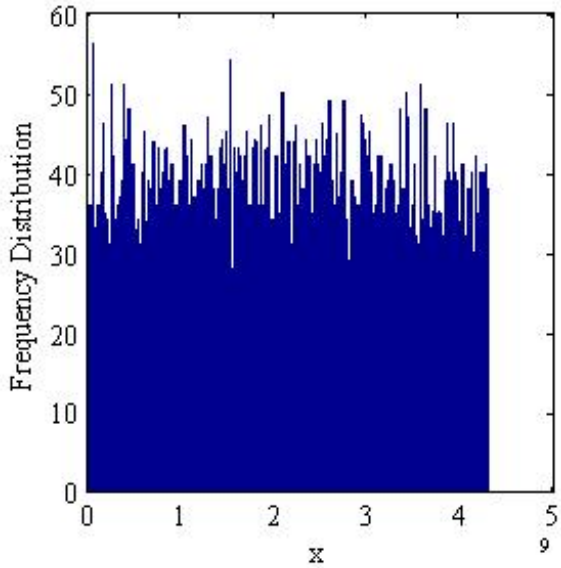


(c) Cross-correlation of sequences X_P and $X_{P'}$

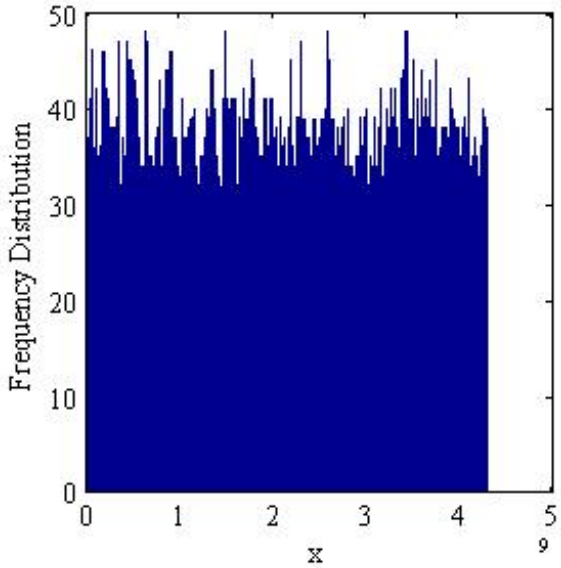


(d) A zoom of the cross-correlation of sequences X_P and $X_{P'}$ and of the auto-correlation of sequence $X_{P'}$

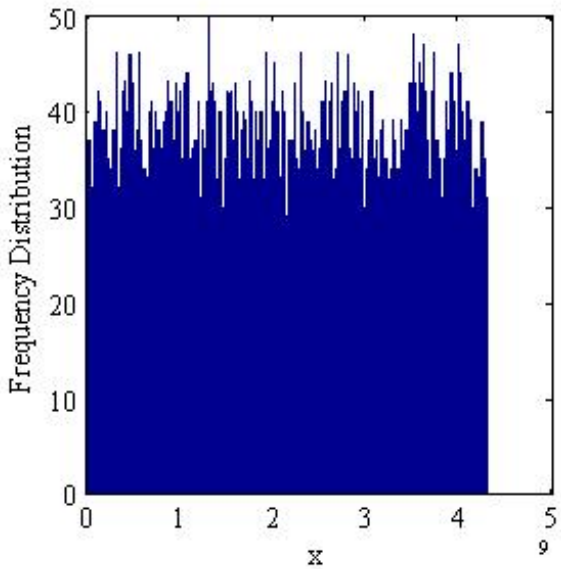
Figure 3.34 – Cross-correlation functions of sequences X_P and $X_{P'}$ generated by the structure of Fig. 3.5.2 with delay equal to 1 and using the disturbed PWLCM map.



(a) delay = 1

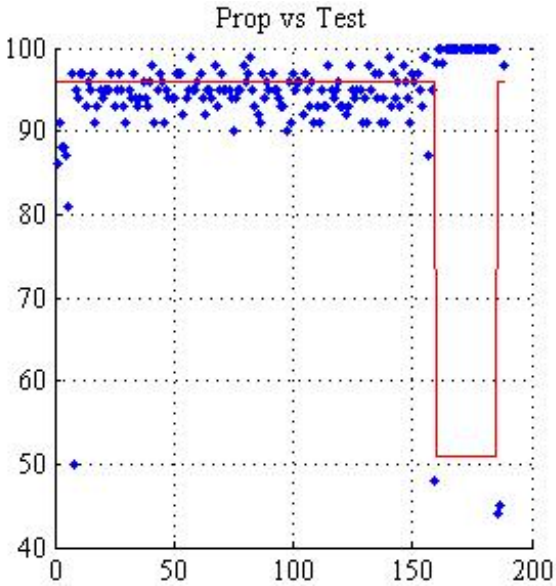


(b) delay = 2

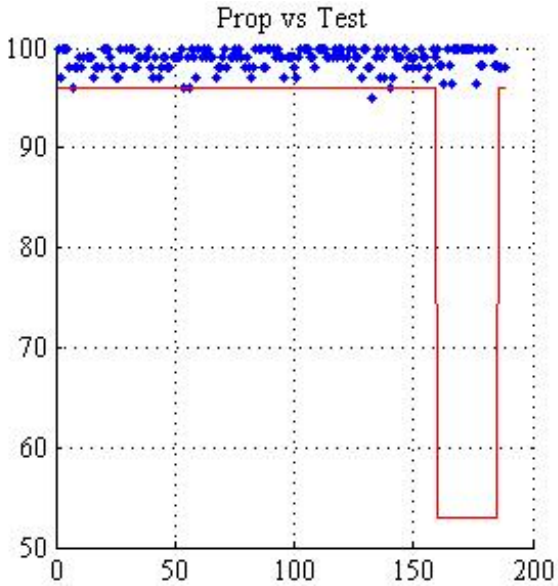


(c) delay = 3

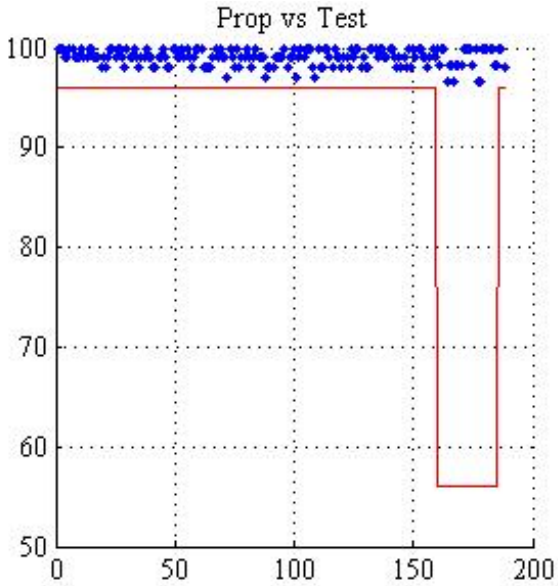
Figure 3.35 – Histograms of sequences X_P generated by a disturbed PWLCM map in a recursive structure with delays equal to 1, 2 and 3 respectively.



(a) NIST test for delay = 1



(b) NIST test for delay = 2



(c) NIST test for delay = 3

Figure 3.36 – NIST test results for sequences generated by the structure of Figure 3.5.2 with delays equal to 1, 2 and 3 and using the disturbed PWLCM map.

4

Design, implementation and analysis of Pseudo-chaotic Number Generators and Stream ciphers

4.1 Introduction

Nowadays, the increasing pervasiveness of technologies concerning the Internet of Things (IoT) and the fast development of digital technologies and communication networks, have given rise to dense traffic of information (documents, images, audio, videos...)[86]. Therefore, it is particularly important and essential to protect data transmission against attackers. Consequently, security of data transmission has been gaining more and more importance in the last decade and has been a subject of intense research [262, 219]. In this context, a growing number of crypto-systems to secure transmitted information have been developed [61, 228, 64, 89, 90]. Among them, chaos-based crypto-systems emerged to be promising. The idea of using digital chaotic systems to design cryptosystems has been extensively studied since 1989 [173]. Many research works have shown that chaos systems have many interesting properties such as ergodicity, similarity to random behavior, and sensitivity to initial conditions and parameters of the system, that make chaos a good candidate for use in information hiding and security systems. Also, chaotic maps present many desired qualities such as simplicity of implementation that leads to high encryption/decryption rates, and excellent security.

The robustness of any chaos-based crypto-system depends on the quality of the used Pseudo-Chaotic Number Generator (PCNG) which is a fundamental block in cryptography. The robustness of such PCNGs is crucial to ensure secure communication and to avoid all the various and existing attacks. Many PCNGs are proposed in the literature [92, 239, 192, 91]. Generated sequences must be absolutely random (practically very close to random) and have some properties such as: long cycle's length, cross-correlation near to zero, high linear complexity and fully distributed phase space. Existing PCNGs satisfy some of these properties, but most of them suffer from short cycle's length since they are performed in finite precision N which causes a dynamical degradation.

In this chapter, we propose and realize in an effective way three stream ciphers, based on three robust Pseudo-Chaotic Numbers Generators (PCNGs). The proposed crypto-systems are very secure, due to the use of chaotic coupling, swap and multiplexing techniques, while having a high speed performance. They can be used for real-time applications.

The three proposed crypto-systems have the same general structure of PCNGs, presented in Section 4.2.1. The main difference between these PCNGs appears in the Internal State and Output functions. The first proposed PCNG, called CM-PCNG, uses three weakly coupled chaotic maps: PWLCM, Skew Tent and Logistic and includes a multiplexing chaotic technique. Its architecture is described in Section 4.2.1. We illustrate the architecture of the second PCNG - DM-PCNG - in Section 4.2.1. In comparison with the architecture of CM-PCNG, the main difference lies in using a binary diffusion matrix on the chaotic coupling technique. Section 4.2.1 describes also the architecture of the third proposed PCNG, named CS-PCNG which is based on using two chaotic maps, namely PWLCM and SkewTent, and includes coupling and swap chaotic techniques. These proposed PCNGs are defined on finite numbers and implemented in C code. Their implementation in sequential and parallel programming is detailed in Section 4.2.2. In Section 4.2.3 and 4.2.4, we give the security and statistical analysis of the proposed PCNGs. Section 4.2.5 presents the computing performance measures of the two PCNGs in terms of average generation time, average Bit Rate (BR), and average Number of Cycles needed to generate one Byte (NCpB) according to the data size. The security analysis and statistical tests of the three proposed stream ciphers and their speed performance are presented in Section 4.3. Finally, Section 4.4 concludes this chapter.

4.2 Proposed Pseudo-Chaotic Number Generators

The proposed PCNGs consists of four main functions: IV-setup, Key-setup, Internal State and Output function. In the following, we describe in detail the general structure of the proposed PCNGs and their architectures. Then, we study the security and computing performance of these PCNGs.

4.2.1 Description of the general proposed structure of PCNGs

The general structure of the proposed PCNGs is presented in Fig. 4.1. It takes the parameters of the system (N and the number of samples Ns), a secret key "K" and a 32-bit initial vector "IV" as input, and as output, it generates pseudo-chaotic samples $X(n)$, n=1, 2, ..., each quantified on N = 32 bits.

The structure consists of four function blocks: IV-setup, Key-setup, Internal State and Output function. All the proposed PCNGs have the same general structure but differ in their internal state and slightly change in their Key-setup, IV-setup and Output function. Each function block will be detailed in the architectural description of the proposed PCNGs.

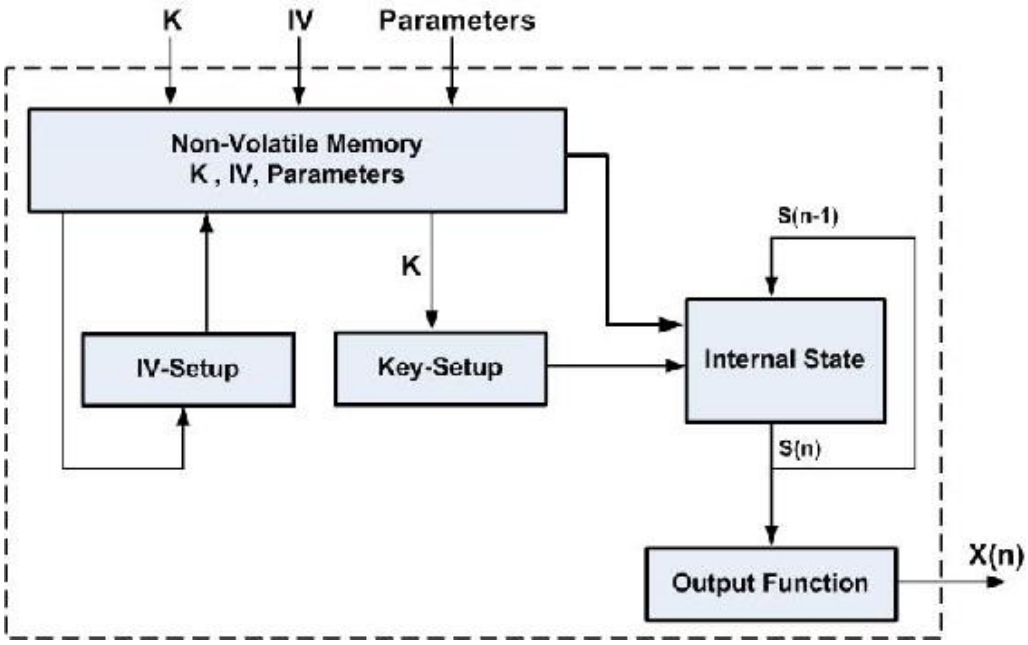


Figure 4.1 – General structure of the proposed PCNGs.

Architecture of the proposed CM-PCNG

The architecture of the first proposed chaotic generator CM-PCNG is given in Fig. 4.2. It uses three weakly coupled chaotic maps: PWLCM, Skew Tent and Logistic and includes a multiplexing chaotic technique [156][26][155][198][79][80].

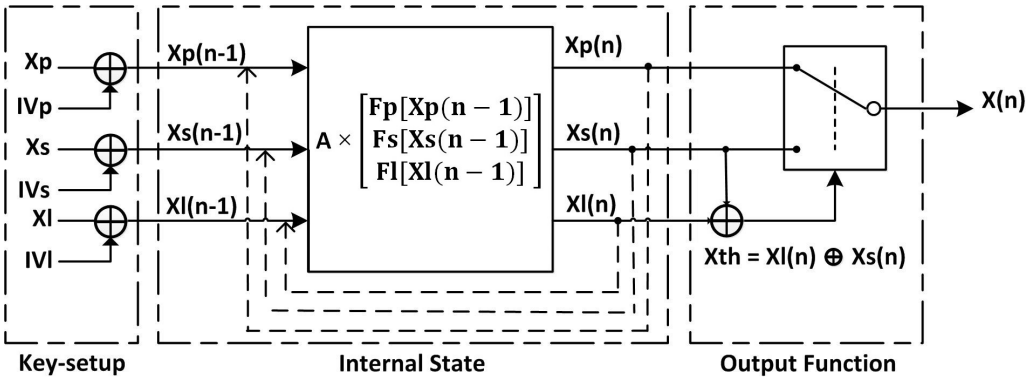


Figure 4.2 – Architecture of the proposed CM-PCNG.

The Key-setup function consists of two main parts. It takes the secret key K and the initial vector IV as input and calculates the initial values $Xp(0)$, $Xs(0)$ and $Xl(0)$ of the three chaotic maps: PWLCM, Skewtent and Logistic respectively.

The secret key of the system is formed by:

— the initial conditions Xp, Xs and Xl of the three chaotic maps: PWLCM, Skewtent and Logistic respectively, ranging from 1 to 2^N-1,

— the control parameter Pp and Ps of PWLCM and Skewtent maps, in the range $[1, 2^{N-1} - 1]$ and $[1, 2^N - 1]$ respectively,

— the parameters of the coupling matrix **A**, ε_{ij}, ranging from 1 to 2^k with k$\leq$ 5.

All the initial conditions, parameters and initial vector are chosen randomly from the file "`/dev/urandom`" which presents a special character file in the Linux environment that provides an interface with the Linux Pseudo-Random Number Generator (LRNG). LRNG is based on generating randomness from entropy of operating system events. It allows access to environmental noise collected from device drivers and other sources. The output of LRNG is used by internal kernel functionalities which use random bits and by calls to its application programming interface (API). Generated random data can also be used for other various purposes, such as generating random identifiers, computing TCP sequence numbers, producing passwords, and generating SSL private keys [104].

The initial values $Xp(0)$, $Xs(0)$ and $Xl(0)$ are calculated as follows:

$$\begin{cases} Xp(0) = Xp \oplus IVp \\ Xs(0) = Xs \oplus IVs \\ Xl(0) = Xl \oplus IVl \end{cases} \tag{4.1}$$

where

$$\begin{cases} IVp = lsb(IV) \\ IVs = L_{cir}[lsb(IV), 3] \\ IVl = L_{cir}[lsb(IV), 2] \end{cases} \tag{4.2}$$

with $\oplus$ denotes the XOR operator, $lsb(IV)$ is the 32 least significant bits of IV and $L_{cir}[S, q]$ performs the q-bits left circular shift on the binary sequence S.

The internal state function achieves the weak coupling of the chaotic maps and produces the future samples $Xp(n)$, $Xs(n)$ and $Xl(n)$ from which the output function, by using a chaotic switching technique, produces the output sequence $X(n)$ (see Fig. 4.2).

The system is governed by the following equation :

$$\begin{bmatrix} Xp(n) \\ Xs(n) \\ Xl(n) \end{bmatrix} = \mathbf{A} \times \begin{bmatrix} Fp[Xp(n-1)] \\ Fs[Xs(n-1)] \\ Fl[Xl(n-1)] \end{bmatrix}. \tag{4.3}$$

where **A** represents the weak coupling matrix:

$$\mathbf{A} = \begin{bmatrix} (2^N - \varepsilon_{12} - \varepsilon_{13}) & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & (2^N - \varepsilon_{21} - \varepsilon_{23}) & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & (2^N - \varepsilon_{31} - \varepsilon_{32}) \end{bmatrix}. \tag{4.4}$$

with ε_{ij} are the weakly coupling parameters, and $Fp[Xp(n-1)]$, $Fs[Xs(n-1)]$ and $Fl[Xl(n-1)]$ are the discrete functions of the chaotic maps PWLCM, Skew Tent and Logistic respectively defined in Chapter 3.

The obtained multiplexed samples of the sequence $X(n)$ are controlled by the chaotic sample $Xth(n)$ and a threshold T, as shown in Fig.4.2, and are defined as follows:

$$X(n) = \begin{cases} Xp(n), & \text{if } 0 < Xth(n) < T \\ Xs(n), & \text{otherwise} \end{cases} \tag{4.5}$$

Where $Xth(n) = Xl(n) \oplus Xs(n)$.

After the generation of all needed samples $X(n)$, the IV-setup function computes a new IV that will be used for the next running of the PCNG. The new IV is generated from the LRNG.

Architecture of the proposed DM-PCNG

The architecture of the second proposed generator called DM-PCNG is presented in Fig.4.3. In comparison with the previous architecture, the main difference lies in the internal-state function, which is based on a binary diffusion matrix **D**.

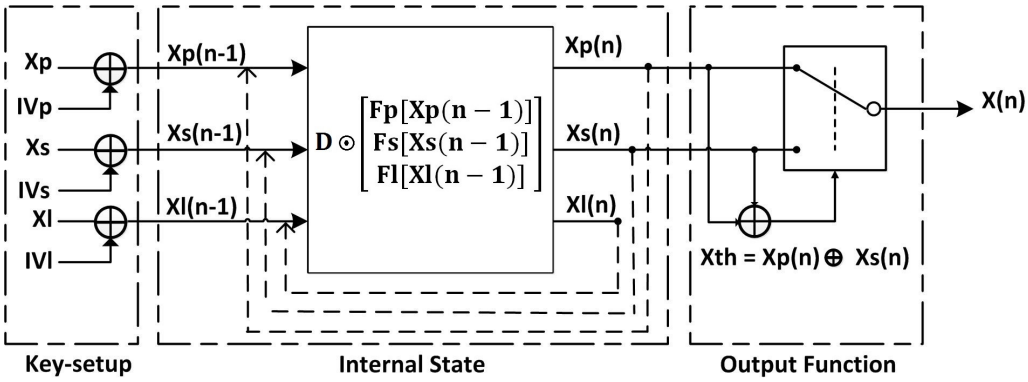


Figure 4.3 – Architecture of the proposed DM-PCNG.

The initial values $Xp(0)$, $Xs(0)$ and $Xl(0)$ are initialized throughout the key-setup function, as in Eq.(A.1) and Eq.(A.2).

The equation of the system is given by:

$$\begin{bmatrix} Xp(n) \\ Xs(n) \\ Xl(n) \end{bmatrix} = \mathbf{D} \odot \begin{bmatrix} Fp[Xp(n-1)] \\ Fs[Xs(n-1)] \\ Fl[Xl(n-1)] \end{bmatrix}. \tag{4.6}$$

where D is the binary diffusion matrix:

$$\mathbf{D} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}. \tag{4.7}$$

And $\odot$ is the operator defined as follows :

$$\begin{bmatrix} Xp(n) \\ Xs(n) \\ Xl(n) \end{bmatrix} = \begin{bmatrix} Fp[Xp(n-1)] \oplus Fs[Xs(n-1)] \\ Fs[Xs(n-1)] \oplus Fl[Xl(n-1)] \\ Fp[Xp(n-1)] \oplus Fl[Xl(n-1)] \end{bmatrix}. \tag{4.8}$$

The choice of the output samples $X(n)$ is governed, as in Eq.(A.5) by a threshold T and the chaotic sample Xth, with $Xth(n) = Xp(n) \oplus Xs(n)$.

For each new running of the system, the initial vector IV is updated using the LRNG.

The secret key is similar to one used in CM-PCNG, but does not include the coupling parameters ε_{ij} .

Architecture of the proposed CS-PCNG

The architecture of CS-PCNG is presented in Fig.4.4. Compared to the architecture of CM-PCNG, CS-PCNG's architecture is differentiated not only to internal-state function, but also to output function. The internal state uses two chaotic maps (PWLCM and SkewTent), and includes coupling and swap chaotic techniques. The output function is a XOR operation between $Xp(n)$ and $Xs(n)$ samples.

The initial values $Xp(0)$ and $Xs(0)$ are calculated as follows:

$$\begin{cases} Xp(0) = Xp \oplus IVp \\ Xs(0) = Xs \oplus IVs \end{cases} \tag{4.9}$$

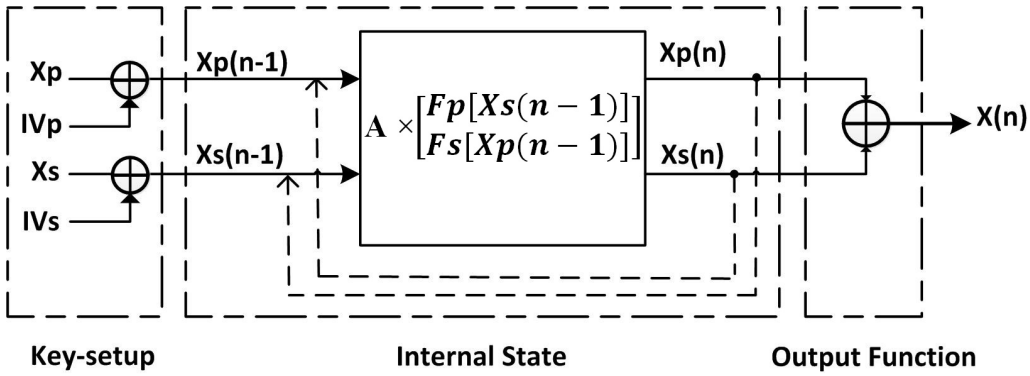


Figure 4.4 – Architecture of the proposed CS-PCNG.

Where $IVp = lsb(IV)$ and $IVs = L_{cir}[lsb(IV), 3]$.

Notice that, the secret key K of the system is formed by:

— the initial conditions Xp and Xs of the chaotic maps Pwlcm and Skewtent, ranging from 1 to 2^N-1,

— the control parameters Pp and Ps of Pwlcm and Skewtent maps, in the range $[1, 2^{N-1} - 1]$ and $[1, 2^N - 1]$ respectively,

— the parameters of the coupling technique, ε_{ij}, ranging from 1 to 2^k with k $\leq$ 5 and $i, j \in \{1, 2\}$.

The samples $Xp(n)$ and $Xs(n)$ are produced by using a coupling and swap chaotic techniques. The coupling technique is based on using the matrix **A** during the calculation of the samples $Xp(n)$ and $Xs(n)$. While the swap technique consists in using $Xp(n-1)$ as an input of the discrete function of the Skew Tent map Fs, and $Xs(n-1)$ as an input of the discrete function of the PWLCM map Fp.

The equation of the system is governed as follows:

$$\begin{bmatrix} Xp(n) \\ Xs(n) \end{bmatrix} = \mathbf{A} \times \begin{bmatrix} Fp[Xs(n-1)] \\ Fs[Xp(n-1)] \end{bmatrix}. \tag{4.10}$$

with

$$\mathbf{A} = \begin{bmatrix} (2^N - \varepsilon_{11}) & \varepsilon_{12} \\ \varepsilon_{21} & (2^N - \varepsilon_{22}) \end{bmatrix} \tag{4.11}$$

Output samples $X(n)$ are calculated throughout the produced samples $Xp(n)$ and $Xs(n)$ as follows:

$$X(n) = Xp(n) \oplus Xs(n). \tag{4.12}$$

4.2.2 Implementation of the proposed PCNGs

In order to study the performance evaluation of the proposed PCNGs, we implement the proposed architectures on an Intel Core i5 @ 2.60 GHz with 15.6 GB Running on Ubuntu 14.04 Trusty Linux, in C language and using the GCC GNU compiler. In this section, we will consider the proposed CM-PCNG as an example to well explain the C implementation. Two versions are implemented: a sequential version and a parallel one, based on the "pthread" library. In next sub-sections we will describe in details these implementations.

Sequential implementation of the proposed PCNGs

The general structure of a PCNG as presented in Figure 4.1 has as input the secret "K", the initial "IV" and the parameters of the system, and generates as an output a random sequence $X(n)$.

In our implementation, the secret key and the parameters are taken from the files "key.txt" and "parameters.txt" respectively. For the CM-PCNG, the secret key is composed of the initial conditions Xp, Xs and Xl, the control parameter Pp and Ps and the parameters of the coupling matrix **A**, ε_{ij}. The initial

conditions and the control parameters are quantified on 4 bytes (32 bits) while ε_{ij} are encoded in one byte (8 bits). The generated sequence $X(n)$ is composed of Ns samples. The "parameters.txt" file contains the number of bits N in which each sample of the generated sequences is encoded and the number of samples Ns. In this thesis, we choose N equal to 32 bits. Figure 4.5 presents the general structure of a generated sequence.

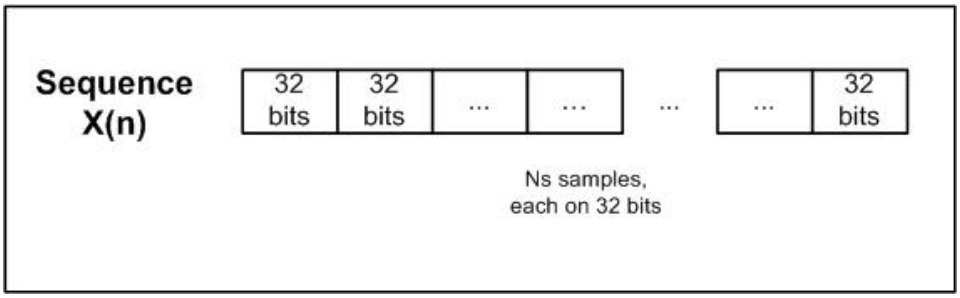


Figure 4.5 – General structure of a generated sequence $X(n)$.

As mentioned in Figure 4.1, in order to generate the required random sequence of Ns samples, once we have the secret key K and the initial vector IV which is chosen randomly from the LRNG, first we calculate the initial samples $Xs(0)$ and $Xp(0)$ using the key-setup function. Second, we generate the samples $Xs(n)$ and $Xp(n)$ and we choose the output sample $X(n)$. These operations (internal state and output function) are repeated Ns times to generate all the desired samples of the sequence. All used functions have been implemented in series. The following Alg.3, illustrate the generation of the pseudo random sequence $X(n)$.

Algorithm 3 Generation of the pseudo random sequence $X(n)$.

Key setup function
Xs = Xs $\oplus$ IVs
Xp = Xp $\oplus$ IVp
Xl = Xl $\oplus$ IVl
samples generation
$A_{11} = 2^{32} - \varepsilon_{12} - \varepsilon_{13}$
$A_{22} = 2^{32} - \varepsilon_{21} - \varepsilon_{23}$
$A_{33} = 2^{32} - \varepsilon_{31} - \varepsilon_{32}$
for int $k = 1$, $k{+}{+}$, while $k < Ns$ **do**
  internal state
  Xp = Fp(Xp)
  Xs = Fs(Xs)
  Xl = Fl(xl)
  Xp=((A_{11} $\times$ Xp) + ($\varepsilon_{12}\times$ Xs) + ($\varepsilon_{13}\times$Xl))
  Xs=(($\varepsilon_{21}\times$Xp) + ($A_{22}\times$ Xs) + ($\varepsilon_{23}\times$ Xl))
  Xl=(($\varepsilon_{31}\times$Xp) + ($\varepsilon_{32}\times$Xs) +($A_{33}\times$ Xl))
  output function
  XTh = Xs $\oplus$ Xl
  if XTh < T **then**
    X[k] = Xp
  else
    X[k] = Xs
  end if
end for

Parallel implementation of the proposed PCNGs

With the progress in computer science field and high computational power available nowadays, parallel implementation has been very ubiquitous. Most laptops, desktops and servers use a multicore processor. The main purpose of parallel implementation is to perform computations faster than can be done with a single processor. In simple terms, parallel software processing provide the possibility to divide a massive computational operation into several separate processes that execute concurrently through different processors to solve a common operation [185] [203].

Many parallel thread libraries for software applications have been implemented in the literature to provide threads - a unit of concurrent/parallel execution- which permit parallel execution according to the system [227][206][71][35]. Each library provides a specific model of parallelism such as geometric modeling or graph algorithms which use dynamic linked data structures. The choice among them will be based on the model performance, data dependencies, statistical feedback, current run-time conditions and also its portability in many operating systems.

The library we choose to implement parallelism in the generation of the random sequence is POSIX threads, or, more often, *Pthreads* [59] which is a part of the IEEE standard for Unix-like operating systems, called POSIX [35]. *Pthreads* presents an application programming interface (API) for multithreaded programming. *Pthreads* is not a programming language (like Java and C), but it is defined as a C library that can be, in principle, linked with a C program.

For our parallel implementation of the proposed PCNG, only the Internal state and output functions are ensured with parallel programming. The used computer to implement our programs is composed of four cores. So, we create four threads in our application which will participate concurrently in the generation of the pseudo random sequence. Each thread Th_i with i=1,2,3,4, is given a section of the sequence (or a specified number of samples) to be generated. Each thread works on the generation of its subsequence using a different secret key. In fact, as the system is deterministic, each thread needs to have a different secret key to not generate the same sequence. Also, chaotic systems are highly sensitive to initial conditions. For that, from the initial conditions Xs, Xp and Xl, we create four different sub-initial conditions using a circular shift function (rotation) as described in Alg.4.

Algorithm 4 Generation of initial conditions for each thread.

{left circular shift by 3 bits}
{$Xp[i]$, $Xs[i]$ and $Xl[i]$ are initial conditions for thread Th_i}.
uint32_t shift = 0
for int $i = 1$, $i{+}{+}$, while $i \leq 4$ **do**
  $shift = Xp[i-1] >> (32-3)$ //shifted left by (32-3) bits
  $Xp[i] = Xp[i-1] << 3$ //shifted right by 3 bits
  $Xp[i] = Xp[i]$ | shift
  $shift = Xs[i-1] >> (32-3)$
  $Xs[i] = Xs[i-1] << 3$
  $Xs[i] = Xs[i]$ | shift
  $shift = Xl[i-1] >> (32-3)$
  $Xl[i] = Xl[i-1] << 3$
  $Xl[i] = Xl[i]$ | shift
end for

Each thread has a fixed number of samples to generate. Suppose that our sequence is composed of Ns=10 samples. The number of samples generated by each thread is calculated as explained in Alg.5. Table 4.1 presents the number of samples generated by each thread when Ns is equal to 10.

Algorithm 5 Number of samples generated by each thread.

```
int remainder = MOD(Ns / 4)
int min = 0, max = 0
for int k = 1, k++, while k ≤ 4 do
  min = (k-1) × (Ns / 4)
  if (k != 4) then
    max = k × (Ns/4)+ remainder
  else
    max = k × (Ns/4)
  end if
  {Nbth[k]is the number of samples to be generated by the thread Th_k }
  Nbth[k] = max - min
end for
```

Table 4.1 – Number of samples generated by each thread.

Thread	min	max	Number of samples
Th_1	0	2	2
Th_2	2	4	2
Th_3	4	6	2
Th_4	6	10	4

As we already noted, all threads participate concurrently in the generation of the random sequence. Each one generates individually a fixed number of samples. Once generation of samples finishes including a waiting process, generated samples are saved in the sequence following a specific structure. The structure of the sequence with the different generated samples from threads, using a parallel programming, is described in Figure 4.6. Finally, the generated sequence is saved a result file to check the randomness of the sequence.

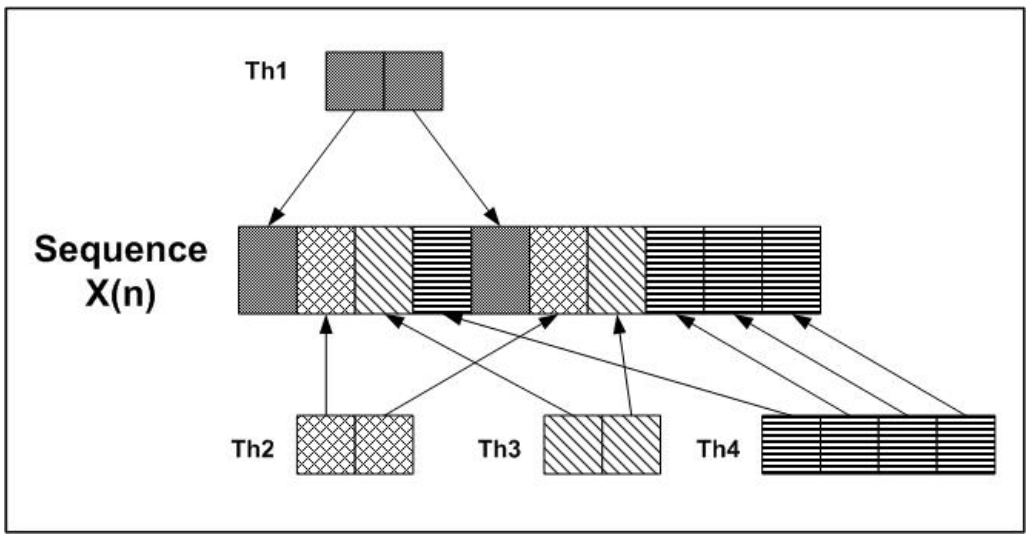


Figure 4.6 – General structure of a generated sequence using a parallel programming.

Software implementation testing

The software testing is an important and integral phase in any software development cycle, responsible for a significant portion of the costs of developing and maintaining software [39][139]. Software testing

aims to evaluate the capability of the program and determine that it meets its required results. Software testing is not limited to treating the specification of the program. But it also include analysing and testing of the software implementation and code in various ways to determine the degree of correspondence between code and specifications [84][123]. There are an abundance of code testing tools and techniques exist. These can be classified into following two categories: static analysis and dynamic testing, which are complementary approaches to code testing. In static analysis, the structure of the code is analyzed, but the code is not executed. Static analysis inspects program code for all possible run-time behaviours and seek out coding flaws, memory leaks, buffer overflows, and potentially malicious code. It can for example, detect that a particular variable is uninitialized on all possible control paths through a code or that a variable is assigned a value which is never used on any subsequent path through the program. On the other hand, dynamic testing adopts the opposite approach and investigates the runtime behaviour of the code. It involves deriving a test plan, executing test cases, and evaluating the results. Dynamic testing can for example, record the exact sequence of values assigned to a variable, although only on the particular control path traversed during the test.

We have performed different static and dynamic tools in order to verify and test our program code. Among the static analysis tools that we used, we quote:

— *Clang Static Analyzer*: It is a industrial-quality static analysis tool for analyzing C, C++, and Objective-C programs. Clang Static Analyzer is part of the Clang project. It is open-source, extensible, and has a high quality of implementation [1]. Clang Static Analyzer helps programmer to find bugs, including some issues that might not be easily detected by the programmer. The analyzer is invoked from the command line, and is intended to be run in tandem with a build of a code-base.

— *GCC compiler on Debugging mode*: GCC, the GNU Compiler Collection, is a collection of compilers created by the GNU project [5]. GCC is free software that can compile various programming languages, including C, C++, Java, etc. GCC has a debugging tool, GNU Debugger (gdb) that allows to track the bugs / errors found in any program code [6]. Also, GCC has some options (such as -wall and -Wextra) which enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.

The dynamic tests include:

— *LeakTracer*: It is a tiny and efficient memory-leak tracer for C and C++ programs [7]. LeakTracer is open-source and available in the Ubuntu software Center. It uses gdb to print out the lines of code that have memory leaks, together with leak count and size. It does not trace malloc etc., but only operator new/delete.

— *Leak-analyzer*: It is memory-leak analyzer, similar to the LeakTracer tool. It uses the gdb debugger to analysis the code and show the memory leaks.

— *Valgrind*: A free and open-source framework for building dynamic analysis tools, available under the GNU General Public License [13][182]. It is goal is to automatically detect many memory management and threading bugs, and profile programs in detail. Valgrind includes many debugging and profiling tools. An interesting example of these tools is *Memcheck* [233]. Memcheck detects a wide range of memory problems and is designed primarily to C and C++ programs. It checks all reads and writes of memory and intercept calls to malloc/new/free/delete functions.

— *Callgrind*: A part of Valgrind framework and extension to Cachegrind tool. It is a cache profiler. It performs detailed simulation of the caches (I1, D1 and L2) in the CPU and so can accurately determine the sources of cache misses in the program code. It provides informations about the number of cache misses, memory references, instructions executed for each line of source code and extra information about callgraphs [270].

— *DRD:* A Valgrind tool for detecting errors in multithreaded C and C++ programs [2]. It works for any programs that uses threading concepts built on the POSIX threading primitives.

We have conducted these static and dynamic analysis tool in order to test our C program code. Obtained results show that our implementations have not memory leaks and anomalies or defects in the code.

Program code security test is another important factor to ensure the software quality and to eliminate every security gaps. In cryptographic applications, ensuring that sensitive data (e.g., cryptographic keys) is no longer accessible, if the application no longer has pointers to it, will reduce the impact of the attack. Therefore, it is often necessary to wipe sensitive data from memory once it is no longer needed. Zeroing buffers which contained sensitive information is an exploit mitigation technique. The *memset()* function , defined below, is an approach that permits to set a range of memory to a value, and is often used to zero out a series of bytes [8].

void *memset(void *str, int c, size_t n);

Where:

— *str* is a pointer to the block of memory to fill;

— *c* is the value to be set. The value is passed as an int, but the function fills the block of memory using the unsigned char conversion of this value;

— *n* is the number of bytes to be set to the value.

Some optimizing compiler could employ "*dead store removal*"; that is, it could decide that *str* is never accessed after the call to *memset()*. Thus, the call to *memset()* could be optimized away. Consequently, the observable behaviour of the program is unchanged by the optimization. The *str* remains in memory and possibly to be discovered by some other process requesting memory. To work around this, we use the volatile function pointer *memset_ptr*, as described in Listening 4.1. The volatile type will prevent the compiler from optimizing the code; and so the compiler is forced to emit the function call which causes the key buffer to be zeroed. This approach should work on any standard-compliant platform.

```
static void * (* const volatile memset_ptr)(void *, int, size_t) = memset;
static void secure\_memzero(void * p, size_t len){
      (memset_ptr)(p, 0, len);
}

void dosomethingsensitive(void){
      uint8_t key[32];
      ...
      / * Zero sensitive information. * /
      secure_memzero(key, sizeof(key));
}
```

Listing 4.1 – Description of secure_memzero function

Another issue which needs to be taken into account in security code analysis is that confidential data in a process' address space might be saved on secondary storage and survive there beyond the expectations of the programmer. To solve this problem, the memory pages containing the sensitive data can be locked to prevent them from being paged to disk or transmitted over a network. One approach that we used in our implementations to prevent memory from being swapped out, is by using the *mlock()* system call to lock the physical pages associated with a virtual address range into memory [103].

The two functions that locks and unlocks pages are described as follows:

int mlock(const void *addr, size_t len);

int munlock(const void *addr, size_t len);

mlock() locks pages in the address range starting at addr and continuing for len bytes. All pages that contain a part of the specified address range are guaranteed to be resident in RAM when the call returns successfully; the pages are guaranteed to stay in RAM until later unlocked [9].

munlock() unlocks pages in the address range starting at addr and continuing for len bytes. After this call, all pages that contain a part of the specified memory range can be moved to external swap space again by the kernel [10].

In next sections, we demonstrate the robustness of the proposed chaotic number generators through, first a theoretical analysis in Section 4.2.3 and second several known statistical tests presented in Section 4.2.4.

4.2.3 Security Analysis of the proposed PCNGs

Key space analysis

A PCNG should have a large key space in order to make brute-force attack infeasible. It is generally accepted that a key space of size equal or greater to 2^{128} is secure. The size of the secret key of the proposed CM-PCNG, DM-PCNG and CS-PCNG are respectively given by:

$$|K1| = (|Xp| + |Xs| + |Xl|) + (|Pp| + |Ps|) + 6 \times |\varepsilon_{ij}| = 189\ bits. \tag{4.13}$$

$$|K2| = (|Xp| + |Xs| + |Xl|) + (|Pp| + |Ps|) = 159\ bits. \tag{4.14}$$

$$|K3| = (|Xp| + |Xs|) + (|Pp| + |Ps|) + 4 \times |\varepsilon_{ij}| = 147\ bits. \tag{4.15}$$

where $|Xp| = |Xs| = |Xl| = |Ps| = 32$ bits; $|Pp| = 31$ bits and $|\varepsilon_{ij}|$ is equal to 5 bits. The proposed algorithms have 2^{189}, 2^{159} and 2^{147} different combinations of the secret key. Therefore the secret key sizes of the three architectures are large enough to make brute-force attack infeasible. Such a large space of keys is a necessary condition, but not sufficient. Indeed, the generated sequences must be cryptographically secure.

Key Sensitivity analysis

The sensitivity on the key is an essential property for any PCNG. Naturally, a small change in the secret key causes a large change in the output sequences. In order to verify this characteristic, we calculate the Hamming Distance of two sequences generated with only one bit change (least significant bit of the parameter Pp). We calculate the average Hamming Distance D_H between two sequences S_1 and S_2, over 100 random secret keys. The $D_H(S_1, S_2)$ is defined by the following equation:

$$D_H(S_1, S_2) = \frac{1}{Nb} \times \sum_{n=1}^{Nb} (S_1[n] \oplus S_2[n]) \tag{4.16}$$

With Nb is the number of bits in a sequence. The obtained average value of Hamming distance for the proposed CM-PCNG, DM-PCNG and CS-PCNG are presented in Table 4.2. These values are close to the optimal value of 50%. This result illustrates the heigh sensitivity on the secret key of the proposed PCNGs.

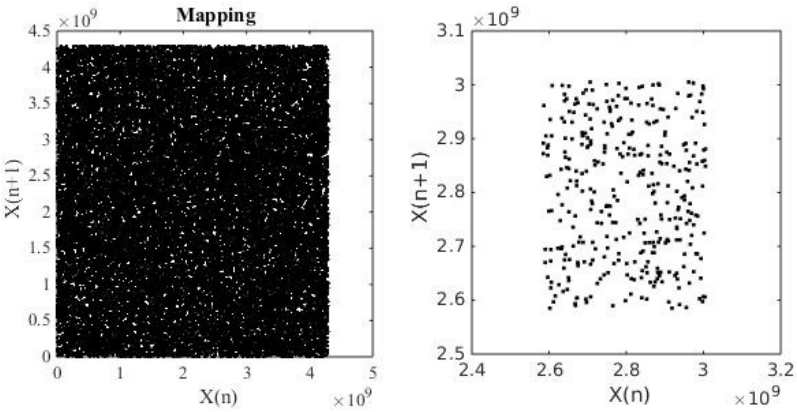
Table 4.2 – Values of D_H for the proposed CM-PCNG, DM-PCNG and CS-PCNG.

PCNG	D_H
CM-SC	0.499988
DM-SC	0.500025
CS-SC	0.500041

4.2.4 Statistical Analysis of the proposed PCNGs

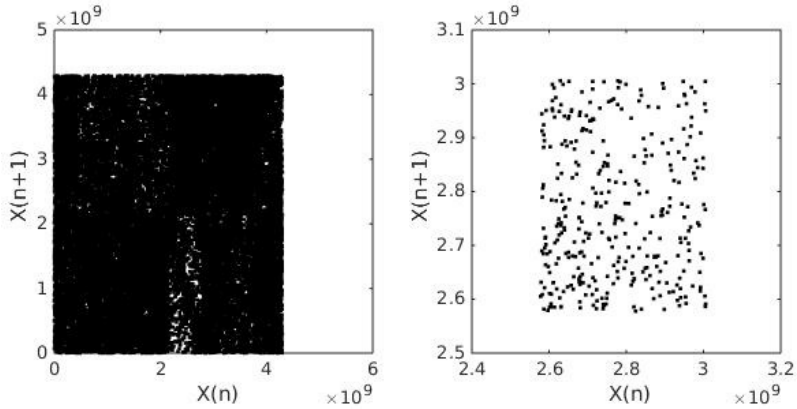
Phase space trajectory or mapping analysis

The mapping or the phase space trajectory is one of the characteristics of the generated sequence that reflects the dynamic behaviour of the system. We draw in Figures 4.7a, 4.7c and 4.7e the mapping of sequences $X1$, $X2$ and $X3$, each containing Ns = 31250 samples, generated by CM-PCNG, DM-PCNG and CS-PCNG respectively and a zoom of these mappings are given in Figures 4.7b, 4.7d and 4.7f.



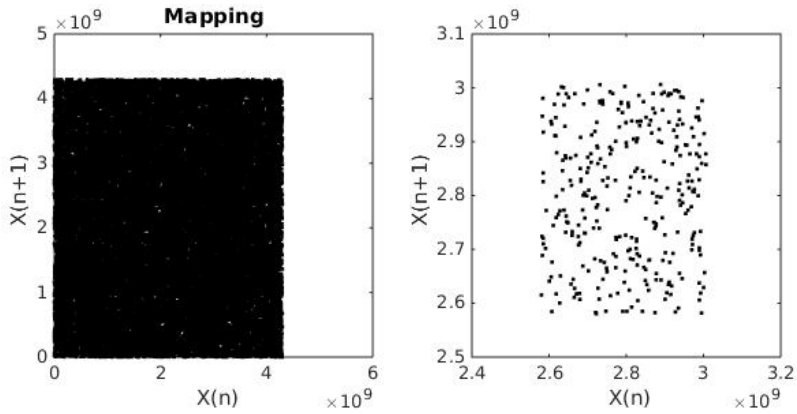
(a) Mapping of sequence $X1$

(b) Zoom on the mapping



(c) Mapping of sequence $X2$

(d) Zoom on the mapping



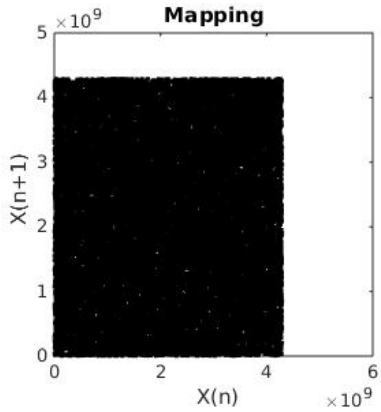
(e) Mapping of sequence $X3$

(f) Zoom on the mapping

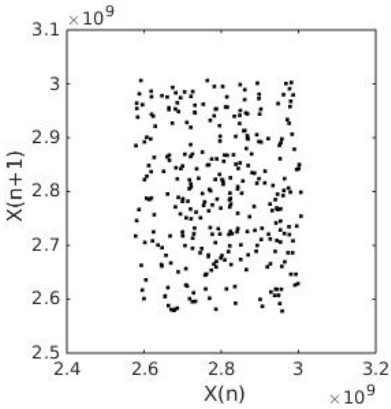
Figure 4.7 – Mapping of sequences $X1$, $X2$ and $X3$, generated by CM-PCNG, DM-PCNG and CS-PCNG respectively, and a zoom of these mappings.

The resulting mapping of $X1$, $X2$ and $X3$ seems to be random. This is due to the used techniques of coupling, swapping and chaotic multiplexing. Therefore, it is impossible from the generated sequences to

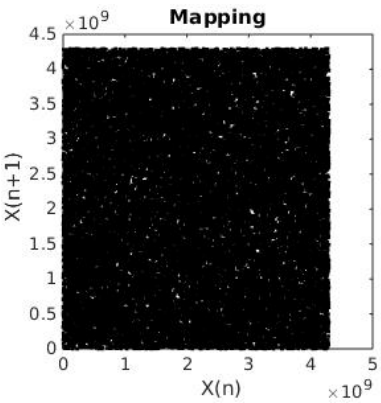
know which type of map is used. However, in the mapping of $X2$, we observe small empty areas. Then, we can say that the generated sequences of the CM-PCNG and CS-PCNG are more uniform than those generated by the DM-PCNG. This observation will be confirmed by the Chi-square test of the generated sequence. Also, we notice that the mapping of the coupled sequences Xp, Xs and Xl seems to be random. We draw in Figures 4.8a, 4.8c and 4.8e the mapping of sequences Xp generated by CM-PCNG, DM-PCNG and CS-PCNG respectively and we give a zoom of these mappings in Figures 4.8b, 4.8d and 4.8f. Similar results are obtained for mappings of the sequences Xs and Xl.



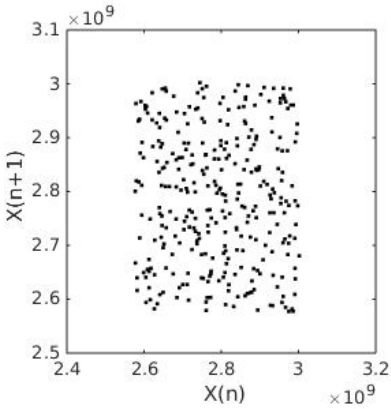
(a) Mapping of sequence Xp generated by CM-PCNG.



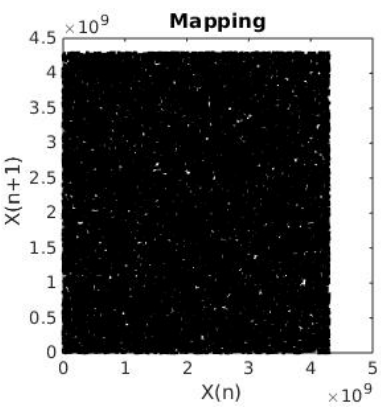
(b) Zoom on the mapping



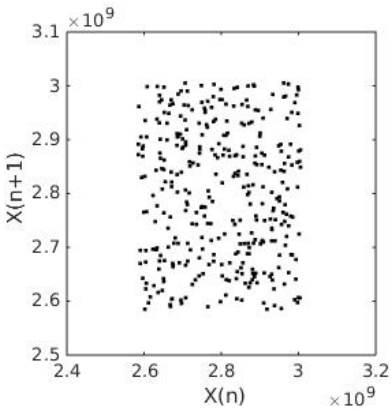
(c) Mapping of sequence Xp generated by DM-SC.



(d) Zoom on the mapping



(e) Mapping of sequence Xp generated by CS-SC.



(f) Zoom on the mapping

Figure 4.8 – Mapping of sequences Xp generated by CM-PCNG, DM-PCNG and CS-PCNG respectively.

Approximated Invariant values

To prove the uniformity of the generated sequences, Lozi [156] uses the "approximated invariant measures". This function was computed with floating numbers and based on the partition of the mapping space to M^2 small squares (boxes). In finite precision N, we defined the approximated invariant measures $Pd_N(si, tj)$ in the same manner as in [156]. First, the space mapping is divided into M^2 boxes $r_{i,j}$ as follows:

$$s_i = X_{min} + i \times l, i = 0, ..., M. \tag{4.17}$$

$$t_j = X_{min} + j \times l, j = 0, ..., M. \tag{4.18}$$

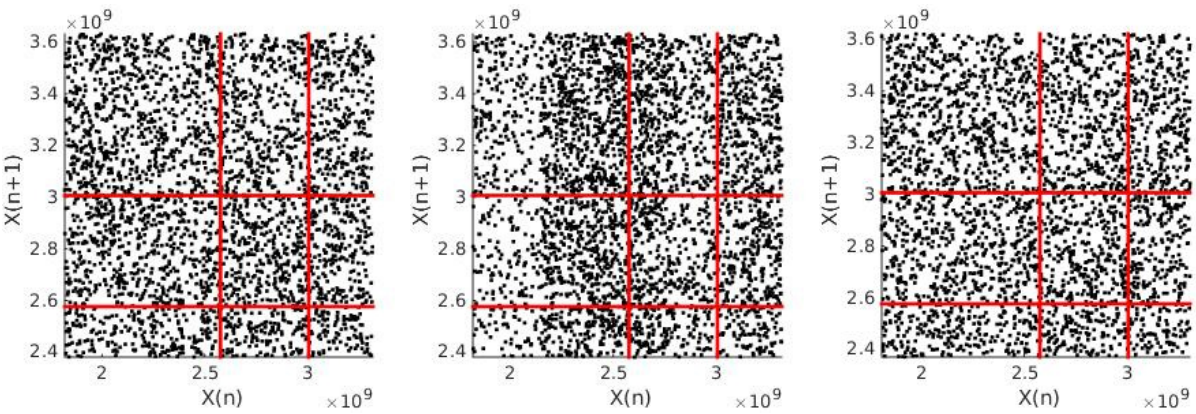
where

$$l = \frac{X_{max} - X_{min}}{M}. \tag{4.19}$$

with $X_{min} = min(Xi(Ns))$, $X_{max} = max(Xi(Ns))$ and Ns is the number of samples under test.
The box $r_{i,j}$ is given by :

$$r_{i,j} = [s_i, s_{i+1}[\times[t_j, t_{j+1}[, i, j = 0, ..., M - 1. \tag{4.20}$$

In Figures4.9a, 4.9b and 4.9c, we show the $r_{6,6}$ box, after zooming the mapping of sequences $X1$, $X2$ and $X3$.



(a) Zoom on the phase space of $X1$ (b) Zoom on the phase space of $X2$ (c) Zoom on the phase space of $X3$

Figure 4.9 – Zoom on the phase space of sequences $X1$, $X2$ and $X3$ generated by CM-PCNG, DM-PCNG and CS-PCNG respectively.

The approximated probability distribution function $Pd_N(si, tj)$ is defined as follows:

$$Pd_N(si, tj) = \frac{\#r_{i,j}}{Ns/M^2}. \tag{4.21}$$

with $\#r_{i,j}$ is the number of samples inside the box $r_{i,j}$.
Obtained values of $\#r_{i,j}$ and $Pd_N(si, tj)$ for sequences $X1$, $X2$ and $X3$ are given in Tables 4.3, 4.4 and 4.5 respectively, for all boxes with M = 10 and Ns= 31250. In Tables 4.6, 4.7 and 4.8 we give the values of $\#r_{i,j}$ and $Pd_N(si, tj)$ for Ns= 31250 $\times$ 100.
Theoretically, the number of samples inside each box $r_{i,j}$ is $Ns/M^2 \simeq 312$. Furthermore, the closer the $Pd_N(si, tj)$ value is to 1, the better the uniformity.
As we can see, compared to results in Tables 4.4 and 4.5, results of Table 4.3 are closer to uniform distribution. Indeed, for sequence $X1$, the smallest value of $Pd_N(si, tj)$ is 0.883 and the biggest $Pd_N(si, tj)$

value is 1.107. Also, for sequence $X3$, the smallest value of $Pd_N(si, tj)$ is 0.832 and we only have 4 values smaller than 0.88. Likewise, the biggest $Pd_N(si, tj)$ value is 1.142 and only we have 5 values bigger than 1.10. For sequence $X2$, we observe that: the smallest value of $Pd_N(si, tj)$ is 0.246 and there are 34 values smaller than 0.88. Additionally, the highest $Pd_N(si, tj)$ value is 1.658 and there are 40 values higher than 1.10.

Table 4.3 – Values of $\#r_{i,j}$ and $Pd_N(si, tj)$ for sequence $X1$ with Ns = 31250 samples.

$\#r_{i,j}$ $Pd_N(si, tj)$									
325 1.04	319 1.020	311 0.995	314 1.005	336 1.075	296 0.947	314 1.005	309 0.989	294 0.941	296 0.947
320 1.0240	320 1.024	325 1.040	311 0.995	301 0.963	295 0.944	308 0.986	327 1.046	301 0.963	297 0.950
333 1.066	334 1.069	341 1.0919	321 1.027	306 0.979	316 1.011	305 0.976	287 0.918	333 1.065	344 1.1
317 1.0144	326 1.043	315 1.008	344 1.1008	337 1.078	346 1.107	310 0.992	319 1.02	334 1.068	313 1.001
325 1.04	292 0.934	304 0.972	331 1.059	303 0.969	309 0.988	297 0.950	321 1.027	297 0.950	303 0.969
310 0.992	300 0.96	324 1.036	338 1.081	297 0.950	301 0.963	301 0.963	304 0.972	295 0.944	319 1.020
309 0.988	333 1.065	317 1.014	307 0.982	293 0.937	310 0.992	318 1.017	327 1.046	293 0.937	285 0.912
298 0.953	276 0.883	304 0.972	328 1.049	319 1.02	314 1.004	310 0.992	329 1.052	342 1.094	286 0.915
311 0.995	308 0.985	341 1.091	338 1.081	284 0.908	298 0.953	330 1.056	293 0.937	331 1.059	296 0.947
266 0.851	299 0.956	338 1.081	329 1.052	306 0.979	304 0.972	299 0.956	289 0.924	309 0.988	309 0.988

Table 4.4 – Values of $\#r_{i,j}$ and $Pd_N(si, tj)$ for sequence $X2$ with Ns = 31250 samples.

$\#r_{i,j}$ $Pd_N(si, tj)$									
291 0.931	307 0.982	284 0.909	293 0.938	287 0.918	313 1.002	332 1.062	349 1.117	328 1.05	353 1.13
474 1.517	399 1.277	415 1.328	352 1.126	422 1.35	205 0.656	211 0.675	214 0.685	184 0.589	196 0.627
381 1.219	344 1.101	359 1.149	331 1.059	361 1.155	292 0.934	233 0.746	224 0.717	250 0.8	328 1.05
328 1.050	332 1.062	339 1.085	452 1.446	377 1.206	277 0.886	292 0.934	288 0.922	236 0.755	210 0.672
518 1.658	420 1.344	421 1.347	425 1.36 0	394 1.261	169 0.541	199 0.637	190 0.608	214 0.685	247 0.790
77 0.246	141 0.451	184 0.589	211 0.675	208 0.666	442 1.414	428 1.37	477 1.526	456 1.459	438 1.402
315 1.008	226 0.723	210 0.672	214 0.685	257 0.822	350 1.12	344 1.101	351 1.123	368 1.178	405 1.296
197 0.63	298 0.954	283 0.906	285 0.912	276 0.883	342 1.094	371 1.187	387 1.238	378 1.21	328 1.05
216 0.691	253 0.81	230 0.736	200 0.64	232 0.742	389 1.245	365 1.168	402 1.286	437 1.398	411 1.315
340 1.088	352 1.126	378 1.21	368 1.178	383 1.226	283 0.906	265 0.848	263 0.842	284 0.909	311 0.995

Besides, compared to results in Table 4.3, the obtained values of $Pd_N(si, tj)$ in Table 4.6, are closer to 1. Indeed, the uniformity is better when the number of samples Ns is larger. The same remark is observed for results of CS-PCNG (see Tables 4.5 and 4.8). However, these results are not valid for DM-PCNG when

Table 4.5 – Values of $\#r_{i,j}$ and $Pd_N(si, tj)$ for sequence $X3$ with Ns = 31250 samples.

$\#r_{i,j}$ $Pd_N(si,tj)$									
312 0.998	338 1.082	291 0.931	322 1.03	330 1.056	289 0.925	348 1.114	296 0.947	323 1.034	301 0.963
343 1.098	343 1.098	311 0.995	284 0.909	306 0.979	314 1.005	305 0.976	337 1.078	316 1.011	338 1.082
295 0.944	313 1.002	293 0.938	297 0.95	295 0.944	312 0.998	312 0.998	316 1.011	321 1.027	286 0.915
313 1.002	314 1.005	318 1.018	315 1.008	317 1.014	307 0.982	301 0.963	314 1.005	287 0.918	324 1.037
319 1.021	292 0.934	314 1.005	306 0.979	337 1.078	285 0.912	315 1.008	357 1.142	319 1.021	333 1.066
299 0.957	304 0.973	301 0.963	343 1.098	326 1.043	313 1.002	314 1.005	278 0.89	320 1.024	298 0.954
339 1.085	303 0.97	313 1.002	289 0.925	304 0.973	310 0.992	349 1.117	294 0.941	318 1.018	321 1.027
290 0.928	319 1.021	313 1.002	307 0.982	312 0.998	328 1.05	320 1.024	290 0.928	324 1.037	310 0.992
319 1.021	337 1.078	273 0.874	298 0.954	319 1.021	346 1.107	298 0.954	320 1.024	260 0.832	315 1.008
321 1.027	333 1.066	313 1.002	348 1.114	331 1.059	292 0.934	278 0.89	312 0.998	297 0.95	314 1.005

Table 4.6 – Values of $\#r_{i,j}$ and $Pd_N(si, tj)$ for sequence $X1$ with $Ns = 31250 \times 100$ samples.

$\#r_{i,j}$ $Pd_N(si,tj)$									
31300 1.002	31602 1.011	31335 1.003	31421 1.005	31404 1.005	31468 1.007	31206 0.999	31201 0.998	31286 1.001	31462 1.007
31568 1.01	31215 0.999	30925 0.99	31044 0.993	31711 1.015	31264 1	31150 0.997	31098 0.995	31072 0.994	31293 1.001
31425 1.006	31076 0.994	31270 1.001	31229 0.999	31304 1.002	31279 1.001	31287 1.001	31068 0.994	31073 0.994	31020 0.993
31375 1.004	30950 0.99	31126 0.996	30988 0.992	31286 1.001	31380 1.004	31203 0.998	30913 0.989	31221 0.999	31396 1.005
31475 1.007	30911 0.989	31154 0.997	31304 1.002	31411 1.005	31362 1.004	31286 1.001	31506 1.008	31358 1.003	31427 1.006
31395 1.005	31360 1.004	31380 1.004	31601 1.011	31251 1	31458 1.007	31173 0.998	31380 1.004	31096 0.995	31440 1.006
31269 1.001	31478 1.007	31470 1.007	31108 0.995	31358 1.003	31499 1.008	31384 1.004	31250 1	31060 0.994	30705 0.983
31233 0.999	31368 1.004	31183 0.998	31053 0.994	31198 0.998	31122 0.996	31271 1.001	31180 0.998	30889 0.988	31361 1.004
31218 0.999	31022 0.993	31083 0.995	31013 0.992	31154 0.997	31265 1	31233 0.999	30911 0.989	31027 0.993	31351 1.003
31427 1.006	31357 1.003	31105 0.995	31077 0.994	31117 0.996	31438 1.006	31388 1.004	31351 1.003	31195 0.998	31206 0.999

comparing Tables 4.4 and 4.7. This is due to the fact that the samples are distributed on a periodic orbit with a small period length.

We give also the cumulative relative error calculated by:

$$CRE = \sum_{i,j=1}^{M} |\frac{Ns/M^2 - \#r_{i,j}}{Ns/M^2}|. \tag{4.22}$$

In table 4.9, we report the obtained values of CRE for sequences $X1$, $X2$ and $X3$. For this experiment, we took three different values for Ns: Ns = 31250, $Ns = 31250 \times 10$, and $Ns = 31250 \times 100$. And for each Ns, we consider two values of M: M = 5 and M = 10.

Table 4.7 – Values of $\#r_{i,j}$ and $Pd_N(si, tj)$ for sequence $X2$ with $Ns = 31250 \times 100$ samples.

$\#r_{i,j}$ $Pd_N(si, tj)$									
29390 0.94	28231 0.903	27973 0.895	28024 0.897	26081 0.835	34062 1.089	33679 1.077	33191 1.062	33276 1.065	34079 1.09
45311 1.45	41954 1.342	38137 1.22	38005 1.216	38769 1.24	22302 0.714	23238 0.743	22161 0.709	20312 0.65	20596 0.659
38874 1.244	32922 1.053	36507 1.168	37833 1.211	35911 1.149	27787 0.889	26633 0.852	23892 0.764	25664 0.821	29221 0.935
3233 2 1.035	33458 1.07	38293 1.225	43204 1.382	40442 1.294	29067 0.93	28551 0.914	29473 0.943	23292 0.745	20085 0.643
4950 3 1.584	43437 1.39	41164 1.317	40803 1.306	39851 1.275	15716 0.503	18061 0.578	20104 0.643	2286 5 0.732	23238 0.743
9008 0.288	14252 0.456	18517 0.592	20938 0.67	20945 0.67	4643 7 1.486	46812 1.498	47074 1.506	45721 1.463	43910 1.405
2771 1 0.887	26873 0.86	2516 2 0.805	22977 0.735	24330 0.779	35524 1.137	33475 1.071	35438 1.134	37720 1.207	40422 1.293
2261 2 0.724	28674 0.917	29421 0.941	27095 0.867	27703 0.886	35328 1.13	35300 1.13	37512 1.2	35492 1.136	34186 1.094
20412 0.653	25354 0.811	24276 0.777	21672 0.693	23290 0.745	40373 1.292	37131 1.188	38233 1.223	39201 1.254	39745 1.272
32781 1.049	35632 1.14	35804 1.146	37640 1.204	37403 1.197	27062 0.866	26757 0.856	26242 0.84	2614 5 0.836	26693 0.854

Table 4.8 – Values of $\#r_{i,j}$ and $Pd_N(si, tj)$ for sequence $X3$ with $Ns = 31250 \times 100$ samples.

$\#r_{i,j}$ $Pd_N(si, tj)$									
31146 0.996	31521 1.008	31105 0.995	31266 1.0005	31271 1.0006	31197 0.998	31087 0.994	31295 1.001	31126 0.996	31314 1.002
30845 0.987	31019 0.992	31233 0.999	31041 0.993	31203 0.998	31163 0.997	31504 1.008	31388 1.004	31448 1.006	31321 1.002
31486 1.007	31101 0.995	31323 1.002	31267 1	31331 1.002	31405 1.004	31134 0.996	31124 0.995	31362 1.003	30858 0.987
31185 0.997	31555 1.009	31419 1.005	31249 0.999	31282 1.001	31210 0.998	31054 0.993	31094 0.995	31282 1.001	31582 1.010
31199 0.998	31192 0.998	31177 0.997	31231 0.999	31041 0.993	31093 0.994	31487 1.007	31350 1.003	31322 1.002	31209 0.998
31552 1.009	30868 0.987	31065 0.994	31629 1.012	31477 1.007	31255 1.0001	31483 1.007	31083 0.994	31176 0.997	31222 0.999
31351 1.003	31091 0.994	31462 1.006	31265 1.0004	31145 0.996	31364 1.003	31353 1.003	31412 1.005	31194 0.998	31236 0.999
31151 0.996	31154 0.996	31302 1.001	31529 1.008	31112 0.995	31528 1.008	30971 0.991	31345 1.003	31340 1.002	30790 0.985
31177 0.997	31043 0.993	31094 0.995	31358 1.003	31277 1.003	31226 0.999	31745 1.015	31194 0.998	31236 0.999	31390 1.004
31236 0.999	31621 1.011	31211 0.998	31076 0.994	31162 0.997	31370 1.003	31055 0.993	30937 0.989	31254 1.000	31335 1.002

We observe that, whatever the values of Ns and M, the Cumulative Relative Error CRE of sequences generated by CM-PCNG is smaller than the CRE of sequences generated by DM-PCNG. Also, we notice that, for each M, the CRE of CM-PCNG decreases with a factor approximately equal to $\sqrt{Ns}$, when Ns increases. However, sequences generated by DM-PCNG do not follow the previous rule.

Histogram and Chi-square analysis

We study the distribution uniformity of the generated sequences. A PCNG must provide a uniform distribution in the whole phase space. We give in Figure 4.10a, Figure 4.10b and Figure 4.10c the histograms of generated sequences $X1$, $X2$ and $X3$, each formed by 10^7 samples, generated by CM-PCNG, DM-PCNG

Table 4.9 – Values of the Cumulative Relative Error.

		Ns		
M	PCNG	31250	31250 × 10	31250 × 100
5	CM-PCNG	0.5432	0.1982	0.0651
	DM-PCNG	2.5512	2.3857	2.3657
	CS-PCNG	0.7959	0.1812	0.0480
10	CM-PCNG	4.4869	1.5268	0.4722
	DM-PCNG	23.0597	22.8217	22.6401
	CS-PCNG	4.4255	1.6144	0.4498

and CS-PCNG respectively.

Visually, we observe that the generated sequences $X1$, $X2$ and $X3$ are nearly uniformly distributed. We then apply the Chi-square test to assert the uniformity of these sequences. The experimental Chi-square χ^2 value is given by:

$$\chi^2_{\text{exp}} = \sum_{i=0}^{K-1} \frac{(O_i - E_i)^2}{E_i}. \tag{4.23}$$

where K is the number of classes (sub-intervals) chosen in our experiment equal to 1000, O_i is the number of observed (calculated) samples in the i-th class and E_i is the expected number of samples of a uniform distribution, $E_i = 10^7/K$.

We compare the experimental value given by Eq.4.23 with a theoretical value obtained for a threshold α=0.05 and a degree of freedom K-1=999. Smaller is the experimental value of Chi-square test compared to the theoretical one, better is the uniformity of the generated sequence. Experimental and theoretical values of the Chi-Square test for sequences $X1$, $X2$ and $X3$ are presented in Table 4.10. These results confirm the uniformity of the generated sequences. We note that sequence $X1$ has a better uniform distribution than other sequences and sequence $X3$ is more uniform than sequence $X2$.

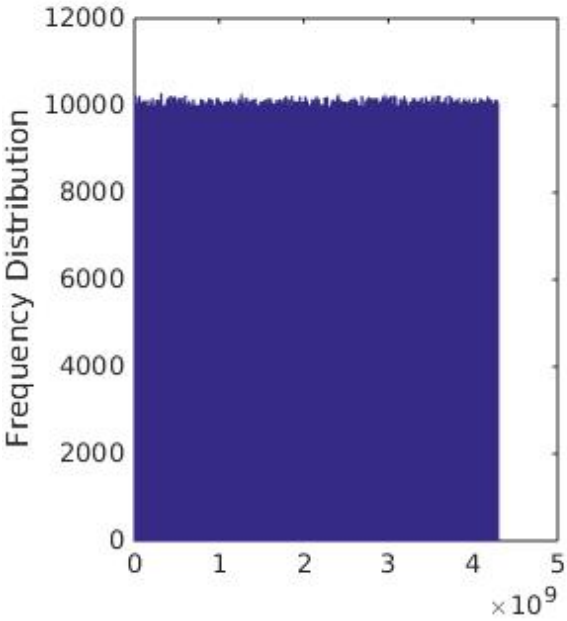
Table 4.10 – Theoretical and experimental values of the Chi-Square test for the proposed PCNGs.

Chi-square test value	**CM-PCNG**	**DM-PCNG**	**CS-PCNG**
χ^2_{th}	1073.642	1073.642	1073.642
χ^2_{exp}	904.652	960.689	918.988

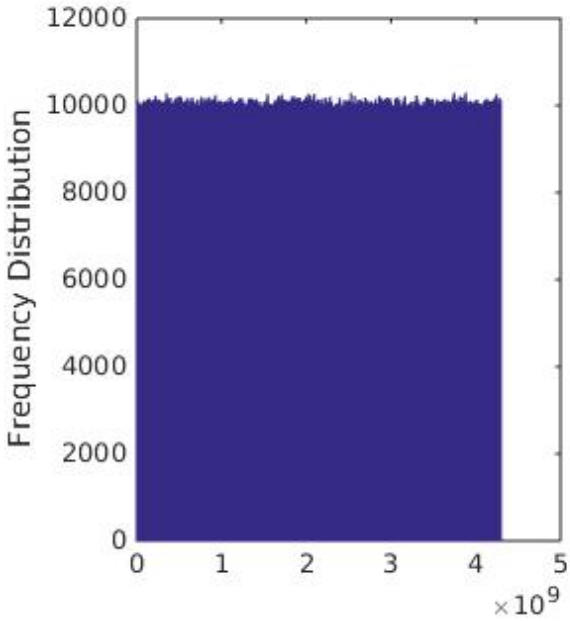
Correlation analysis

To evaluate the security of the proposed PCNGs regarding to the correlation analysis, we calculate the correlation coefficient between two sequences X and Y which are produced with nearby initial conditions and also the correlation coefficient between a generated sequence X and the coupled sequences Xp, Xs and Xl for the proposed PCNGs. The obtained results are given in Table 4.11.

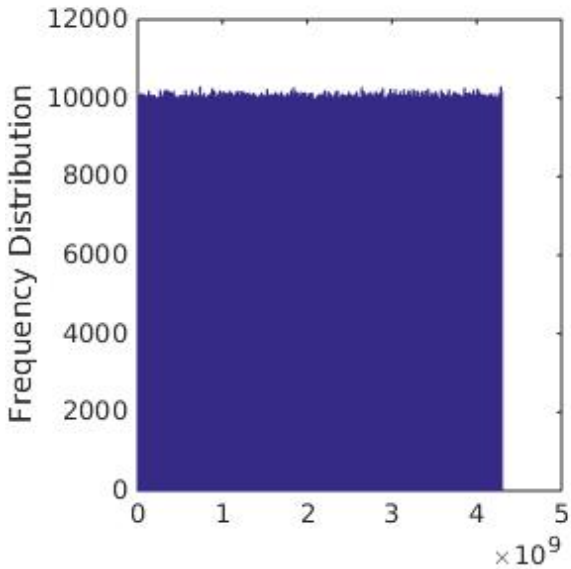
We give in Figure 4.11, the auto-correlation function of sequence X, a zoom of the autocorrelation on 200 samples of sequence X and of the cross-correlation of the sequences X and Y generated by CM-PCNG, and a zoom of the cross-correlation of sequences X and Y and of the auto-correlation of sequence X. We note the cross-correlation function of sequences X and Y given by Figure 4.11c is very low (maximum value = 0.025) compared to the auto-correlation function of sequence X. Correlation coefficients given in Table 4.11 are close to zero. Consequently, there is no correlation between the generated sequences, that are produced using slightly different keys. Similar observations are concluded for both proposed DM-PCNG and CS-PCNG. (See Figures 4.12 and 4.13)



(a) The histogram of generated sequence $X1$.



(b) The histogram of generated sequence $X2$.



(c) The histogram of generated sequence $X3$.

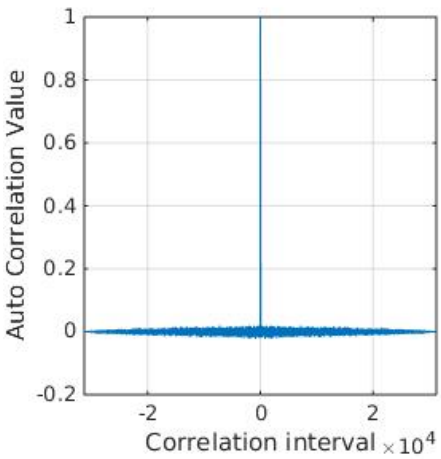
Figure 4.10 – The histograms of sequences $X1$, $X2$ and $X3$ generated by CM-PCNG, DM-PCNG and CS-PCNG respectively.

Table 4.11 – Correlation coefficients of the proposed PCNGs.

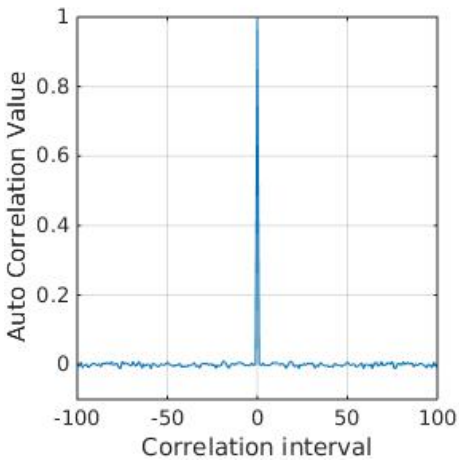
Correlation coefficient	**CM-PCNG**	**DM-PCNG**	**CS-PCNG**
$\rho_{X,Y}$	0.0025	0.0104	0.0080
ρ_{X,X_p}	0.0015	0.0078	-0.0034
ρ_{X,X_s}	-0.0047	0.0150	0.0063
ρ_{X,X_l}	0.0020	-0.0010	/

NIST test analysis

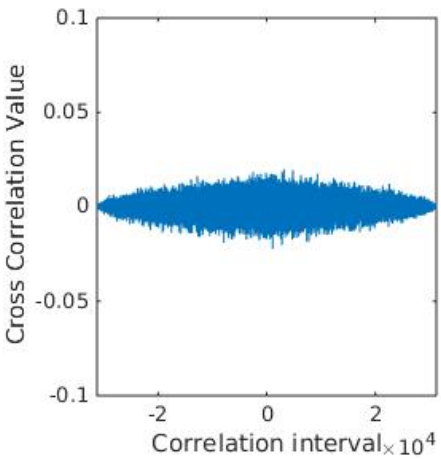
We also use one of the most popular standard test for investigating the randomness of binary data, namely the NIST statistical test [81, 224]. It focus on variety of different types of non-randomness that



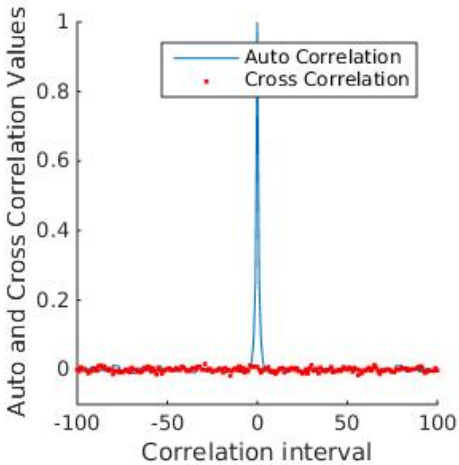
(a) Autocorrelation of sequence X



(b) A zoom of the autocorrelation on 200 samples of sequence X



(c) Cross-correlation of sequences X and Y



(d) A zoom of the cross-correlation of sequences X and Y and of the auto-correlation of sequence X

Figure 4.11 – Auto and cross-correlation functions of sequences X and Y generated by CM-PCNG.

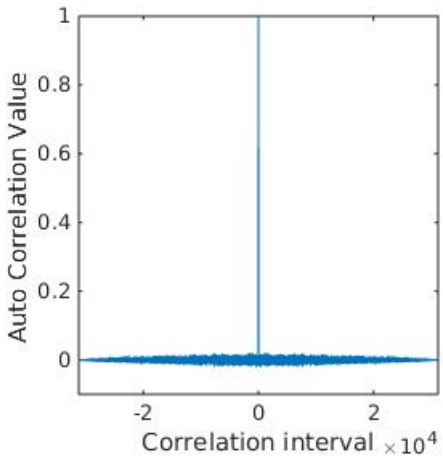
could exist in a binary sequence. For each test, a $P - value$ is calculated. The associated test is a success, if $P - value \geq \alpha$ (α is a fixed value set for all tests equal to 0.01).

Figure 4.14 and Table 4.12 give the results of NIST test obtained for sequences $X1$, $X2$ and $X3$ generated by the proposed CM-PCNG, DM-PCNG and CS-PCNG respectively.

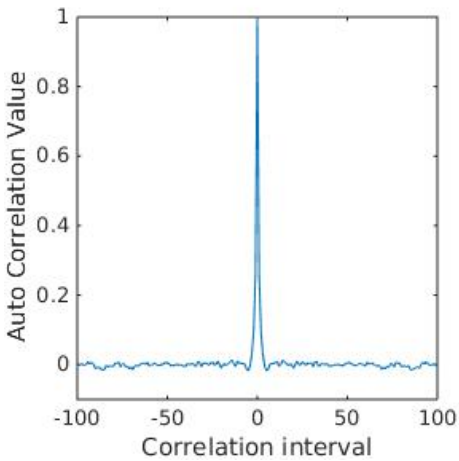
We observe that, sequences $X1$, $X2$ and $X3$ have successfully passed all the NIST tests. Therefore, the proposed chaotic generators are robust against statistical attacks. In addition, we observe that globally, the sequence $X1$ generated by the CM-PCNG algorithm pass NIST tests more efficiently than sequences $X2$ and $X3$ generated by DM-PCNG and CS-PCNG. This is in accordance with results obtained by the others statistical tests.

4.2.5 Speed performance of the proposed PCNGs

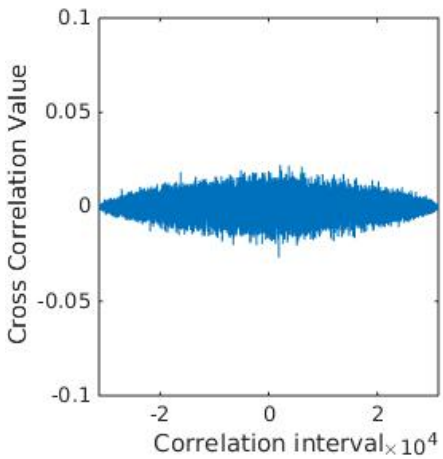
Speed performance of a PCNG is an important factor for practical applications, such in encryption algorithms for example. We study the computing performance of the proposed PCNGs. In Tables 4.13, 4.14 and 4.15 we give, the average generation time in microsecond (μs), the average bit rate in Megabits/second (Mbits/s) and the average required number of cycles to generate one byte for different lengths of sequences,



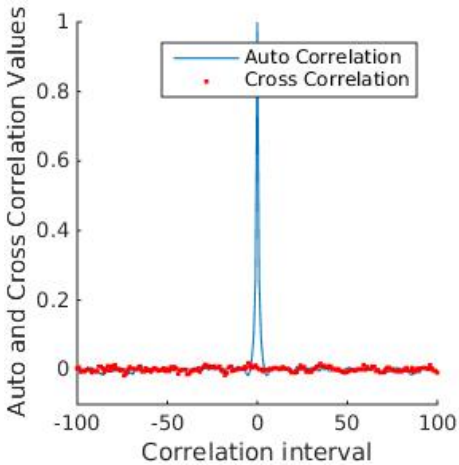
(a) Autocorrelation of sequence X



(b) A zoom of the autocorrelation on 200 samples of sequence X



(c) Cross-correlation of sequences X and Y



(d) A zoom of the cross-correlation of sequences X and Y and of the auto-correlation of sequence X

Figure 4.12 – Auto and cross-correlation functions of sequences X and Y generated by DM-PCNG.

using sequential and parallel programming for the three proposed chaotic generators CM-PCNG, DM-PCNG and CS-PCNG respectively. The average is calculated over 100 different sequences using a different secret key for each one.

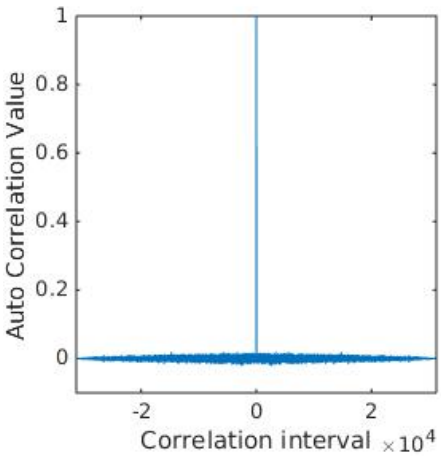
The bit rate and the number of cycles needed to generate one byte $NCpB$ is defined as follows:

$$Bit\ rate(Mbits/s) = \frac{Generated\ data\ size(Mbits)}{Average\ generation\ time(s)} \tag{4.24}$$

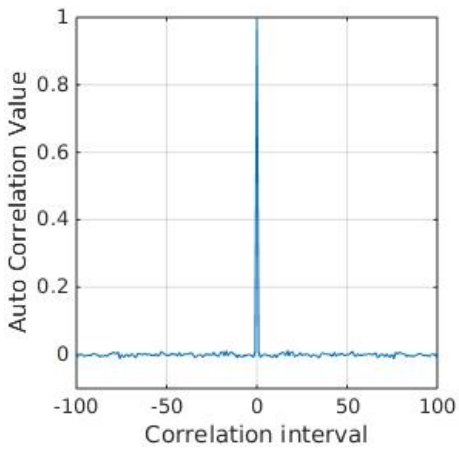
$$NCpB = \frac{CPU speed(Hz)}{Bit\ rate(Byte/s)} \tag{4.25}$$

From results of Tables 4.13, 4.14 and 4.15, we remark first that, due to its less complex internal state, the speed performance of CS-PCNG is better than one of CM-PCNG and DM-PCNG. Also, DM-PCNG is faster than CM-PCNG. Second, we observe that, for small size data (up to 32768 bytes) the PCNG implemented with sequential programming is faster than that programmed in parallel (see also Figures 4.15, 4.16 and 4.17). This is due to the time synchronization between the four threads.

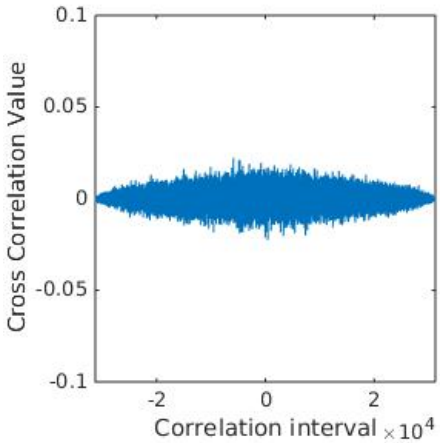
Notice that, in addition of stream ciphers, the proposed PCNGs can be used in several applications that require the generation of a large amount of secure random numbers.



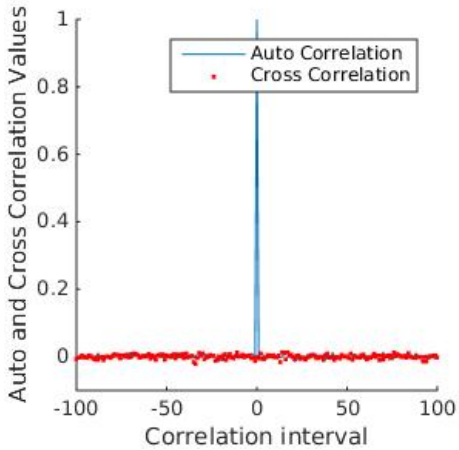
(a) Autocorrelation of sequence X



(b) A zoom of the autocorrelation on 200 samples of sequence X



(c) Cross-correlation of sequences X and Y



(d) A zoom of the cross-correlation of sequences X and Y and of the auto-correlation of sequence X

Figure 4.13 – Auto and cross-correlation functions of sequences X and Y generated by CS-PCNG.

In Table 4.16, we give the performance in terms of $NCpB$ of some known pseudo random number generators: Wang et al., [267], Akhshani et al., [19] and our proposed PCNGs. The comparison is performed for a data size equal to 786432 bytes. It can be observed that the $NCpB$ performance of the proposed PCNGs is better than the others cited.

In the following section, we will study the security analysis and the speed performance of the three proposed stream ciphers CM-SC, DM-SC and CS-SC which use as a pseudo number generator CM-PCNG, DM-PCNG and CS-PCNG respectively.

4.3 Proposed chaos-based stream ciphers

A stream cipher, as shown in Figure 4.18, is a symmetric encryption algorithm. It takes a stream of plaintext P_i, a secret key K and an initial vector IV as input and then operates the P_i with a keystream which is produced by a PCNG using the secret key K and IV to obtain a ciphered text C_i. The keystream must be different for each encryption round. As the PCNG is deterministic, the same keystream can be generated in the decryption. Then, one can recover the original plaintext P_i, by XORing the same keystream with the cipher text C_i.

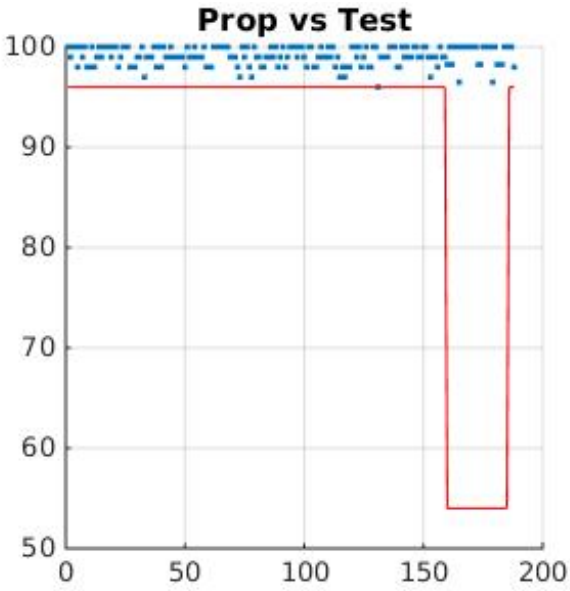
Table 4.12 – P-values and Proportion results of NIST for the proposed stream ciphers.

	CM-PCNG		**DM-PCNG**		**CS-SC**	
Test	**P-value**	**Proportion**	**P-value**	**Proportion**	**P-value**	**Proportion**
Frequency test	0.946	100	0.740	100	0.249	98
Block-frequency test	0.883	99	0.091	100	0.063	99
Cumulative-sums test	0.376	100	0.646	100	0.862	98
Runs test:	0.616	98	0.658	100	0.456	100
Longest-run test	0.898	100	0.596	99	0.720	100
Rank test	0.290	99	0.534	98	0.924	100
FFT test	0.534	100	0.554	100	0.596	98
Non-periodic-templates	0.483	99.061	0.494	99.088	0.540	98.784
Overlapping-templates	0.063	100	0.798	100	0.817	98
Universal	0.172	99	0.040	99	0.720	98
Approximty entropie	0.419	99	0.097	98	0.972	98
Random-excursions:	0.335	99.123	0.545	97.656	0.325	98.674
Random-excursions-variant	0.436	99.318	0.576	99.566	0.273	98.401
Serial test	0.478	100	0.627	99.5	0.720	99.5
Linear-complexity	0.249	98	0.262	98	0.475	100

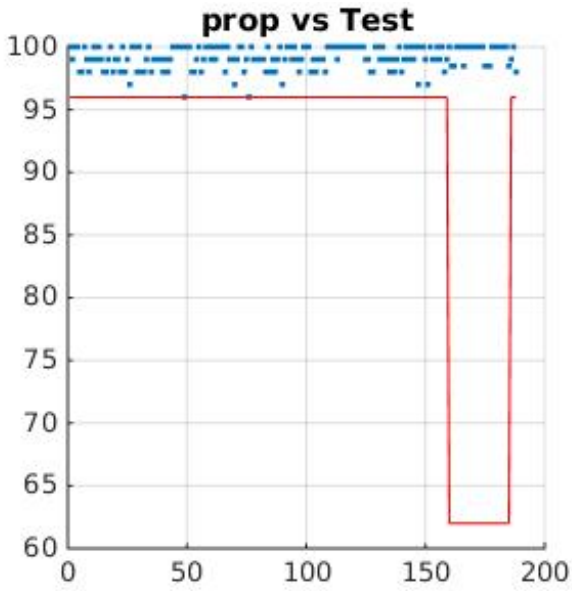
Table 4.13 – Speed Performance of CM-PCNG using sequential and parallel implementations.

Data Size (Byte)	Generation Time (μs)		Bit Rate (Mbits/s)		NCpB	
	Sequential Impl.	Parallel Impl.	Sequential Impl.	Parallel Impl.	Sequential Impl.	Parallel Impl.
64	4.02	79.56	127.36	6.43	163.31	3082.39
128	5.69	98.13	179.96	10.43	115.58	1900.93
256	8.9	95.29	230.11	21.49	90.39	922.96
512	14.89	96.90	275.08	42.27	75.61	492.07
1024	27.36	93.51	299.41	87.60	69.47	237.43
2048	49.63	89.95	330.12	182.14	63.01	108.9
4096	89.33	99.75	366.81	328.50	56.7	60.38
8192	144.19	141,03	454.51	464.67	45.76	44.76
16384	262.31	271,64	499.68	482.52	41.63	43.11
32768	521.49	359.05	502.68	724.48	41.38	27.38
65536	782.22	616.22	670.25	850.81	31.03	23.31
125000	1269.86	1140.89	787.48	876.50	25.19	22.63
196608	1970.40	1548.23	798.24	1015.91	24.85	19.53
393216	3930.71	2838.58	800.29	1108.2	24.79	17.9
786432	7826.19	4279.44	803.89	1470.15	24.68	13.49
3145728	31229.65	16936.79	805.83	1485.86	24.63	13.35

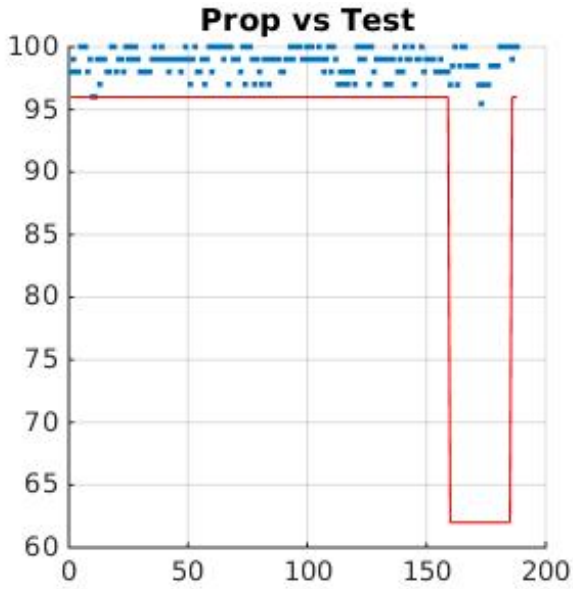
The keystream must be random enough to ensure that if an attacker has access to the keystream, he cannot recover the secret key or derive the internal state. Thus, the security of any stream cipher depends on the randomness of the keystream, therefore on the robustness of the used PCNG which is the main element of a stream cipher. Note that the same secret key and IV must be shared by the emitter and the receiver in order to encrypt/decrypt the message sent through the communication channel and must be protected from access by others.



(a) NIST tests results of the proposed CM-SC.



(b) NIST tests results of the proposed DM-SC.



(c) NIST tests results of the proposed CS-SC.

Figure 4.14 – NIST test results of the proposed PCNGs.

Several techniques have been proposed for the distribution of keys and IV. Concerning our algorithms, a symmetric key distribution is used in the generation and management of the secret keys and IV, in order to provide confidentiality and integrity of the keys. This technique is based on the use of a master key, which is infrequently used and is long lasting, and session keys which are generated and distributed for each communication between emitter and receiver [248].

A good stream cipher algorithm should be robust against cryptanalytic, statistical and brute-force attacks. Also, it should provide a high encryption speed. In this section, we discuss the security analysis of the proposed stream cipher algorithms namely CM-SC, DM-SC and CS-SC, based on the proposed CM-PCNG, DM-PCNG and CS-PCNG respectively, described in Section 4.2 and their speed performance. Key space, Key sensitivity and Statistical analysis are carried out in order to prove that the proposed stream ciphers are secure against the most common attacks.

As most encryption algorithms (AES-CTR, Rabbit, HC-128...) encrypt 128 bits by 128 bits, our stream

Table 4.14 – Speed Performance of DM-PCNG using sequential and parallel implementations.

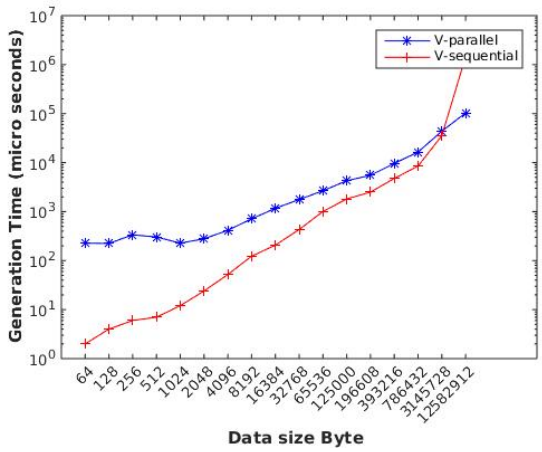
Data Size (Byte)	Generation Time (μs)		Bit Rate (Mbits/s)		NCpB	
	Sequential Impl.	Parallel Impl.	Sequential Impl.	Parallel Impl.	Sequential Impl.	Parallel Impl.
64	1.92	52.79	266.66	9.69	74.39	2144.59
128	2.58	65.16	369.89	15.71	49.98	1323.56
256	3.59	98.96	570.47	35.59	34.77	584.39
512	6.16	79.45	664.93	51.55	29.83	384.77
1024	12.03	76.29	680.96	107.37	29.13	184.73
2048	21.55	84.26	760.27	194.44	26.09	106.97
4096	42.52	77.00	770.51	425.55	25.74	48.88
8192	84.72	127.67	773.52	513.32	25.64	38.64
16384	169.33	222.98	774.03	587.81	25.63	33.75
32768	337.30	293.37	777.17	893.56	25.52	23.28
65536	671.59	491.9	780.65	1065.84	25.41	19.52
125000	1194.58	718.93	837.11	1458.52	24.85	14.26
196608	1867.03	1293.01	842.44	1621.91	24.69	12.82
393216	3465.25	2235.05	907.79	1789.66	22.91	11.62
786432	6413.96	3417.52	980.90	1840.94	21.2	11.3
3145728	25664.74	14827.4	980.56	1697.25	21.08	12.26

Table 4.15 – Speed Performance of CS-PCNG using sequential and parallel implementation.

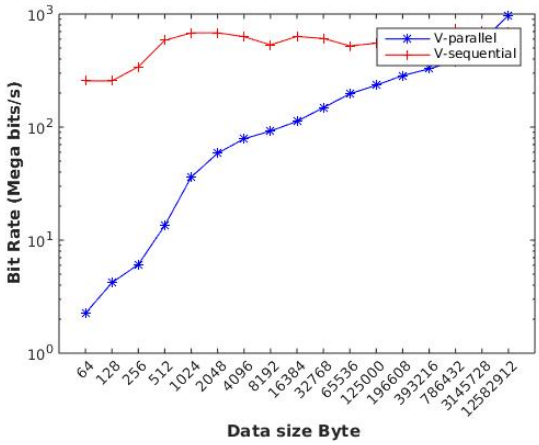
Data Size (Byte)	Generation Time (μs)		Bit Rate (Mbits/s)		NCpB	
	Sequential Impl.	Parallel Impl.	Sequential Impl.	Parallel Impl.	Sequential Impl.	Parallel Impl.
64	1.19	17.47	429.83	29.30	46.15	676.93
128	1.92	18.07	531.81	56.67	37.30	350.4
256	2.70	16.57	756.84	123.55	26.21	160.56
512	4.65	19.43	880.46	210.76	22.5 3	94.12
1024	8.61	23.41	950.95	349.85	20.86	56.70
2048	16.56	37.32	989.36	439.65	20.05	45.12
4096	31.73	62.07	1032.63	527.85	19.2 1	37.58
8192	62.50	103.73	1048.45	631.74	18.92	31.40
16384	122.50	170.67	1069.94	767.97	18.54	25.83
32768	238.26	232.84	1100.21	1125.81	18.03	17.62
65536	473.36	447.72	1107.58	1171	17.9 1	16.94
125000	884.22	656.86	1130.94	1522.39	17.54	13.03
196608	1339.21	994.30	1174.47	1581.88	16.89	12.54
393216	2492.65	1712.66	1262.69	1836.74	15.71	10.8
786432	4836.71	3124.03	1300.77	2013.89	15.25	9.85
3145728	18217.75	10656.62	1381.39	2361.52	14.36	8.4

cipher algorithms are adjusted also to encrypt 128 by 128 bits of the plain text. For this, the keystream generator produces 128 bits of keystream to be combined with 128 bits of plain text by an XOR operation. Recall that for each new encryption, a new IV is produced.

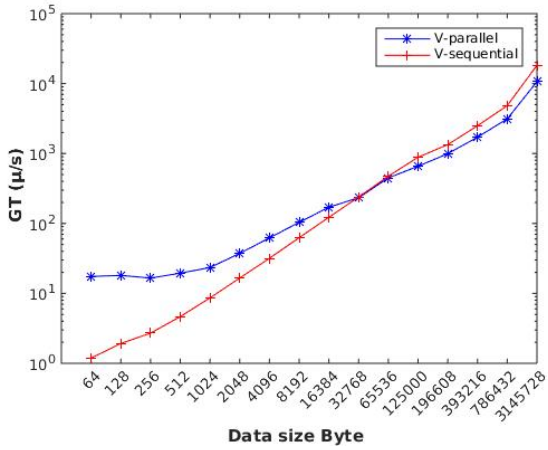
From the speed performance of the PCNGs given in Tables 4.13, 4.14 and 4.15, the PCNGs are faster



(a) Generation Time of the proposed CM-PCNG.



(b) Generation Time of the proposed DM-PCNG.



(c) Generation Time of the proposed CS-PCNG.

Figure 4.15 – Generation Time of the proposed PCNGs.

Table 4.16 – Computing performance of some known pseudo random number generators.

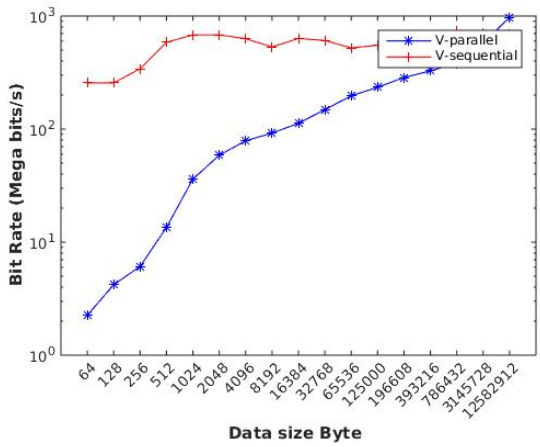
Pseudo random generator	NCpB
Wang et al., [267]	160
Akhshani et al., [19]	45
Abu Taha et al., [112]	17.3
CM-PCNG	24.68
DM-PCNG	21.2
CS-PCNG	12

when generating 128 bits of keystream in sequential programming. For this, we use sequential programming in the implementation of the three proposed stream ciphers.

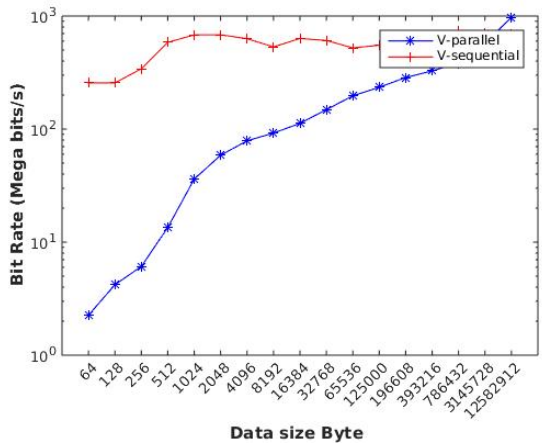
To evaluate the performance of the proposed stream ciphers, a number of experiments were performed based on several color images, which were used as plain images having the sizes $(128 \times 128 \times 3)$, $(256 \times 256 \times 3)$, $(512 \times 512 \times 3)$ and $(1024 \times 1024 \times 3)$.

4.3.1 Security analysis of the proposed stream ciphers

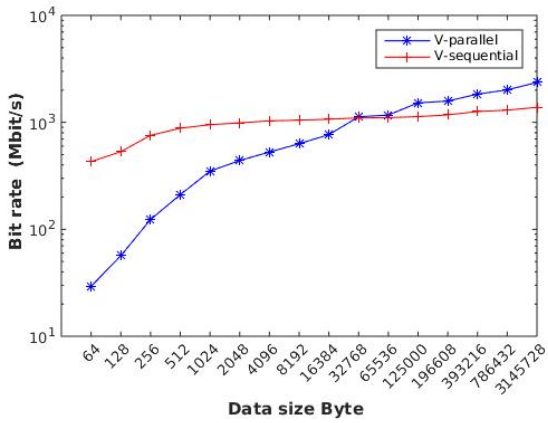
In the following part, some classical cryptanalytic analysis is performed.



(a) Bit Rate of the proposed CM-PCNG.



(b) Bit Rate of the proposed DM-PCNG.



(c) Bit Rate of the proposed CS-PCNG.

Figure 4.16 – Bit Rate of the proposed PCNGs.

Key space analysis

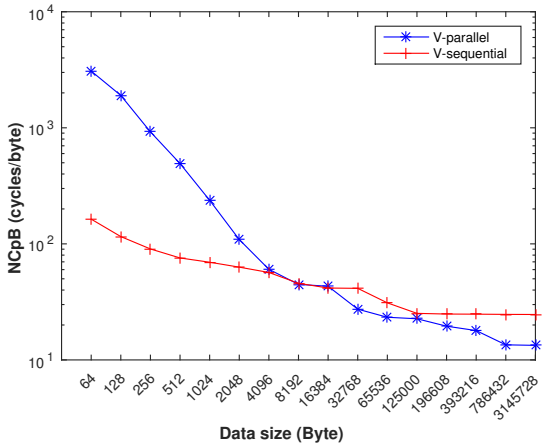
For any secure crypto-system, the key space should be large enough to resist a brute-force attack. The spaces of the secret keys for the proposed stream ciphers are related to the key sizes of the used PCNGs, given by Eqs.4.13, 4.14 and 4.15 respectively. Consequently, the key spaces of the secret keys are equal to 2^{189}, 2^{159} and 2^{147} for the proposed stream ciphers CM-SC, DM-SC and CS-SC respectively. Therefore, they are large enough to resist any brute-force attacks.

Key sensitivity analysis

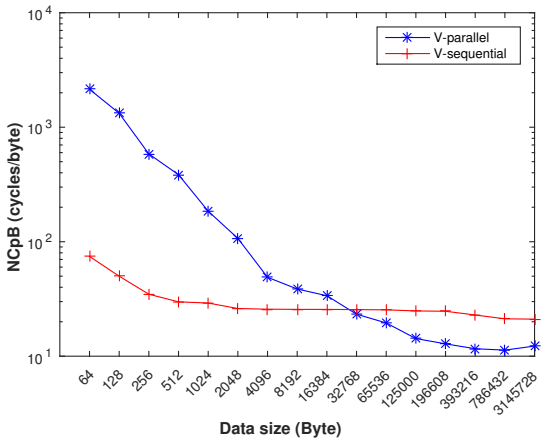
An efficient stream cipher should be very sensitive to the secret key. The change of a single bit in the secret key should produce a completely different encrypted image. Indeed, to verify this feature, we calculate the average Hamming Distance $D_H(X, Y)$ (using 100 secret keys), between two ciphered images C_1 and C_2, of the same plain image P, with only one change in the least significant bit of the parameter Pp. $D_H(C_1, C_2)$ is given by the following equation :

$$D_H(C_1, C_2) = \frac{1}{Nb} \times \sum_{n=1}^{Nb} (C_1[n] \oplus C_2[n]) \tag{4.26}$$

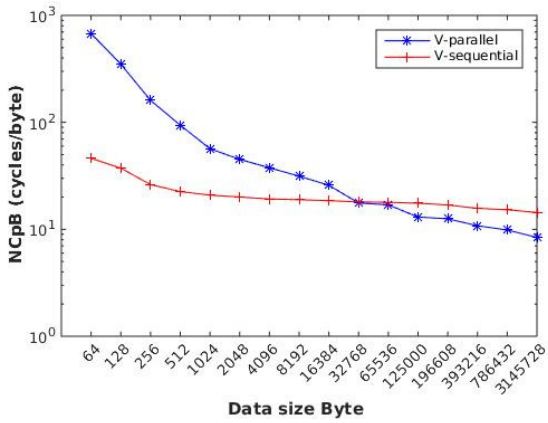
With Nb is the number of bits in an encrypted image. The obtained results of the Hamming distance for three different ciphered images by the three algorithms are close to the optimal value of 50% (see



(a) NCpB of the proposed CM-PCNG.



(b) NCpB of the proposed DM-PCNG.



(c) NCpB of the proposed CS-PCNG.

Figure 4.17 – NCpB of the proposed PCNGs.

Table 4.17). Such results are obtained regardless of the position of the changed bit in the secret key. This demonstrates that the proposed algorithms are highly sensitive to the secret key. Other common measures used to test sensitivity to the secret key on the encrypted image when changing one bit are the Number of Pixel Change Rate (NPCR) and Unified Average Changing Intensity (UACI). The former is used to measure the number of different pixels between the two images, whereas the latter is used to measure the average intensity difference. Let $C_1[i,j,p]$ and $C_2[i,j,p]$ be the (i,j,p)th pixel of two ciphered images C_1 and C_2, respectively. The NPCR and UACI are defined by Eqs.(4.27) and (4.29), respectively.

$$NPCR = \frac{1}{L \times C \times P} \times \sum_{p=1}^{P} \sum_{i=1}^{L} \sum_{j=1}^{C} D[i,j,p] \times 100\% \tag{4.27}$$

$$D[i,j,p] = \begin{cases} 0, & \text{if } C_1[i,j,p] = C_2[i,j,p] \\ 1, & \text{if } C_1[i,j,p] \neq C_2[i,j,p] \end{cases} \tag{4.28}$$

$$UACI = \frac{1}{L \times C \times P \times 255} \times \sum_{p=1}^{P} \sum_{i=1}^{L} \sum_{j=1}^{C} |C_1 - C_2| \times 100\% \tag{4.29}$$

Table 4.17 also shows the obtained results of NPCR and UACI for the previous three ciphered images. The resulting values are near to the expected values of NPCR and UACI which are 99.60% and 33.46%, respectively [281] [158]. So, as we can see, the proposed algorithms are very sensitive with respect to small changes in the secret Key.

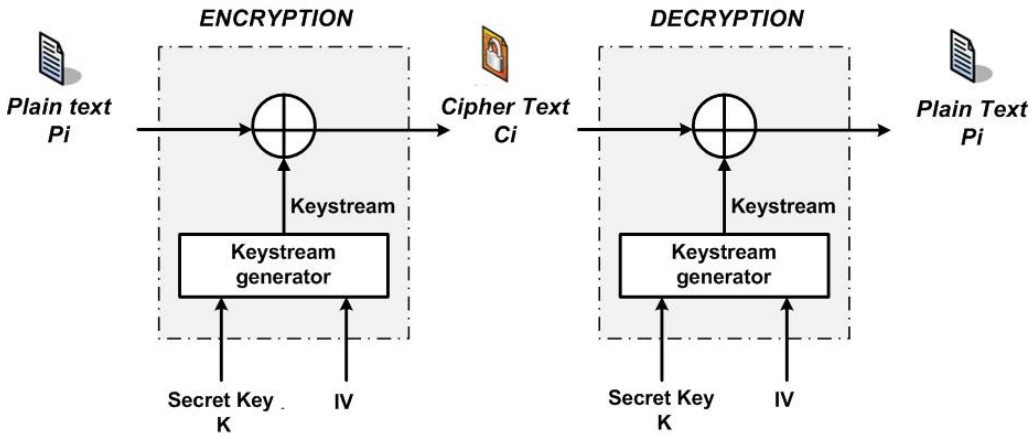


Figure 4.18 – General scheme of a stream cipher.

Table 4.17 – D_H, NPCR and UACI performance.

		Baboon	Peppers	Lena
CM-SC	D_H	0.500022	0.500009	0.500015
	$NPCR$	99.60918	99.60866	99.61024
	$UACI$	33.46386	33.46330	33.465842
DM-SC	D_H	0.500017	0.500018	0.500018
	$NPCR$	99.60954	99.60987	99.60899
	$UACI$	33.469	33.46388	33.47010
CS-SC	D_H	0.499995	0.499985	0.499995
	$NPCR$	99.60415	99.60412	99.61072
	$UACI$	33.46402	33.46258	33.46153

Chosen plaintext attack analysis

The proposed stream ciphers resist to the chosen plain text attack. Assume one knows a plaintext P_i and the corresponding ciphertext C_i. For each new encryption of the same P_i and using the same secret K, we utilize a new IV. Therefore, as the algorithms have a high sensitivity to small changes in the secret key or the IV, a new ciphertext C_i' is obtained.

4.3.2 Statistical analysis of the proposed stream ciphers

To prove the robustness of the proposed stream ciphers against statistical attacks, we perform the following experiments: histogram, chi-square test and correlation analysis.

Histogram analysis

A key property of a secure stream cipher algorithm is that the encrypted image should have a uniform distribution. We applied the proposed CM-SC stream cipher on three different plain images (Lena, Baboon and Peppers) of size ($512 \times 512 \times 3$). The obtained results are given in Figures.4.19, 4.20 and 4.21. On each one, we show (a) the plain image, (b) the corresponding cipher image, (c) the histogram of the plain image and (d) the histogram of the ciphered image.

We can visually observe that the histograms of the encrypted images are uniform and significantly different from those of the plain-images. The same visual results are obtained for the two proposed algorithms DM-SC and CS-SC.

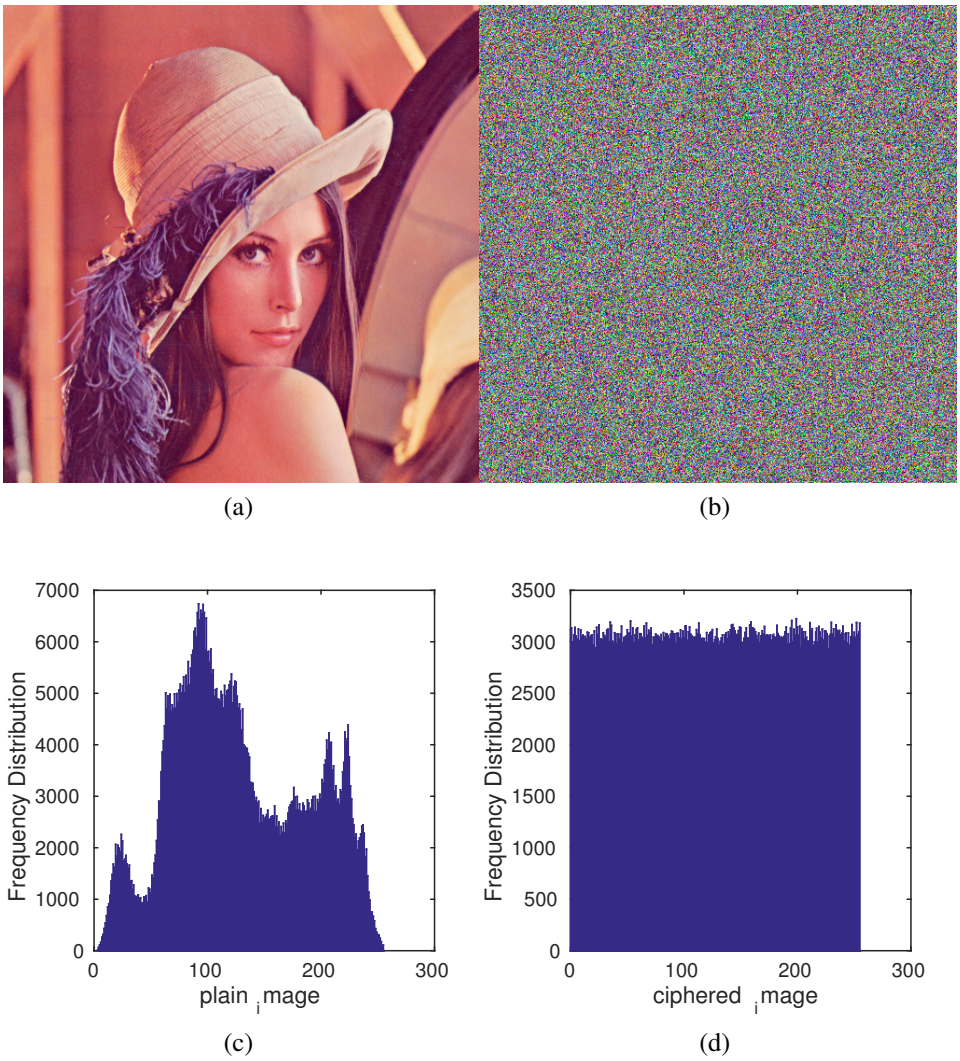


Figure 4.19 – (a) Lena image, (b) Lena cipher image, (c) the histogram of Lena image and (d) the histogram of the ciphered Lena image.

Chi-square test analysis

In order to assert the uniformity of the encrypted images, we apply the Chi-Square test. The experimental Chi-Square test χ^2 is calculated by the following formula:

$$\chi^2_{\text{exp}} = \sum_{i=0}^{K-1} \frac{(O_i - E_i)^2}{E_i}. \tag{4.30}$$

Where K is the number of levels (here 256), O_i are the observed occurrence frequencies of each color level (0-255) in the histogram of the ciphered image, and E_i is the expected occurrence frequency of the uniform distribution, given here by $E_i = (L \times C \times P)/256$ [77][85].

We compare the experimental value with the theoretical value obtained for a threshold α=0.05 and a degree of freedom K -1 = 255. To prove the uniformity of a sequence, the experimental value of Chi-Square must be lower than the theoretical one $\chi^2_{\text{exp}} < \chi^2_{th}$ (255, 0.05). The smaller the experimental value of Chi-Square is than the theoretical one, the better the uniformity of the histogram.

In Table 4.18, we reported the experimental and theoretical values of the Chi-Square test for the three ciphered images (Baboon, Peppers, and Lena) obtained by the proposed algorithms. We note that for the three images, $\chi^2_{\text{exp}} < \chi^2_{th}(255, 0.05)$. In addition, images encrypted by the CM-SC algorithm have a better uniform distribution than those encrypted by the DM-SC and CS-SC algorithms. Also, the distribution of

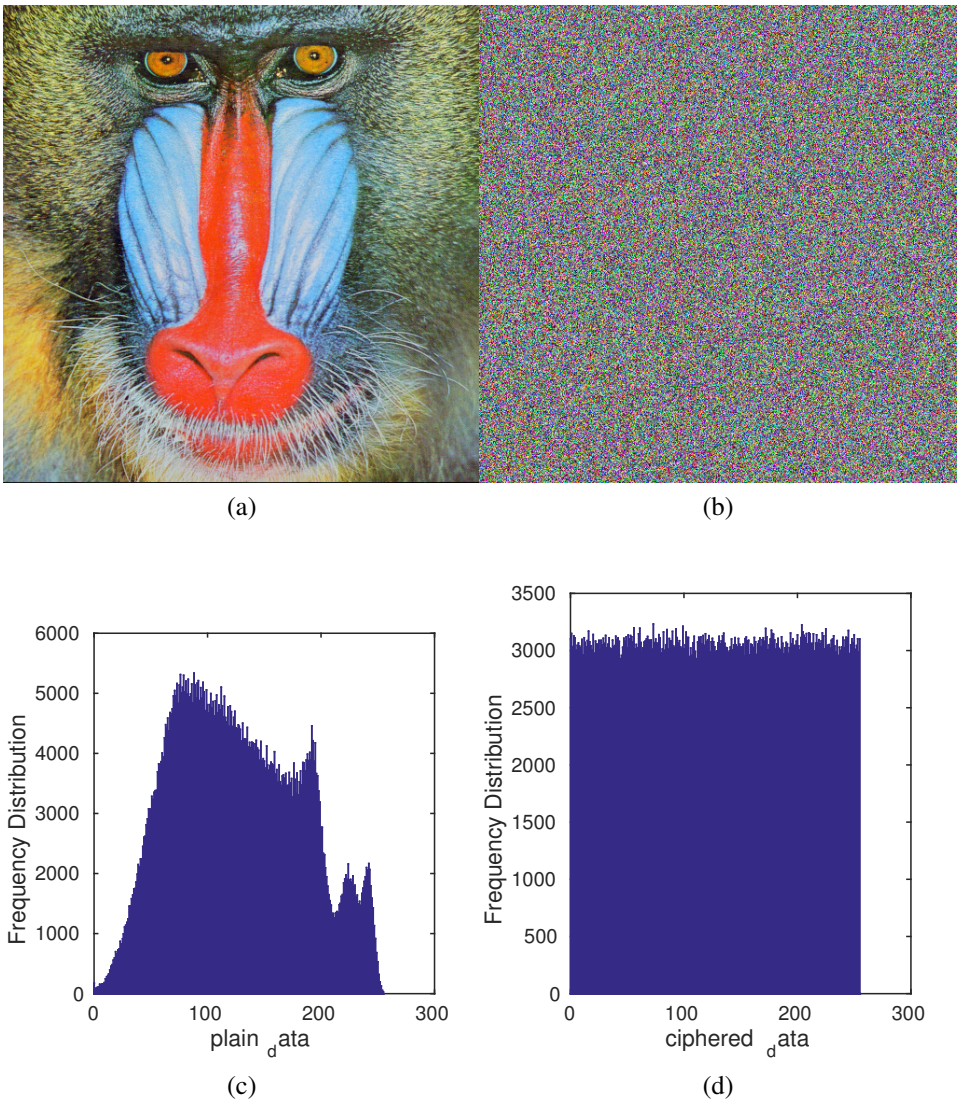


Figure 4.20 – (a)Baboon image, (b) Baboon cipher image, (c) the histogram of Baboon image and (d) the histogram of the ciphered Baboon image.

images encrypted by the CS-SC is more uniform than those of DM-SC algorithm.

Table 4.18 – Theoretical and experimental values of the Chi-Square test for the proposed stream ciphers.

	Baboon	Peppers	Lena
χ^2_{th}	293.247	293.247	293.247
χ^2_{exp} of CM-SC	211.966	239.10	252.703
χ^2_{exp} of DM-SC	267.293	265.37	262.31
χ^2_{exp} of CS-SC	253.5	259.1	253.2

Correlation analysis

The adjacent pixels in a plain image normally have strong correlation. Also, the pixels in an encrypted image, with a high security level, is expected to be randomly distributed. Therefore, a good encryption scheme should have the ability to efficiently reduce the correlation among adjacent pixels. We measured

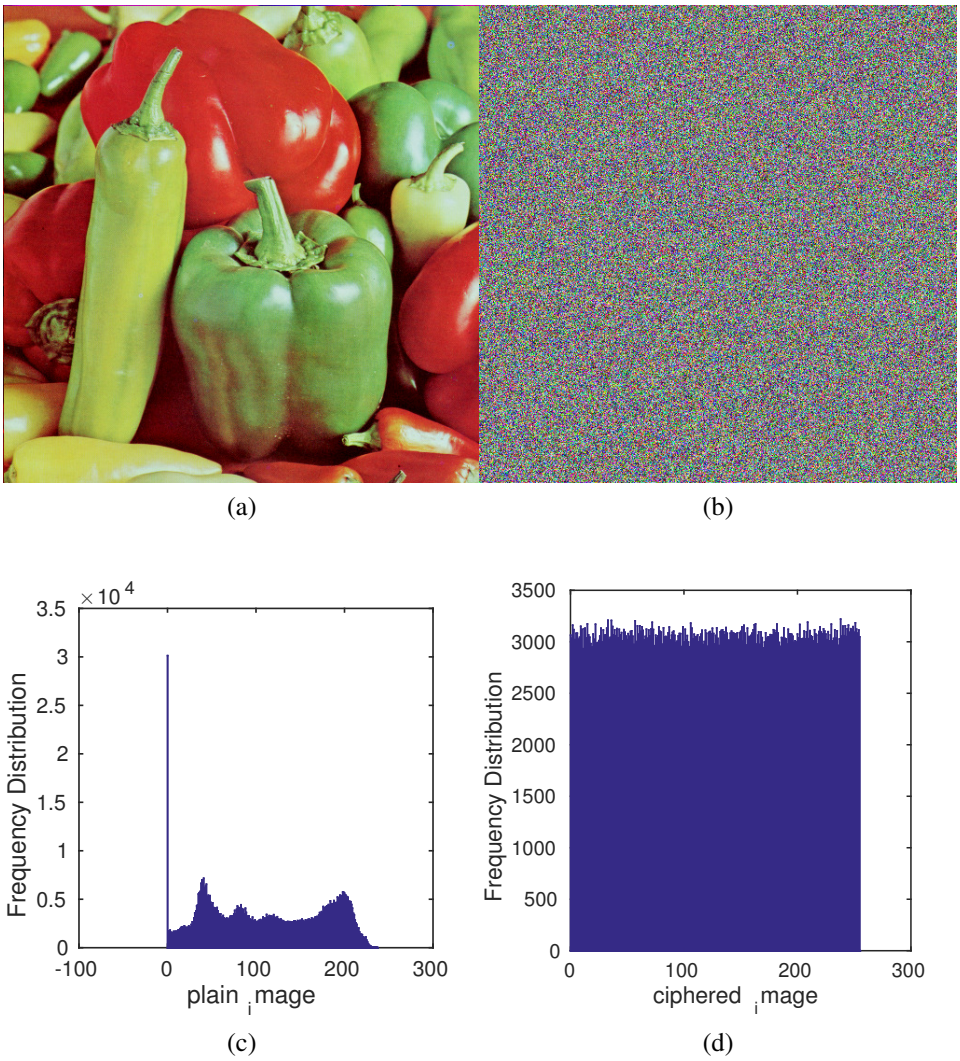


Figure 4.21 – (a) Peppers image, (b) Peppers cipher image, (c) the histogram of Peppers image and (d) the histogram of the ciphered Peppers image.

the correlation coefficient between adjacent pixels, selected randomly from three directions: horizontally, vertically and diagonally. The correlation coefficient ρ_{xy} of adjacent pixels is calculated by the following equation:

$$\rho_{xy} = \frac{\sum_{i=1}^{M}(x_i - \frac{1}{M}\sum_{j=1}^{M} x_j)(y_i - \frac{1}{M}\sum_{j=1}^{M} y_j)}{[\sum_{i=1}^{M}(x_i - \frac{1}{M}\sum_{j=1}^{M} x_j)^2]^{1/2} \times [\sum_{i=1}^{M}(y_i - \frac{1}{M}\sum_{j=1}^{M} y_j)^2]^{1/2}}. \tag{4.31}$$

where x_i and y_i form i^{th} pair of horizontally/vertically/diagonally adjacent pixels, M is the total number of pairs of horizontally/ vertically/diagonally adjacent pixels.

In Table 4.19, we give the obtained correlation coefficients in horizontal, vertical and diagonal directions of 1000 pairs of adjacent pixels of the plain images mentioned above and their corresponding ciphered images.

These results show that the correlation coefficients of the plain images are close to 1 while those of encrypted images are near to 0. Then, the proposed encryption schemes generate an image with uncorrelated adjacent pixels. This indicates that the proposed algorithms are secure against statistical attacks.

In addition, such results are confirmed in Figure 4.22, which shows the correlation of two horizontally, vertically and diagonally adjacent pixels in the plain and ciphered Baboon image ($512 \times 512 \times 3$) using the CM-SC stream cipher. Similar results are obtained when using the DM-SC and CS-SC algorithms.

Table 4.19 – Correlation coefficients between pairs of plain and encrypted images.

Image	Direction	Plain Image	Ciphered Image by CM-SC	Ciphered Image by DM-SC	Ciphered Image by CS-SC
Lena	Horizontal	0.993176	0.008713	0.00131	-0.00578
	Vertical	0.997055	0.008154	0.00121	-0.00467
	Diagonal	0.988176	0.008324	0.00117	-0.00509
Baboon	Horizontal	0.99233	0.00157	0.00317	0.00278
	Vertical	0.99649	-0.00151	-0.00326	0.00261
	Diagonal	0.98712	-0.00158	-0.00309	0.00273
Peppers	Horizontal	0.96775	0.00320	0.01183	0.00508
	Vertical	0.95753	-0.00309	0.00016	0.00541
	Diagonal	0.93002	-0.00306	0.01480	0.00527

NIST test analysis

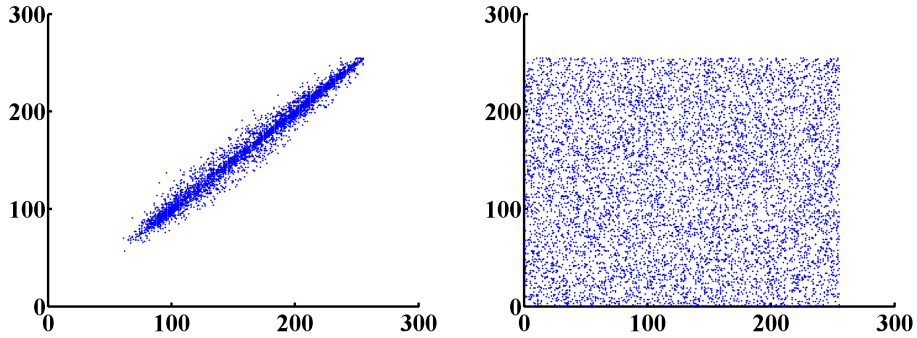
To evaluate the performance of the proposed algorithms, we use the NIST test. For that, we encrypted 100 different binary sequences of plain text, P_1, P_2 and P_3; using the proposed stream ciphers CM-SC, DM-SC and CS-SC, respectively, each one with a different secret key and containing 10^6 bits. We present the results of NIST for the encrypted sequences C_1, C_2 and C_3 in Table 4.20, with a level of significance of the test $\alpha = 0.01$. Results show that sequences C_1, C_2 and C_3 have successfully passed all NIST tests. Therefore, the proposed stream ciphers can resist statistical attacks.

Table 4.20 – P-values and Proportion results of NIST for the proposed stream ciphers.

	CM-SC		DM-SC		CS-SC	
Test	**P-value**	**Proportion**	**P-value**	**Proportion**	**P-value**	**Proportion**
Frequency test	0.225	100	0.760	98	0.319	98
Block-frequency test	0.798	99	0.051	93	0.067	100
Cumulative-sums test	0.696	100	0.527	100	0.657	98
Runs test:	0.575	100	0.898	100	0.514	99
Longest-run test	0.137	98	0.596	100	0.760	99
Rank test	0.798	99	0.367	100	0.067	99
FFT test	0.554	99	0.154	100	0.249	100
Non-periodic-templates	0.483	98.966	0.522	99	0.506	99.223
Overlapping-templates	0.720	100	0.063	98	0.851	99
Universal	0.834	100	0.276	99	0.130	100
Approximty entropie	0.740	99.000	0.834	99	0.475	99
Random-excursions:	0.483	98.321	0.576	99.414	0.364	98.843
Random-excursions-variant	0.312	98.425	0.615	98.351	0.475	99.383
Serial test	0.257	99.500	0.519	98.500	0.480	98
Linear-complexity	0.868	100	0.182	98	0.016	99

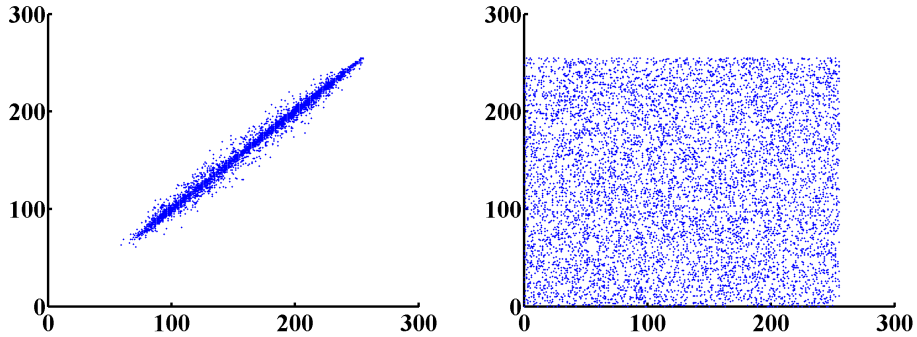
4.3.3 Computing performance of the proposed stream ciphers

We have performed the computing performance of the proposed stream ciphers. In this implementation, we do not parallelize processes and operations. We evaluated the computing performance as follows: for



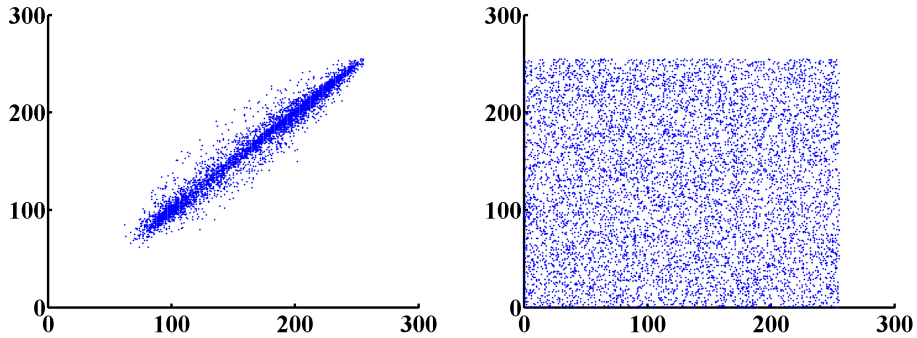
(a) Distributions of two horizontally adjacent pixels in the plain 'Lena' image

(b) Distributions of two horizontally adjacent pixels in the encrypted 'Lena' image



(c) Distributions of two vertically adjacent pixels in the plain 'Lena' image

(d) Distributions of two vertically adjacent pixels in the encrypted 'Lena' image



(e) Distributions of two diagonally adjacent pixels in the plain 'Lena' image

(f) Distributions of two diagonally adjacent pixels in the encrypted 'Lena' image

Figure 4.22 – Distributions of two horizontally/ vertically/ diagonally adjacent pixels in the plain and encrypted 'Lena' images.

100 different keys, we executed our algorithm and then, we calculated the average encryption time in (Micro second), the average encryption throughput (ET) in (MByte) and the number of cycles per byte (NCpB).

$$ET = \frac{Image\ size(MByte)}{Encryption\ time(s)} \tag{4.32}$$

$$NCpB = \frac{CPU speed(Hz)}{ET(Byte/s)} \tag{4.33}$$

The obtained results of computing performance for the proposed stream ciphers CM-SC, DM-SC and CS-SC are given in Tables 4.21, 4.22 and 4.23, respectively. We remark that globally, the speed performance of the stream ciphers is approximately 17 % less than that of the corresponding PCNGs. In Table 4.24, we

give a comparison of NCpB for the proposed algorithms (for Lena $512 \times 512 \times 3$) with other chaos-based algorithms and the most known stream ciphers [52][28].

Table 4.21 – Speed Performance of the proposed CM-SC stream cipher.

Data Size (Byte)	Encryption Time (μs)	Encryption throughput (Mbits/s)	NCpB
512	21.26	183.73	107.96
1024	37.05	210.86	94.07
2048	44.28	352.86	56.21
4096	73.58	424.70	46.71
256×256×3	2403.04	624.20	31.78
512×512×3	8511.00	704.97	28.14
1024×1024×3	32710.50	733.70	27.04

Table 4.22 – Speed Performance of the proposed DM-SC stream cipher.

Data Size (Byte)	Encryption Time (μs)	Encryption throughput (Mbits/s)	NCpB
512	12.50	312.50	63.48
1024	23.35	334.58	59.29
2048	34.70	450.28	44.05
4096	49.05	637.10	31.14
256×256×3	2168.31	691.783	28.71
512×512×3	7267.27	825.6195	24.03
1024×1024×3	28808.11	833.0987	23.81

Table 4.23 – Speed Performance of the proposed CS-SC stream cipher.

Data Size (Byte)	Encryption Time (μs)	Encryption throughput (Mbits/s)	NCpB
512	10.66	384.13	51.64
1024	18.70	437.89	45.3
2048	24.54	667.45	29.72
4096	28.21	1161.40	17.08
256×256×3	1059.5	1415.76	14
512×512×3	3964.52	1586.94	12.5
1024×1024×3	1834964.6	1755.47	11.3

We observe that the proposed algorithms have a better speed performance than the cited chaos-based algorithms except that of [265]. However, in [265], the authors do not explain the measurement method used to obtain such excellent results, given that the complexity of their system is similar to ours. Compared to the AES-CTR, Rabbit, HC-128, Salsa20/12 and SOSEMANUK, the obtained performance is not as good. However, the non linearity of the proposed systems is higher than the other systems, consequently, its robustness against known attacks is higher.

4.4 Conclusion

In this chapter, we designed and developed three novel chaos-based stream ciphers, defined on finite precision N=32, for secure data transmission in real-time applications.

Table 4.24 – Comparison of NCpB performance between different algorithms.

Algorithms	NCpB
Ref.[17]	321
Ref.[151]	226
Ref.[265]	1.77
CM-SC	28.14
DM-SC	24.03
CS-SC	12.5
AES-CTR	21.2
Rabbit	9.5
HC-128	14.4
Salsa20/12	9.9
SOSEMANUK	10.5

The high efficiency obtained from these crypto-systems is due to the designed PCNGs structure. Indeed, their architectures integrate chaotic maps weakly coupled using a predefined matrix or coupled by a binary diffusion matrix. The CM-PCNG and DM-PCNG uses a chaotic multiplexing technique while the CS-PCNG includes a swap technique. The used techniques permit to decrease the degradation caused by the descretizing process and the finite precision N. For that, we do not include in the proposed architectures the perturbation technique and the recursive structure proposed in Chapter 3. Simulation tests, security analyses and computing performance were carried out to prove the efficiency in terms of robustness and speed performance of the proposed PCNGs and stream ciphers. The obtained results show that the proposed stream ciphers are robust against known attacks of the literature and can be used in practical applications including secure network communication.

5

Energy and Power consumption evaluation and Real-Time Implementation of the proposed stream ciphers

5.1 Introduction

The concept of Internet of Things (IoT) is becoming an increasingly growing topic, due to huge advancements in wireless networking technology and standardization of communication protocols [86] [82]. The core idea of this concept lies in the presence of everyday physical objects known as things which are connected to the internet. Interconnection between things is made possible by technologies such as Radio Frequency IDentification (RFID), Wireless Sensor Networking (WSN), cloud servicing, machine-to-machine interfacing (M2M), etc.

Secure data transmission in the IoT is a very significant issue because confidential and proprietary information have to be transmitted, especially in healthcare applications. Unfortunately, existing cryptographic techniques developed for enterprise and desktop computing might not satisfy embedded applications with strong real-time requirements as they can be too slow, huge and very power consuming [128].

Smart devices of the IoT, including sensors, are inherently resource constrained with regard to memory, communication bandwidth, processing power and energy [100]. The most widespread energy sources are typically batteries, renewable energy sources of the environment, or both. Most of wireless devices should be autonomous i.e. operate for several years even for decades without any human intervention. Energy consumption is consequently a central performance factor since it directly impacts lifetime of the device. Hence, a challenging topic concerns the design of efficient and lightweight (from the point of view of energy and processing time consumption) cryptographic techniques to guarantee secure data transmission in the IoT. Such techniques should fit the low energy, computation and memory capabilities of cyber-physical systems and provide an optimized security/cost/performance trade-off [74]. This is why the new field of Light Weight Cryptography (LWC) is emerging.

The four main characteristics that differentiate one crypto-system from another are: ability to secure the protected data, speed i.e. computational complexity, energy consumption and memory required in doing so. The first objective of this chapter is to study the performance of two chaotic stream ciphers that we recently designed in terms of energy and power consumption and memory assessment, since we have presented their security and speed performance in the previous chapter. We will show that the proposed stream ciphers are lightweight crypto-systems. Compared to other crypto-systems of the literature, we demonstrate that our designed stream ciphers are suitable for practical secure applications of the IoT with constrained resources environment. In section 5.2, we present some energy and power measurements tools proposed in the literature. And we give energy and power consumption measurements for our proposed algorithm. Section 5.3 deals with the requirements of the designed stream ciphers in terms of RAM and code size requirements.

The second part of this chapter concerns the actual integration of the proposed crypto-systems with real-time features. Indeed, the development of real-time embedded systems is continuously growing. In some real-time applications such as automotive, robotic, tele-medicine and avionics, real time performance becomes critical because each component of the system must work in cooperation and coordination. "The correctness of a real-time system depends not only on the correctness of the logical result of the computation but also on the physical time when this result is produced"[251] The role of an operating system in communication devices is important. Besides scheduling the real-time tasks for access to the processor, it realizes synchronization between tasks and in particular it controls access to multiple resource through specific resource management protocols.

In order to study the effective behaviour of our crypto-systems as part of a real-time application, we implemented the proposed stream ciphers using a famous RTOS (Real Time Operating System) named Xenomai . In section 5.4, we give a brief introduction to the field of real-time computing, especially to the topics which are relevant in the context of this thesis. We also define a number of basic terms including embedded systems, real-time systems, real-time operating system, etc. In addition, we report in subsection 5.4.5 the model used to implement the real-time crypto-systems using Xenomai which presents a software framework able to give real time capabilities to the operating system. Tools for execution time measure-

ments and resulting results about computation performance are given in 5.4.6. Section 5.5 concludes the chapter.

5.2 Energy and Power consumption evaluation

Energy and power consumption are currently a major concern in the design and usage of high performance crypto-systems. Many researchers in the energy field are contributing to elaborate multiple tools and libraries to measure energy and power consumption of software components. In spite of their efforts, there is no current standard for energy and power measurements. Traditionally, the energy consumption of an application is measured using power meters and performance counters. In the last years, different tools have been proposed to provide access to power and energy measurements. Each tool offers its own level of precision and intrusion. The choice of a concrete tool is often a matter of availability, compatibility and precision. In the next subsection 5.2.1, we present different existing tools that have been proposed in the literature. In particular, we have selected several ones to measure the energy and power consumption of our proposed algorithms, namely the 'the Intel's RAPL technology' and 'powertop' tool. Results on our measurements are reported in subsections 5.2.2 and 5.2.3 respectively.

5.2.1 Energy and power measurement tools

Several software tools are being used to measure energy consumption. We classify the set of existing tools in different groups according to their interaction with hardware and software:

External devices

There are many energy measurement systems that have been used to measure energy consumption and energy efficiency outside the experimental nodes. These measurements can be done without interfering the experiment. Nonetheless, they could be unsuitable for experiments that require high precision measures. These measurement systems include:

— The *power distribution units* (PDUs) are devices that are used to supply energy to data center servers and at the same time have energy monitoring capabilities. They are used to facilitate, control, and optimize energy generation and transmission. PDUs are usually composed of a number of outlets where devices are connected [159].

— The *PowerMon2* presents a low-cost power monitoring device that operates inside commodity computer systems. It is used to analyse performance and power consumption trade-offs in computer applications. PowerMon2 measures voltage and current on the individual DC power rails between a system's power supply and the motherboard and peripherals [37].

— *PowerPack* presents a power/energy/performance profiling infrastructure [98]. It is a combination of hardware (e.g., sensors and digital meters) and software (e.g., drivers, instrumentation APIs, benchmarks, and analysis tools) and used to evaluate energy efficiency and power-aware techniques for parallel applications.

— *PowerScope* [87] is a tool for profiling energy usage by applications. It uses a digital multimeter to perform off-line analysis using statistical sampling. It provides a kernel-level interface (via system calls) to start and stop measurements; this requires modifying the operating system. PowerScope maps energy consumption to program structure, in much the same way that CPU profilers map processor cycles to specific processes and procedures.

— The *energy endoscope* [252] is an new embedded networked sensor platform architecture that combines hardware and software tools. It offers detailed, fine-grained real-time energy measurements.

Intra node tools

The intra-node group includes highly customized tools which have a restricted hardware platform and a costly development:

— The *Linux Energy Attribution and Accounting Platform* (LEA^2P) [225][226] presents an energy attribution and accounting architecture for multi-core systems that can provide accurate, per-process energy information of individual hardware components. (LEA^2P) consists on a hardware-assisted direct energy measurement system that integrates seamlessly with the host platform and provides detailed energy information of multiple hardware elements at millisecond-scale time resolution. These informations are passed into the Linux kernel and made available via the */proc* file system and can be read in-band.

— The *Seoul National University Energy Scanner* (SES) [237] is an integrated energy monitoring tool, consists in a power instrumentation board that connects via PCI(E) bus, supporting various power measurement tools although it was not a generic API. SES collects power consumption information in a cycle-by-cycle resolution and associates the collected power data with C program and assembly language source code.

— Bellosa [41] presents *Joule Watcher*, an approach based on information about active hardware units (e.g., integer/floating-point unit, cache/memory interface) gathered by event counters to establish a thread-specific energy measurement.

Other libraries

Other energy measurement tools have been proposed which are focused on a general API to access the information provided by the hardware counter but are restricted by the platform:

— The *performance API* (PAPI) [56] is a standard application programming interface (API) for accessing hardware performance counters available on most modern CPUs. It provides the ability to measure system's energy and power consumption. This is offered by using the RAPL interface (which will be later described)[269].

— *LIKWID* ("Like I Knew What I am Doing") [261] is a performance-oriented library that is targeted towards applications in a Linux environment and does not require any kernel patching. It has the ability to measure energy consumption by measuring performance counter metrics over the complete run time of an application or, with support from a simple API, between arbitrary points in the code. LIKWID is suitable for Intel and AMD processor architectures. It only supports x86-based processors.

— The Intel® Energy Checker Software Developer Kit (Intel® EC SDK) provides tools to help developers design energy-efficient applications. It has a small set of simple APIs for software instrumentation to measure the energy consumption. It is restricted to Intel architectures [110].

— The *NVIDIA Management Library* (NVML) is a C-based API for monitoring and managing various states of the NVIDIA GPU devices. It provides the power usage and the power limits of the supported products [72].

— The *PowerAPI* is a software library to monitor the energy consumed at the process-Level. It provides an application programming interface (API) that estimates, in real-time, the energy consumed at the granularity of a system process, from formulas based on CPU, memory and disk usage metrics. PowerAPI supports only a single Power-Spy2 PDU [53].

— The *Energy Measurement Library* (EML) [60] is a software library created to facilitate the experimentation of energy consumption in distributed systems. It is based on using an API that offers multiple options to users in order to speed up the measurement and experimentation process. It provides energy information by accessing energy counter measurements through PAPI.

5.2.2 Measurement of energy consumption with RAPL

Running Average Power Limit (RAPL) tool

In this thesis, we use another software tool which is named "Running Average Power Limit" (RAPL) interface, to measure energy consumption of our proposed algorithms.

Recently, the Intel Company has equipped its systems with a new tool for obtaining fine-grain energy models: on board energy sensors for measuring the energy consumed by on-core hardware components and the energy consumed by the code that runs on these components. Intel introduced these sensors called "Running Average Power Limit" (RAPL) with their Sandy Bridge micro-architecture [68]. RAPL provides a set of counters which reports energy and power consumption information. RAPL is available in newer versions of the Xeon server-level CPUs. RAPL is not an analog power meter, but rather it uses a software power model. This one estimates energy usage of the CPU-level components, listed in Table 5.1, through hardware performance counters and I/O models [222].

Table 5.1 – List of available RAPL sensors.

RAPL_PKG	Whole CPU package
RAPL_PP0	Processor cores only
RAPL_PP1	A specific device in the uncore
RAPL_DRAM	Memory controller

RAPL reports various energy readings measurements that include: the energy consumption of the total processor package (PKG), the total combined energy used by all cores (Power-Plane 0 (PP0) which includes all processor caches) and the energy readings for the DRAM interface (DRAM). Also, some versions of SandyBridge chips report power usage due to the on-board GPU (PP1).

Experiment methodology

The results of RAPL model are available to the user via a model specific register (MSR), with an update frequency on the order of milliseconds [222]. To access MSRs, we require a ring-0 access to the hardware. Typically, only the OS kernel can do this. This means that accessing the RAPL values needs a kernel driver. Currently, Linux does not provide such a driver. To overcome this problem, we use the "MSR driver" that exports MSR access to user space via a special device driver. When the MSR driver is enabled and given proper read-only permission, the PAPI can access these registers without needing kernel support.

On Linux, the "MSR driver" is not auto-loaded. On modular kernels we might need to use the following command to load it explicitly before use: *sudo modprobe msr*.

We use the RAPL interface and the MSR driver to measure the energy consumption of the encryption task. Energy consumption presents the difference of the energy amount energy available in the CPU level components before starting and after completing the encryption function.

Experimental results on energy consumption

We have conducted our energy measurement experiment for the two stream-cipher algorithms CM-SC and CS-SC proposed respectively in [114] and [113]. Table 5.2 gives the different average values for energy consumption of the two stream ciphers CM-SC and CS-SC which are compared to that of Rabbit and HC-128.

The different average values of energy consumption are calculated after running 100 times the stream ciphers using 100 different secret keys and the Lena image ($256 \times 256 \times 3$).

Table 5.2 – Energy consumption (J).

Stream cipher	CM-SC	CS-SC	Rabbit	HC-128
PKG energy(J)	0.078613	0.022672	0.013855	0.038768
PP0 (J)	0.036316	0.010297	0.006104	0.020316
PP1 (J)	0.007568	0.000112	0.000150	0.000030
DRAM (J)	0.012939	0.002669	0.001648	0.003806

5.2.3 Measurement of power consumption with PowerTOP tool

We also report a power estimation through the *PowerTop* Linux tool provided by Intel, which permits to diagnose issues with power consumption and power management. This tool helps the user to point out the power inefficiencies of a program. It shows how well the different hardware power-saving features are used. It reports the software components that prevent optimal usage [15] [136]. It also returns a power estimation for each device.

To use PowerTOP, one must meet the following requirements:

- an Intel processor,
- a Linux kernel 2.6.21 or better,
- PowerTOP is mainly useful for a laptop since it can only study battery consumption. For best results, when using PowerTOP on a laptop, do so when running on battery.

Power consumption measurements are given in table 5.3.

Table 5.3 – Power consumption in milliwatt (mW).

Stream cipher	CM-SC	CS-SC	Rabbit	HC-128
Power Estimation (mW)	3.4	2.9	2.7	3.18

Results shown in Tables 5.2 and 5.3 indicate that CS-SC algorithm has less energy and power consumption compared to CM-SC and HC-128 algorithms. Indeed, CS-SC consumes about 30% of energy consumed by the CM-SC algorithm, and 60% of that of the HC-128 algorithm. However, CS-SC consumes about two times more energy than Rabbit.

5.3 Memory assessment

Whatever the complexity of the cryptographic primitives in terms of computational overhead and memory usage, the hardware resources available must be performant enough to minimize the execution time of the secured applications. However, embedded devices often have inherent limitations in terms of memory space. Hence, it would be necessary to analyse how these primitives perform over highly-constrained devices. We calculate the requirements of the designed stream ciphers in terms of RAM consumption and code size. We use the FELICS framework which is an open source benchmarking framework [73]. We describe FELICS framework in section 5.3.1 and the main modifications to operate on the code so as to use FELICS. Also, we give the code size and RAM consumption values for our algorithms.

5.3.1 FELICS framework

FELICS (Fair Evaluation of Lightweight Cryptographic Systems) is a free, open source and flexible framework, which determine the performance of C and assembly software implementations of lightweight primitives on resource constrained devices commonly used in the IoT [263].

To the best of our knowledge, FELICS is the only open source and free framework that provide fair and consistent performance evaluation of software implementations of lightweight ciphers on different IoT embedded devices in the same usage conditions. FELICS is designed to work on Linux operating systems. It is able to benchmark C and assembly implementations of lightweight block and stream ciphers on three different devices: 8-bit AVR, 16-bit MSP and 32-bit ARM. The three extracted metrics are: code size, RAM consumption and execution time.

— The *code size* is measured in bytes. It corresponds to the amount of data that is stored in the Flash memory of the target device. To calculate the code size for each target device, FELICS uses the GNU size tool that lists the section sizes and the total size in bytes for a given binary file.
The code size extraction process is completely automated and can be done using the cipher code *size.sh* script for a given cipher implementation and a given evaluation scenario.

— The *RAM consumption* is split into data consumption and stack consumption. The data requirement represents the static RAM and it is given in bytes by the size of the constants stored in target device RAM. It includes data which is specific to each scenario such as data to encrypt, master key, round keys or initialization vectors. The stack consumption permits to assess, in bytes, the RAM used to store local variables.
The cipher *ram.sh* script is able to extract the RAM requirement for a given cipher in a given usage scenario.

— The *execution time* is expressed in number of CPU clock cycles required to execute a set of operations. It is computed from the system timer at the finishing point of the operations minus the system timer at the starting point of the operations. The cipher execution *time.sh* script extracts the execution time for a given cipher implementation and scenario.

FELICS framework source code, with implemented ciphers source code and performance figures are available on the web site [263]. Information on how to use the FELICS framework is also available.

In this research work, we use the FELICS virtual machine that we downloaded from its web site. We had to adapt our C implementation to the framework requirements to be able to evaluate the new stream ciphers.

A template cipher implementation is provided to integrate a new algorithm implementation into FELICS framework. The process of integration consists on filling the functions from the template cipher with the source code of the cipher according with the requirements described in the README file.

The main modifications required to integrate the new stream cipher algorithms are the followings:

- The cipher state size, key size and initialization vector size have to be defined in "*constants.h*" file.
- The constants used by the stream cipher must be declared in *"constants.h"* file and defined in *"constants.c"* or any other *"*.c"* file, except the predefined *"*.c"* files.
- Declaration and definition of the cipher test vectors in *"test_vectors.c"*.
- Implementation of the *setup* function in *"setup.c"* using the following function signature:
  void Setup(uint8_t *state, uint8_t *key, uint8_t *iv);
  Note: After running the setup, the key "key" and IV "iv" should not be modified.
  In our algorithmic structure, the setup function presents the two functions *key-setup* and *IV-setup*.
- Implementation of the encryption function in *"encrypt.c"* using the following function signature:
  void Encrypt(uint8_t *state, uint8_t *stream, uint16_t length);
- Add a description of the cipher implementation in *"implementation.info"* file, in the "ImplementationDescription" section. If there are other functions used in the cipher implementation and defined in *"*.c"* files, add the *"*.c"* name in *"implementation.info"* file, in the corresponding section(s) ("SetupCode", "EncryptCode").

FELICS parses the *implementation.info* file to be able to count the common source code and constants only once in the extracted metrics. The implementation of each of the required functions can be split into several files provided that the implementation information is correctly given in the *implementation.info* file.

Two evaluation scenarios are implemented for the stream cipher module:

- The first scenario (Scenario 0) is evaluated using the provided test vectors.

- The second scenario (Scenario 1) consists in encryption of 128 bytes of data. It covers the need for secure communication sensor networks and between IoT devices.

Results are given using the second scenario (Scenario 1).

To evaluate and test the cipher implementation, we do not need to compile the provided "*makefile*" that can build the cipher in different scenarios and test cases, either in debug or in release mode. If the cipher builds without errors or warnings and the two tests (test-cipher, test-scenario1) run as expected, the cipher implementation is correctly integrated into the FELICS framework.

In order to analyse and post process the results for the different metrics (Code Size, the RAM consumption and the execution time), the framework can export the extracted results for each scenario and target architecture in various formats, including: raw data table, CSV file, XML, MediaWiki table and LaTeX table.

5.3.2 Code size and RAM consumption results

The code size measures the amount of data that is stored in the Flash memory of the target device. The RAM consumption includes the stack requirements and data requirement. The former presents the maximum value of RAM used to store local variables. The later forms the static RAM, given by the size of the constants stored in target device RAM (such as data to encrypt, key, initial vectors...). Table 5.4 clarifies the code size and RAM consumption measurements of the four tested algorithms using FELICS framework. The code size and RAM consumption values, for Rabbit and HC-128 stream ciphers, are given by [163].

Table 5.4 – Code size and RAM consumption (bytes).

Stream cipher	Code size (bytes)	RAM consumption (bytes)
CM-SC	7240	660
CS-SC	6562	564
Trivium	5764	1516
Snow	12861	1741
Rabbit	1714	216
HC-128	23100	4556

These experimental results show that RAM and ROM required by the two designed stream ciphers CM-SC and CS-SC, are less than 8 KB and 32 KB respectively. Theses values are consequently very low and compatible with limitations of most of small devices. In conclusion, we may state that stream ciphers CM-SC and CS-SC are suitable for memory-constrained devices as those encountered in the IoT.

5.4 Integration of a real-time crypto-system

5.4.1 Definitions

This section recalls concepts and terminology relating to real-time computing such as embedded system, real-time system and real-time operating system.

Embedded system

An embedded system can be defined as a combination of computer hardware and software, with either fixed or programmable capabilities. It is particularly dedicated towards a specific kind of application device in which the resources are constrained. Embedded systems are commonly deployed in microprocessor or microcontroller that impose severe space, weight, and power constraints. They mostly have no user interface[169][253]. Today, embedded systems are the dominant form of computing due to the IoT,

vastly outnumbering "traditional" computers such as PCs. They are found in transportation systems like cars, trains, or air-planes as well as telecommunication equipment like cell phones and internet routers. Production and process control systems are among the many possible hosts of an embedded system.

Real-time system

According to [57], a real-time system is defines as "any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period". In other words, the correct behaviour of a real time system depends not only on the logical result of computation, but also on the time at which the results are produced[129][58]. If the timing constraints of the system are not respected, a system failure occurs or a sanction is incurred for the violation of the timing constraints. It is therefore essential that the timing constraints of the system are guaranteed to be met. It is also desirable for the system to achieve a high degree of utilization while satisfying the timing constraints of the system.

There are three fundamental characteristics that define the behaviour that a real time system must adopt: *predictability*, *determinism* and *reliability* [50]. Indeed, activities must be planned and executed within the specified time constraints. To ensure this, the real-time system designer must always be in the worst case. Determinism is to remove any uncertainty about the behaviour of individual activities, including when they must interact. Among the sources of non-determinism, we can cite the calculation load, the input-outputs, the interruptions, etc. Concerning the reliability constraint, the hardware and software components must be reliable in a real time context. This is why, in general, real time systems are designed to be *fault-tolerant*.

Real-time operating system

A real-time operating system (RTOS) is the underlying software or operating system (OS) that manages hardware resources and coordinates the execution of user applications. It is adopted to fulfill the demands of a real time system and to explicitly satisfy response-time constraints. So, it supports a scheduling method that guarantees response time especially to critical tasks[135]. Nowadays, there are many RTOSes (for example RTLinux,LynxOS, Windows CE, VxWorks, QNX,...) that offer the basic functions such as multi-tasking, synchronization, communication, resource access, fault tolerance and so on. However, they differ in the ease of use, performance and debugging facilities.

According to [152], what makes an OS a Real-Time OS (RTOS) is the imperative presence of several specific properties:

— A RTOS has to be multi-threaded and preemptible.
— The existence of thread (or process) priority notion,
— The OS has to support predictable thread synchronization mechanisms,
— A system of priority inheritance has to exist to limit priority inversion,
— OS behaviour should be predictable.

A RTOS allows real time application software to be easily designed and expanded. Functions can be added without the requirement of major changes to the software. Using a RTOS simplifies the design process by splitting the application code into different separate software elements called process or task. A RTOS allows the designer to make better use of the resources by providing precious services like semaphores, queues, time delays, timeouts, mailboxes, etc.

Simple real time applications are often implemented without any operating system and consequently with no pseudo-parallelism. In this case, the designer should take care of hardware resources accesses and should verify that timing constraints of the software operations be guaranteed. RTOSes are now used whenever the application gets complexity and requires modularity in software for debugging. RTOSes are now widely used in the development of embedded real time systems which are deployed in high number.

5.4.2 Classification of real-time systems

There are three classes of real time systems depending on the consequence of violating specifications on timing constraints mostly expressed in terms of deadlines for finishing the executions of programs. The three classes are defined here below.

We call a *soft real-time* system [108][102], the system in which response time is important but the consequence of missing a deadline is relatively mild. The system will still function correctly. Its performance is degraded without causing dramatic consequences on the environment under control and without calling into question the integrity of the system. Exceeding deadlines will have a certain cost for the system, which will result, for example, in lower calculation accuracy, lower data refresh rate, and so on.

In contrast, a *hard real-time system* [232][150][47] is a system for which all processing operations must imperatively respect all their temporal constraints in nominal operating condition. The system must be predictable [137] in terms of its logical and temporal performances. The inability of the system to meet temporal constraints causes failures and often leads to catastrophic consequences on the controlled environment. A single time fault can then have an intolerable cost in terms of human lives, material damage or economic losses. Hard real-time systems are mainly present in the fields of aeronautics, aerospace, robotics, supervision of chemical or nuclear power plants, etc.

On the boundary between the two previous constraints levels, we find the *Firm real-time system* in which the real-time processing is based on strict constraints, but a low probability of missing the temporal limits can be tolerated [42]. The measure of the respect of temporal constraints can then take the form of a probabilistic data: the *Quality of Service* (QoS). This is directly related to a service offered by the system and/or to the behaviour of the system as a whole. Deadline misses may degrade the system's quality of service. These new types of applications displaying firm real-time systems are for the most part emerging applications in the fields of multimedia, automatic control, and surveillance.

5.4.3 Real-time task characterization and modelling

The functionality of a real-time system is yielded by the software controlling the system's hardware and peripherals. Generally, several jobs are handled by a single embedded system, e.g. a personal monitoring device that allows one to measure one's heart rate in real-time and send the heart rate for later study. In order to ensure the different jobs and improve reuseability and maintainability, the jobs are modularized and each module is called a *task*. In next sections, we define a real-time task and its different properties .

Definition of a real-time task

Nissanke defines a task as "a software entity or program intended to process some specific input or to respond in a specific manner to events conveyed to it" [183].

A task is a software entity that performs a particular function within a software application. It corresponds to the execution of a sequence of operations given on the processor. This sequence of operations relating to the service provided by the task can be repeated several times, periodically for example. Each of the executions is then considered individually in the form of instances (or works).

In a multi-tasking environment, tasks can be in one of four states: *executing*, *ready*, *suspended* and *dormant*. The first three states are considered active states (The tasks exist and provide a special service for the application), the last being an inactive state (the tasks do not exist from the point of view of the application). The transition from one state to another is made by the decision of the scheduler. In practice, a change of state will often result in a change of context. Figure 5.1 illustrates the active states and transitions from one state to another.

A task in the *executing* state has the control of the processor and executes its code. The task at execution is the one that at any given time is considered to have the highest priority among candidates for CPU allocation. A *suspended* task is not a candidate for CPU assignment. Its execution is temporarily suspended

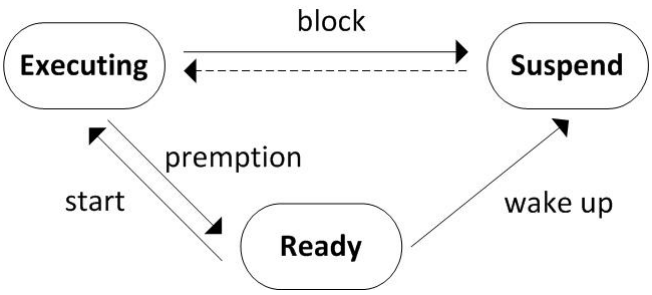


Figure 5.1 – Task active state Diagram.

until it gets the resource that it lacks to run (in addition to the processor). A *ready* task is waiting to be selected to be able to execute. A task in *dormant* state is either not yet created or is definitively terminated.

In addition to the time constraints, other constraints can be associated with the execution of real-time tasks. Among these, we can cite [241]:

— *Resource constraints*: The resources required by a task at its activation are not always assumed to be available whenever this task requires execution, and their access must be protected to ensure consistency (eg.shared variables in memory).

— The *synchronization constraints* which can be described by a set of precedence relations that determine the order in which the tasks must execute. When there is no precedence relationship between tasks, then we are talking about independent tasks.

— The *execution constraints* are based on two modes of execution of the tasks, respectively qualified as *preemptive* and *non-preemptive*. A preemptible task means that its execution can be interrupted at any time and can be retrieved later. Unlike the non-preemptive case in which the task reserves access to the processor from the beginning to the end of its execution.

— The *placement constraints* that relate to the identity of the processor(s) of a multicore system on which a task is allowed to execute.

Based on the way real-time tasks recur over a period of time, a real-time task is generally placed into two main categories: the periodic tasks and the aperiodic tasks.The model of each of these tasks is written below.

In the following, we consider a task τ_i belonging to the system of tasks τ to which is associated the set of jobs J_i executing on the set of processors M.

Model of a periodic real-time task

A periodic task is one that repeats regularly, after a certain fixed time interval which is the period of the task τ_i. A common use of periodic task is to process sensor data and update the current state of the real-time system on a regular basis. Periodic tasks, typically used in control and signal-processing applications have hard deadlines.

From a temporal point of view, a periodic task τ_i is characterized by (r_i, C_i, D_i, P_i) where:

— The release time r_i*:* It represents the wake-up date of the task, that is, the time at which this task is ready for processing.

— *The worst-case execution time* C_i*:* It denotes the maximum uninterrupted/undisturbed execution time taken to complete the task.

— *The deadline* D_i*:* It is the critical time-delay by which execution of the task should be completed, after the task is released.

— *The period* P_i*:* It corresponds to the period of activation, that is to say to the duration which separates two successive arrivals of work for τ_i.

Figure 5.2 shows the model of a periodic real-time task. The time is plotted on the horizontal axis, while the vertical axis is used to indicate whether the task is active (high state) or inactive (low state).

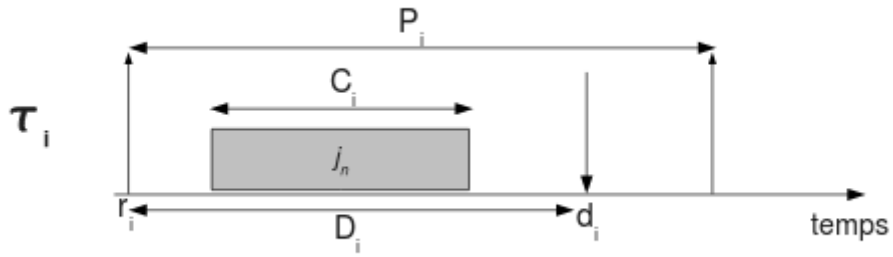


Figure 5.2 – Model of a periodic real-time task

The periodic task τ_i defines an infinite number of jobs all having the same execution time. Every job $j_n \in J$ that wakes up at time $r_i + (n-1) \times P_i$, must end before its deadline $d_i = r_i + (n-1) \times P_i + D_i$; $\forall n \in \mathbb{N}^*$.

In the case where $P_i = D_i$ then τ_i is said task with *deadlines on requests*. Furthermore, if all the periodic tasks of τ have the same initial release time $(\forall i, j \in \mathbb{N}, r_i = r_j)$, then this task configuration is said to be *synchronous*, otherwise it is said *asynchronous*.

Model of an aperiodic real-time task

An aperiodic task requires execution only once. The activation of such a task takes place when an event occurs, which can be either external when it is emitted by the environment or when it comes from another task.

An aperiodic task τ_i is characterized by (r_i, C_i, D_i) where:

— r_i*:* it represents the moment of the first job of τ_i

— C_i*:* It denotes the worst-case execution time.

— D_i*:* It is the critical time-delay or deadline by which execution of the task should be completed.

Figure 5.3 illustrates the model of an aperiodic task. The date of arrival a_i in the system of an aperiodic task is not known a priori. Once taken into account by the system, the aperiodic task begins its execution on a date r_i where $r_i \geq a_i$ and must have completed its execution before $d_i = r_i + D_i$.

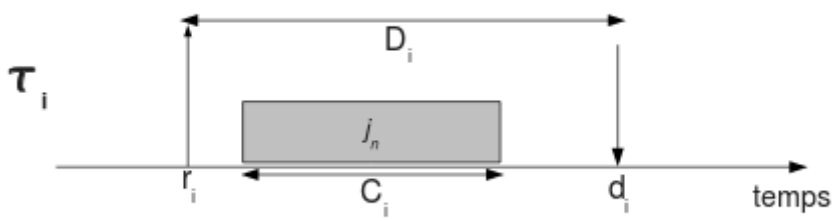


Figure 5.3 – Model of an aperiodic real-time task

Processor utilization factor

An important property which characterize each task is the *Processor utilization factor* U_i, which is the ratio of the worst-case execution time (WCET) and the period ($U_i = C_i/P_i$). The utilisation provides a very simple scheduling check for any processor. Processor utilisation U refers to the sum of the task

utilisations U_i of all tasks scheduled on that processor ($U = \sum U_i$). If the processor utilisation is greater than 1, the tasks cannot all be scheduled successfully on that processor by any scheduler.

5.4.4 Real-time operating system

RTLinux, RTAI and Xenomai are common real-time Linux operating systems developed by open-source projects. We specifically focus on these operating systems because they are free. Our objective is to select one of them for the integration phase of our work. These RTOSes have several benefits. They can be built at a lower cost and provide as good performance as proprietary RTOSes. They are developed using different approaches and techniques. In the next sections, we present a detailed description of these RTOSes.

RTLinux

RTLinux [283] uses a dual-kernel approach which has a transparent, modular and extensible architecture (see figure 5.4). One of them is the Linux kernel, which provides all the features of a general purpose OS, whereas the other one is the RTLinux kernel. RTLinux provides hard real-time capabilities. It has a hybrid kernel architecture with a small real-time kernel that coexists with the Linux kernel which runs at the lowest priority level. This combination allows RTLinux to provide highly optimized, time-shared services in parallel with the real-time, predictable, and low-latency execution[284]

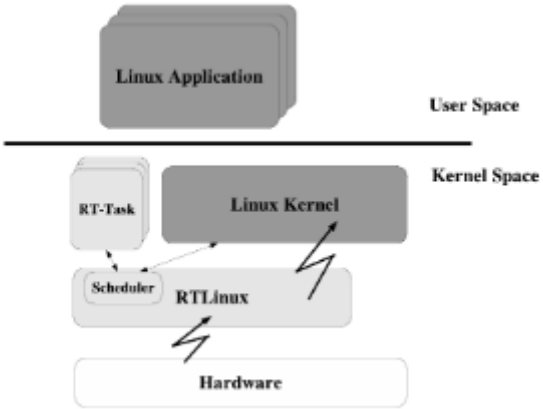


Figure 5.4 – RTLinux architecture

The RTLinux scheduler is pure priority driven. The priority can be fixed by Rate Monotonic algorithm. Nonetheless, a dynamic priority scheduler called Earliest Deadline First can be used. The priority of every job is computed from the current deadline of the job. Higher is the urgency of the job, higher is its priority.

RTAI

RTAI abbreviated from Real Time Application Interface, is a real-time extension for the Linux kernel, bringing it hard real-time features. RTAI was developed by The "Dipartimento di Ingegneria Aerispaziale del Politecnico di Milano" (DIAPM) in 1997. It is originally developed as a variant of RTLinux, at a time when neither floating point support nor periodic mode scheduling were supported by RTLinux. RTAI has now added many new features without compromising performance. One of which is RTHAL [164]. Unfortunately, a patent on the design concept of RTLinux brings the open source RTAI project some potential problems [180]. Thus, the RTAI project has been working to replace RTHAL with ADEOS (Adaptive Domain Environment for Operating Systems) which is a resource virtualization layer available as a Linux kernel patch, to be free of the RTLinux patent.

RTAI takes a unique approach of running Linux as a task (lowest priority) that competes with other real-time tasks for the CPU [66]. RTAI provides deterministic response to POSIX, interrupt, native RTAI real-time task. RTAI consists mainly of two parts:

— The Linux kernel (patch with Adeos-based) which introduces a hardware abstraction layer.

— A broad variety of services which make real-time programmer's lives easier.

The general architecture of RTAI framework is presented in Figure 5.5.

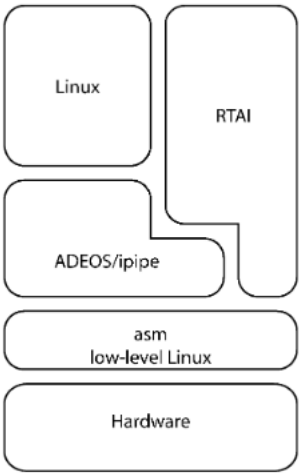


Figure 5.5 – General architecture of RTAI.

RTAI supports several hardware architectures, including IA-32, x86-64, PowerPC, ARM, and MIPS.

Xenomai

The increased requirements of hard real-time capability lead to develop all kinds of real-time operating systems. All these real-time systems are characterized by their own APIs, which makes developers spending more and more time to learn how to use and familiar with the APIs. Also, developers have to rewrite their code when they run their applications in different RTOSes.

The Xenomai project [99] has been launched in August 2001. It has merged in 2003 with the RTAI project, to produce an industrial-grade real-time Free Software platform for GNU/Linux called RTAI/fusion, on top of Xenomai's abstract RTOS core. Eventually, the RTAI/fusion effort became independent from RTAI in 2005 as the Xenomai project. Xenomai aims to design a new framework that supports traditional real-time operating system APIs. This makes that existing industrial applications from different RTOSes can easily run with stringent response time requirements, on embedded Linux platforms. Xenomai offers hard real-time capabilities to the mainline Linux kernel [14].

Xenomai implements an abstract real-time nucleus to support the traditional real-time APIs. It implement the pseudo APIs in different modules, called skins including VxWorks®, pSOS®, VRTX, ulTRON, RTAI and three other skins: POSIX 1003.1b, RTDM and Native skin.

As RTAI, Xenomai uses Adeos as its micro kernel, in order to handle interrupts from the hardware. Adeos is a resource virtualization layer which is available as a Linux kernel patch. In particular, hardware interrupts are intercepted by ADEOS and logically propagated through the pipe structure. This organization is fully achieved in Xenomai as illustrated in Figure 5.6. RTAI has a somewhat different organization, at the point of using ADEOS , as shown in Figure 5.5. Instead of letting ADEOS handle all the interrupt sources, it intercepts them, using ADEOS to propagate those interrupt notifications to Linux in which RTAI is not interested in (i.e., the interrupt does not affect real-time scheduling). In contrast, Xenomai handles all interrupts using ADEOS.

Xenomai can runs on X86, x86-64, PowerPC, ARM, PowerPC64, Blackfin platform Architecture.

In this thesis, we will be interested in Xenomai RTOS for the real-time integration phase of the cryptosystem. Our option for Xenomai is motivated by the following reasons:

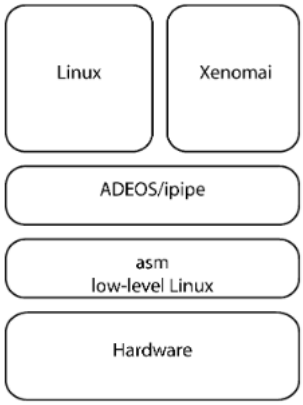


Figure 5.6 – General architecture of Xenomai.

— Xenomai has a strong focus on embedded systems, although it runs over mainline desktop and server architectures as well.
— It has proven its high performance level of compliance with hard real-time constraints.
— Good documentation is available on the website of the project.
— The mailing list of Xenomai project provides good support for developers with prompt and active responses.

5.4.5 Integration of crypto-systems in Xenomai framework

This section deals with the real-time implementation of two proposed stream ciphers using the Xenomai framework. We started by installing Xenomai 2.6 with linux kernel 3.16.0 on our personal computer. An installing documentation was provided by the Xenomai project in [14].

Once the real-time framework was set up and functional, the next step had two main objectives:

- firstly, design of a cryptosystem around real time tasks, and integration with the Xenomai RTOS.
- secondly, performance evaluation of the cryptosystems under the Xenomai RTOS in comparison with the Linux OS.

Real-time cryptosystem design

A stream cipher is a symmetric cryptographical system. It means that the emitter (Alice) and the receiver (Bob) must share the same secret key in order to encrypt/decrypt the message which is transmitted through the communication channel. With this secret key, both emitter and receiver generate the same keystream using a keystream generator, having as input a secret Key "K" and an Initial Vector "IV". XORing this keystream with the plaintext/ciphertext enables us to obtain the ciphertext/plaintext as it is shown in Figure 5.7

In this thesis, we focus on the design of the cryptosystem from the point of view of the emitter. We implement the proposed stream ciphers in order to encrypt the plaintext and transmit the cipher text to the receiver.

First, we have divided our application software into four main tasks:

- τ_K: this task ensures the function of reading the secret Key and the Initial vector IV.
- τ_P: The plain text can be a text file or an image. In this research work, we use images as plaint-text. This task reads the plain-text which is an image. We use for that the opencv library.
- τ_E: the encryption function is ensured by Xoring the generated keystream and the plain text. This task ensures generation of keystream by the PCNG and the Xor function between the plain text and the keystream to create the cipher text.
- τ_T: once the encryption bas been performed, the cipher text can be transmitted to the receiver.

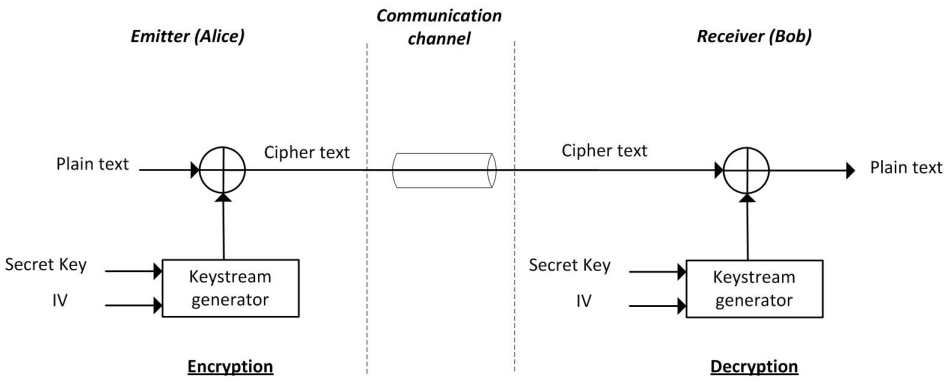


Figure 5.7 – General communication scheme using a stream cipher.

We use the Xenomai API 2.6.5 to implement the real-time stream cipher. Xenomai provides a set of multitasking mechanisms. The creation of different real-time tasks is ensured by calling the function *int rt_task_create (RT_TASK *task, const char *name, int stack_size, int priority, int mode)*. With:

— *task* is a pointer to an RT_TASK type structure which necessarily has been declared and its structure is filled.
— *name* is an ASCII string for the symbolic name of the task.
— *stack_size* is the size of the stack to be used by the new task.
— *priority* is the priority to be assigned to the task. The highest priority is 99, while the lowest one is 1.
— *mode* is a set of flags which affect the task.

Creation of a Real-time task is described by the following code:

```
#include <native/task.h>
#include <native/timer.h>
#include <native/sem.h>
/ * tasks declarations * /
RT_TASK read_key_task;
RT_TASK read_plaintext_task;
RT_TASK encryption_task;
RT_TASK transfer_task;
/ * tasks creation * /
void taskCreate(void){
  int err;
     if (err = rt_task_create(&read_key_task, "read_key_task", 0, 99, 0)){
        rt_printf("Error task create: %s\n", strerror(-err));
        exit(EXIT_FAILURE);
     }
     if (err = rt_task_create(&read_plaintext_task, "read_plaintext_task",
        0, 99, 0)){
        rt_printf("Error task create: %s\n", strerror(-err));
       exit(EXIT_FAILURE);
     }
    if (err = rt_task_create(&encryption_task, "encryption_task", 0,99, 0)){
        rt_printf("Error task create: %s\n", strerror(-err));
        exit(EXIT_FAILURE);
     }
     if (err = rt_task_create(&transfer_task, "transfer_task", 0, 99, 0)){
        rt_printf("Error task transfer create: %s\n", strerror(-err));
        exit(EXIT_FAILURE);
     }
```

```
28  }
```

Listing 5.1 – Description of Tasks creation code

Description of tasks dependencies

The four tasks τ_K, τ_P, τ_E and τ_T have execution dependency on each other. τ_E should begin execution only after the two tasks τ_K and τ_P terminate. Once encryption is complete, τ_T may start to transfer the encrypted plaintext or the ciphertext to the receiver.

We use three binary semaphores namely Sem_{key}, Sem_P and Sem_E. The initial values of the binary semaphores are 0.

Task τ_E has to wait for task τ_K to start execution, at which time, task τ_K signals to task τ_E its completion by unlocking the semaphore and changing the value of the binary semaphore Sem_{key} to 1. In the same way, Task τ_E must also wait for task τ_{IV} to start execution. At the end of its execution, task τ_P unlocks i.e. changes the value of Sem_P to 1 and signals to task τ_E its completion. Once the two semaphores Sem_P and Sem_{key} are equal to 1 (both are unlocked), Task τ_E start execution. Task τ_T cannot begin execution before completion of τ_E. τ_E signals to τ_T the end of its execution by changing the value of Sem_E to 1. Figure 5.8 describes the dependency and the synchronisation of the four tasks τ_K, τ_P, τ_E and τ_T.

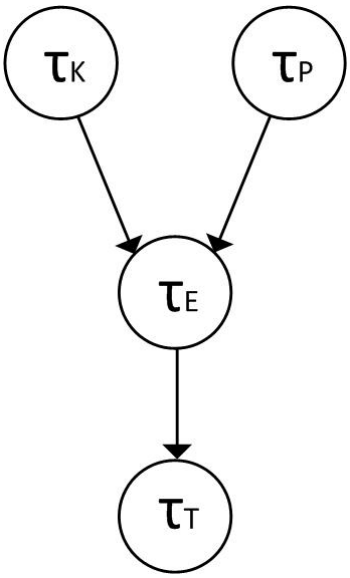


Figure 5.8 – Tasks Dependency Graph.

Mechanisms of Xenomai

Xenomai offers a complete set of classic synchronization mechanisms e.g. semaphores, mutex, variable conditions, event waiting. The description of the Xenomai API is available online or in the Xenomai installation directory.

To manage the dependency of tasks and synchronise their activities, we use binary *binary semaphores*. Binary semaphore can be used for tasks synchronisation. It is initially set equal to 0 (empty), because it acts as en event other tasks are waiting for. Other tasks that need to run in a particular sequence then wait (block) for the binary semaphore to be equal to 1 (until the event occurs) to take the semaphore from the original task and set it back to 0. The semaphores in Xenomai provide fast intertask communication. Semaphores are the primary means for addressing the requirements of task synchronization. In general we can say:

— The creation of semaphores is assured by the function: *int rt_sem_create (RT_SEM* sem, const char* name, unsigned long icount, int mode)* .
— The interface of a semaphore consists of two atomic operations, V and P, which affect an internal counter associated with the semaphore.
— The 'V' ("Verhogen" from Dutch, signal, release, increment) operation increments the counter and returns immediately. This is ensured by the function *int rt_sem_v (RT_SEM* sem)*.
— The 'P' operation ("Proberen", wait, acquire, take) decrements the counter and returns immediately, unless the counter is already zero and the operation blocks until another process releases the semaphore. This is done by the function *int rt_sem_p(RT_SEM* sem, RTIME timeout)*.

The creation and use of semaphores are described in 5.2 and 5.3 respectively.

```
#include <native/task.h>
#include <native/timer.h>
#include <native/sem.h>
#define SEM_INIT 0 / * Initial semaphore count * /
#define SEM_MODE S_FIFO / * Wait by FIFO order * /
/ * Semaphores declarations * /
RT_SEM sem_K;
RT_SEM sem_P;
RT_SEM sem_E;
/ * Semaphores creation * /
void SemaphoreCreate(void){
  if (err = rt_sem_create(&sem_K,"Semaphore_readkey",SEM_INIT,SEM_MODE)) {
     rt_printf("Error semaphore_readkey create: %s\n", strerror(-err));
     exit(EXIT_FAILURE);
  }

  if (err = rt_sem_create(&sem_P,"Semaphore_readplaintext",SEM_INIT,SEM_MODE
     )) {
     rt_printf("Error semaphore_readplaintext create: %s\n", strerror(-err))
        ;
     exit(EXIT_FAILURE);
  }

  if (err = rt_sem_create(&sem_E,"Semaphore_En",SEM_INIT,SEM_MODE)) {
     rt_printf("Error semaphore_en create: %s\n", strerror(-err));
     exit(EXIT_FAILURE);
  }
}
```

Listing 5.2 – Description of Semaphores creation code

```
#include <native/task.h>
#include <native/timer.h>
#include <native/sem.h>
void read_key(void *arg){
/ *** code of the function read_key *** /
rt_sem_v(&sem_K); / * Signal to Encryption task * /
}

void read_plaintext(void *arg){
/ *** code of the function read_plaintext *** /
rt_sem_v(&sem_P); / * Signal to Encryption task * /
}
```

```
13  void transfer_ciphertext(void *arg){
14  rt_sem_p(&sem_K,TM_INFINITE); / * Pend on semaphore sem_K * /
15  rt_sem_p(&sem_P,TM_INFINITE); / * Pend on semaphore sem_P * /
16
17  / * Passing TM_INFINITE causes the caller to block indefinitely until a unit
        is available. * /
18
19  / *** code of the function transfer cipher text *** /
20  }
```

Listing 5.3 – Use of Semaphores by real time tasks code

A task can be started by calling : *int rt_task_start (RT_TASK *task, void(*task_func) (void *arg), void *arg)* where:

— *task* is a pointer to an RT_TASK type structure which must be already initialized by a call to rt_task_create().
— *task_function* is the task function to be executed by this real-time task.
— *arg* is the void pointer argument given to the task function.

```
1  void startTasks() {
2     int err;
3     if (err = rt_task_start(&read_key_task, &read_key, 0)) {
4        rt_printf("Error task start: %s\n", strerror(-err));
5        exit(EXIT_FAILURE);
6     }
7     if (err = rt_task_start(&read_plaintext_task, &read_plaintext, 0)) {
8        rt_printf("Error task start: %s\n", strerror(-err));
9        exit(EXIT_FAILURE);
10    }
11    if (err = rt_task_start(&encryption_task, &encryption, 0)) {
12       rt_printf("Error task start: %s\n", strerror(-err));
13       exit(EXIT_FAILURE);
14    }
15    if (err = rt_task_start(&transfer_task, &transfer, 0)) {
16       rt_printf("Error task start: %s\n", strerror(-err));
17       exit(EXIT_FAILURE);
18    }
19 }
```

Listing 5.4 – Execution start tasks code

5.4.6 Measurements under the RTOS Xenomai

The objective of this section is to measure the time required for the execution of the encryption task under Xenomai. We will present two techniques.

Over the years, many different software- and hardware-based timing measurement methods have been developed [255][254], but there is no single best technique. Rather, each technique is a compromise between multiple attributes including [83]:

- *Resolution*: it is a representation of the limitations of the timing hardware.
- *Accuracy*: it represents the closeness of the measured value using a given method of measuring, as compared to the actual time if a perfect measurement was obtained.
- *Retargetability*: a solution suitable for a particular processor and hardware platform might not be directly applicable on another one.

- *Availability*: there are some type of systems that are not using an OS with suitable timing facilities support.
- *Cost*: Special purpose hardware solutions, such as an emulator, are more often costly than general purpose ones, such as an oscilloscope.
- *Difficulty*: subjectively defines the effort to obtain measurements. A technique that requires usage of hardware equipment such as a logic analyzer or filtering of data to obtain the answers is considered hard. A technique that requires a simple execution of the code and produces an instant measurement is considered easy. Typically, software techniques are easier, but yield only coarse-grain results. Hardware-assisted techniques are hard, but they can provide fine-grain results with high accuracy.

In this thesis, we used two different software techniques to calculate the execution time of the tasks. The first one consists on using the *gettimeofday* system-call which is present in the Linux environment. *gettimeofday* gives answers in micro-second resolution. The second method is using a predefined function present in the Xenomai API, called *int rt_task_inquire* and produces information about current task such us execution time nanoseconds. In next sections, we give a description of these two techniques.

First Method: *gettimeofday* system-call

The POSIX standard C-library available in Linux, provides the system-call *gettimeofday* to access the timing resource and determine the current time.

The *gettimeofday* system call gets the system's wall-clock time. It takes a pointer to a struct timeval variable (as specified in <sys/time.h>):

$$int\ gettimeofday\ (struct\ timeval\ *tv,\ struct\ timezone\ *tz);$$

struct timeval {
time_t tv_sec; /* seconds */
suseconds_t tv_usec; /* microseconds */
};

This structure represents a time, in seconds, split into two fields. The *tv_sec* field contains the integral number of seconds, and the *tv_usec* field contains an additional number of microseconds. This struct *timeval* value represents the number of seconds elapsed since the start of the UNIX epoch, on midnight UTC on January 1, 1970. The *gettimeofday* call also takes a second argument, which should be NULL.

To use the *gettimeofday* approach, the program must be instrumented such that the clock is read at the beginning (*start*)and the end (*end*)of the code segment(s) being measured.

The time spent to execute this code segment(s) is equal to the difference between the *end* and *start* clock time. Instrumenting the code means adding lines of code explicitly to perform the timing measurements. Such lines of code are temporary, and are removed once the desired data has been collected.

Here is an example of a code that uses *gettimeofday* to measure the execution time of the encryption task.

```
#include <sys/time.h>
#include <time.h>

struct timeval start, end; / * absolute times (start/end times) * /
double elapsed;

void encryption(void *arg){
  gettimeofday(&start, NULL);
  / *** encryption process code *** /
  gettimeofday(&end, NULL);
}

```

```
/ * measurement of encryption time in microsecond * /
elapsed=((end.tv_sec - start.tv_sec)*1000000.0)+(end.tv_usec - start.tv_usec)
    ;
rt_printf("Encryption time = %.9lf Microsecond",elapsed);
```

Listing 5.5 – Execution time measurement using *gettimeofday* method.

Second method: *rt_task_inquire* function

Xenomai offers a function that returns various information about the status of a given task such us task name, task status (blocked, ready, delayed, etc.), initial priority, execution time in nanoseconds, time of the next activation, etc. This function is called *rt_task_inquire (RT_TASK * task, RT_TASK_INFO * info)* having as parameters:

— *task*: The descriptor address of the inquired task.

— *info*: The address of a structure the task information will be written to.

In the following example 5.6, we give part of C code in which we use $rt\_task\_inquire$ to calculate the execution time in nanoseconds of a given task.

```
#include <native/task.h>
#include <native/timer.h>
/ * execution time measurements * /
  double execTime (RT\_TASK *curtask){
    RT_TASK_INFO curtaskinfo1;
    double execTimeResult;
    rt_task_inquire(curtask,&curtaskinfo1);
    execTimeResult=(double)curtaskinfo.exectime;
    return execTimeResult;
  }
```

Listing 5.6 – Execution time measurement using *rt_task_inquire* method.

Specifically, we mention that the resulting values of execution time with the first and second methods, present estimations of how long the task takes to execute. It is the difference between the time instant when the task finishes execution and the time instant when the task starts execution. It does not take into consideration preemptions and interrupts. That means that the execution time C_i a task τ_i is calculated as $t_{end} - t_{start} - t_{preempt}$; where t_{start} is the time that $gettimeofday(\&start, NULL)$ returned, t_{end} is the time given by $gettimeofday(\&end, NULL)$ / $(t_{end} - t_{start})$ is the time returned by $rt\_task\_inquire$ and $t_{preempt}$ is computed as the amount of time that another task executed during that time period.

As the *rt_task_inquire* function gives better resolution (nanosecond) than to *gettimeofday* system-call method, we use the *rt_task_inquire* function to calculate the execution time of the encryption task. In the next section, we describe our results for execution time measurements using the second method (rt_task_inquire function).

Measurements for execution times

In the design of any cryptosystem, the computational efficiency of the encryption algorithm is an important factor to exhibit its performance. We calculate the average encryption time in micro second (μs), the encryption throughput in Mega bit par second BR(Mbit/s), and the number of cycles needed to encrypt one byte (NCpB), given as follow:

$$ET = \frac{Image\ size(MByte)}{Encryption\ time(s)} \tag{5.1}$$

$$NCpB = \frac{CPUspeed(Hz)}{ET(Byte/s)} \tag{5.2}$$

Obtaining information about the execution time of a program is by executing the program 100 times with different secret keys and IVs, and then measuring the execution time for each test run. We report in table 5.5 the computation performance measurements of our proposed stream ciphers CM-SC and CS-SC proposed respectively in [114] and [113], implemented in Xenomai RTOS. Also, we restore the computation performance of the algorithms when they are running on ubuntu in table 5.6.

Table 5.5 – Computation performance measurements of stream ciphers implemented in Xenomai RTOS.

Stream cipher	Average encryption time (μs)	ET (Mbits/s)	NCpB
CM-SC	4695,72	319,44	62.10
CS-SC	3214,42	466,64	42.51
Rabbit	1966	762,96	26
HC-128	3327,06	450,842	44

Table 5.6 – Computation performance measurements of stream ciphers implemented in Ubuntu.

Stream cipher	Average encryption time (μs)	ET (Mbits/s)	NCpB
CM-SC	2403.04	624.2	31.78
CS-SC	1059.5	1415.76	14
Rabbit	855.45	1753.46	11.31
HC-128	1330	1127.81	17.59
AES-CTR	-	-	21.2

Results from tables 5.5 and 5.6 show that CS-SC algorithm has better computing performance than CM-SC algorithm as mentioned earlier. In addition, for all the algorithms, the encryption time is greater when the program runs on xenomai. This is due to preemption and interrupts time taken into consideration. Therefore, to better evaluate the time required to perform encryption/decryption, it is necessary to consider non real time OS in which there will be no preemption interferences and interruption issues.

5.5 Conclusion

Standard cryptographic algorithms can be huge, slow or very energy-consuming and consequently not adapted to small electronic devices with severe limitations. Lightweight cryptography concerns the design of new cryptographic algorithms tailored for implementation in constrained environments including RFID tags, sensors, smart cards, health-care devices and also applications of the IoT field. In software implementation, the code and RAM sizes are the important features to evaluate the lightweight properties.

In the first part of this chapter, we have provided a quantitative evaluation of energy, power consumption and memory size requirement of lightweight stream cipher algorithms including CM-SC and CS-SC that were proposed in [114] and [113]. We presented the tools which are well known to measure energy and power consumption such as RAPL and powertop that we used for our experiments. We have choosen these tools among the others since they are open source and free, and do not require external hardware materials.

Also we calculated the requirements of the designed stream ciphers in terms of RAM consumption and code size. Our experiment demonstrates that our chaos-based stream ciphers can be efficiently implemented on energy and time constrained resources devices of the IoT where security is a big concern.

The second part of this chapter concerned the integration of the cryto-systems under the real-time Xenomai software framework. Xenomai has many advantages over other real time operating systems. First, it was developed for use in the Linux kernel as a microkernel and it was primarily considered as a migration tool. Second, its principal characteristic is the presence of skins to adapt the source code from another RTOS to the Linux environment extended with Xenomai. Third, it offers an interesting particularity that is the development of real-time applications in user space and not only as modules in kernel space. For Xenomai, the development in kernel space is reserved for real-time RTDM drivers (Real Time Driver Model). We recommend Xenomai in the design, development and running of real-time application on Linux.

We studied the performance of the crypto-systems by calculating the execution time of the encryption task. Table 5.7 exhibits a comparison between the different measurements of the average encryption time, energy consumption, code size and RAM consumption, obtained by the proposed stream ciphers CM-SC and CS-SC and some algorithms of the literature notably Rabbit and HC-128.

Table 5.7 – Comparative study of the proposed stream ciphers in terms of encryption time, energy consumption, code size and RAM consumption.

Stream cipher	Average encryption time (μs)	PKG energy (J)	Code size (bytes)	RAM consumption (bytes)
CM-SC	4695,72	0.078613	7240	660
CS-SC	3214,42	0.022672	6562	564
Rabbit	1966	0.013855	1714	216
HC-128	3327,06	0.038768	23100	4556

Results show that CS-SC algorithm has better computing performance than CM-SC algorithm. It consumes less energy and has less memory requirements. Also, these performances are comparable or better than some lightweight stream ciphers of the literature.

6

Conclusions and Perspectives

Progress in mobile and wireless technologies, coupled with embedded devices, has led to giving rise to the notion of ubiquitous computing. The vision of Mark Weiser in his article "The Computer of the 21st Century", according to which "the most profound technologies are those that disappear; they weave themselves into the fabric of everyday life until they are indistinguishable from it," is today a reality [271]. We can observe that from automobiles to smart phones, environmental sensors to medical devices and personal communication - embedded computing increasingly pervade our lives. Therefore, computing is becoming revolved around the huge amount of information gotten from a large number of embedded devices that form the Internet of Things (IoT). The core idea of this concept lies in the presence of uniquely identifiable physical objects known as things which can interact with other objects through the Internet. The vast majority of devices that will integrate the IoT are expected to work under severe constrained resource in terms of computing capabilities, memory capacitance, and limited battery and computing power. The communication technique among a large number of constrained devices that generate huge amount of data has an impact on the security and privacy of the applications. These requirements lead to the need of specific security primitives for pervasive devices. Hence, there is an increasing demand for lightweight cryptography, capable of guaranteeing secure data transmission and providing an optimized security/cost/performance trade-off.

In Chapter 2, we presented the fundamental concepts of cryptography primitives. We started by discussing principles of foundation and basic concepts of cryptography and the two major categories of modern cryptographic primitives, namely symmetric and asymmetric algorithms. We described in details block ciphers and stream ciphers. The next step was to introduce chaos theory and briefly present some chaotic maps including Gauss map, Tent map, Hénon map, Lozi map, Lorenz attractor and Rössler attractor. Then, we provided the state of the art of block ciphers, pseudo-random number generators and stream ciphers based on chaotic maps.

In Chapter 3, we studied the security and computing performance of some discrete chaotic maps including: Logistic, Skew Tent and PWLCM maps, as base of proposed chaos-based stream ciphers during this thesis. First, we presented a collection of common and standard security tools useful to define that assessment. Second, we discretized the chaotic maps making them running over a finite precision (N=32), and we analyzed their cryptographic properties and speed. Then, we introduced a perturbation technique which permits to decrease the degradation caused by the discretizing process. We performed some security analysis of chaotic maps using this perturbation technique. In order to improve the cryptographic performance of chaotic maps, we have proposed a recursive structure. Afterwards, we gave the security and speed performance of chaotic maps using the perturbation technique and the recursive structure.

In Chapter 4, we presented our first contribution. It consists of designing and implementing in an efficient and secure manner the three proposed stream ciphers, based on three robust Pseudo-Chaotic Numbers Generators (PCNGs). We described in details the general structure of the three proposed PCNGs. The first proposed PCNG, called CM-PCNG, uses three weakly coupled chaotic maps: PWLCM, Skew Tent and Logistic and includes a multiplexing chaotic technique. In comparison with the architecture of CM-PCNG, the second PCNG - DM-PCNG - uses a binary diffusion matrix on the chaotic coupling technique. The architecture of the third proposed PCNG, named CS-PCNG, is based on using two chaotic maps, namely PWLCM and SkewTent, and includes coupling and swap chaotic techniques. We gave the security and statistical analysis, and the computing performance measures of the proposed PCNGs and stream ciphers. The proposed crypto-systems are very secure, due to the use of chaotic coupling, swap and multiplexing techniques, while having a high speed performance.

In Chapter 5, we focused first on studying the performance of two proposed chaotic stream ciphers CM-SC & CS-SC in terms of energy and power consumption and memory assessment. We showed that the proposed stream ciphers are lightweight crypto-systems. Compared to other crypto-systems presented in

the literature, we demonstrated that our designed stream ciphers are suitable for practical secure applications of the IoT with constrained resources environment. The second part of this chapter concerned the implementation of the proposed crypto-systems as real-time crypto-systems, using a real time operating systems named Xenomai which presents a software framework able to give real time capabilities to the operating system. We reported the used model to implement the real-time crypto-systems and we gave the execution time measurements tools and obtained results of computation performance for the two proposed crypto-systems.

In future work, we plan on continuing the real time performance analysis study. During this thesis, we measured the execution time of individual tasks of the proposed crypto-systems, which presents a necessary step toward fully understanding the timing of a real-time system, but it is not sufficient to analyze its real-time performance. Among the basis measures to be quantified, we quote the *worst-case utilization* of each task. The *worst-case utilization* U_i of a task τ_i is computed as the ratio between the task's worst-case execution time (C_i) and its period (P_i). An important piece of information that also must be considered in the computing performance evaluation of any real-time system is the presence of timing errors, notably the *missed deadline* and *improper scheduling rates*.

In addition, we will propose the realization of a library of crypto-systems based on the proposed algorithms, available to the users in order to implement their real time applications. The users can choose the appropriate crypto-systems according to the requirements of the application in terms of security performance level, and the available resources (energy and available memory) in the device. Also, we plan to prepare a user guide to help developers to implement their real-time crypto-systems under Xenomai RTOS.

Appendices

A

Synthèse des travaux réalisés: Sécurité basée Chaos sous contraintes temps réel et d'énergie pour l'Internet des Objets

A.1 Contexte et Objectives

De nos jours, les périphériques mobiles et embarqués sont devenus omniprésents dans notre vie quotidienne. Cela s'explique par la croissance rapide des technologies de pointe récentes d'informatique et de communication. Ces systèmes embarqués sont utilisés dans nombreuses applications dans divers domaines tels que l'électronique numérique, les télécommunications, les réseaux informatiques, les systèmes satellites, les équipements du système de défense militaire, les équipements du système de recherche, etc. Ces systèmes sont connectés entre eux soit localement, soit via Internet. Ce phénomène est appelé *Internet des objets* (Internet of Things IoT). L'idée centrale de ce concept réside dans la présence d'objets physiques quotidiens liés à Internet. L'interconnexion entre eux est assurée par des technologies telles que l'identification par radiofréquence (RFID), Wireless Sensor Networking (WSN), etc. L'IoT est actuellement en train d'émerger: 50 milliards de périphériques sont estimés être connectés sans fil à Internet d'ici 2020. Le déploiement massif de dispositifs de l'IoT a soulevé la problématique cruciale de la sécurité des données, transmises via des canaux publiques non protégés et utilisées dans nombreuses applications très sensibles à la sécurité (par exemple, défense, militaire, financier, automobiles ou aéronautiques...). De tels dispositifs doivent être invulnérables aux tentatives malveillantes de sabotage de la communication ou qui peuvent limiter leurs fonctionnalités. Ainsi, ces dispositifs doivent inclure des stratégies de protection contre les attaques cryptographiques. Par conséquent, il est nécessaire d'augmenter la sécurité des données transmises afin d'éviter le piratage d'informations et de fraudes, tout en conservant les avantages de l'IoT.

L'utilisation des techniques cryptographiques est appropriée pour fournir de nombreux services de sécurité, tel que la protection de la transmission des données contre les attaques cryptographiques passives et actives. Un nombre croissant de techniques cryptographiques efficaces pour sécuriser l'information transmise ont été développées dans la littérature. Ces techniques cryptographiques existantes développées pour l'informatique d'entreprise pourraient ne pas satisfaire les exigences des applications des systèmes embarqués car elles peuvent être lentes et très consommatrices d'énergie. En effet, les dispositifs de l'IoT sont intrinsèquement contraints aux ressources en matière de mémoire et d'énergie. Par conséquent, il est nécessaire de conçevoir des crypto-systèmes "légères" et efficaces pour garantir une transmission de données sécurisée dans l'IoT.

Au cours des dernières années, la cryptographie basée chaos a reçu beaucoup d'attention suite à son efficacité dans la protection des données. En effet, les propriétés des systèmes chaotiques et déterministes telles que: ergodicité, sensibilité aux conditions initiales et paramètres de contrôle, nombre important de trajectoires très longues (apériodiques), etc., sont très recherchées pour tout système cryptographique dédié à la sécurité des données.

Dans ce contexte, nous avons étudié, dans cette thèse, la problématique de la sécurité de l'information basée chaos sous contraintes temps réel et d'énergie. A ce sujet d'abord, nous avons étudié les performances de trois cartes chaotiques (Logistique, SkewTent et PWLCM) seules et intégrées dans des cellules récursives de 1 à 3 retards. Basé sur les cartes précédentes, nous avons conçu, implémenté de façon efficace et analysé trois générateurs de nombres pseudo-chaotiques (PCNGs). Ces générateurs utilisent une matrice de couplage faible ou une matrice de couplage binaire à forte diffusion entre les cartes chaotiques de base, et une technique de multiplexage chaotique. Puis, nous avons réalisé de façon sécurisée trois systèmes de chiffrement/déchiffrement par flux basés sur les PCNGs proposés. L'analyse cryptographique des systèmes chaotiques réalisés montrent leur robustesse contre des attaques connues. Ce résultat est dû aux structures proposées qui intègrent une forte non-linéarité, une technique de couplage faible, ou de couplage binaire à fort diffusion, un multiplexage chaotique et une technique de permutation pour le troisième système. La performance obtenue en complexité de calcul indique leurs utilisations dans des applications temps réel.

Ensuite, nous avons intégré ces systèmes de chiffrement/déchiffrement chaotiques au sein d'un système d'exploitation temps réel appelé Xenomai. Enfin, nous avons mesuré la consommation d'énergie et de puissance des trois systèmes chaotiques réalisés, et nous avons montré comment adapter le degré de sécurité de ces systèmes en fonction de la disponibilité énergétique temporelle.

A.2 Contributions:

A.2.1 1 ère contribution: Conception, mise en oeuvre et analyse de générateurs de nombres pseudo-chaotiques et systèmes de chiffrement par flux

La structure générale des PCNG proposés est présentée dans la Fig. A.1. Elle prend comme entrée, les paramètres du système (N et le nombre d'échantillons Ns), une clé secrète "K" et un vecteur initial "IV" de taille 32 bits, et comme sortie, elle génère une séquence d'échantillons pseudo-chaotiques $X(n)$, n = 1, 2, ..., chacun quantifié sur N = 32 bits. Les PCNG proposés se composent de quatre fonctions principales:

— IV-Setup: bloc de configuration IV;
— Key-setup: bloc de configuration de clés secrétes;
— Internal function: bloc de fonction interne;
— Output function: configuration de sortie.

Tous les PCNG proposés ont la même structure générale mais diffèrent dans leur état interne (Internal function) et légèrement leur fonction de configuration de clé, de configuration IV et de sortie (Key-setup, IV-setup and Output function). Chaque bloc fonctionnel sera détaillé dans la description architecturale des PCNG proposés. Dans ce qui suit, nous décrivons en détail la structure générale des PCNG proposés et leurs architectures. Ensuite, nous étudions la sécurité et la performance en termes de vitesse de ces PCNGs.

Architecture du CM-PCNG proposé

L'architecture du premier générateur chaotique proposé CM-PCNG est donnée dans la Fig. A.2. Il utilise trois cartes chaotiques faiblement couplées: PWLCM, Skew Tent et Logistique; et comprend une technique de multiplexage chaotique.

La fonction Key-setup se compose de deux parties principales. Elle prend la clé secrète K et le vecteur initial IV comme entrée et calcule les valeurs initiales $Xp(0)$, $Xs(0)$ et $Xl(0)$ des trois cartes chaotiques: PWLCM, Skewtent et Logistique respectivement.

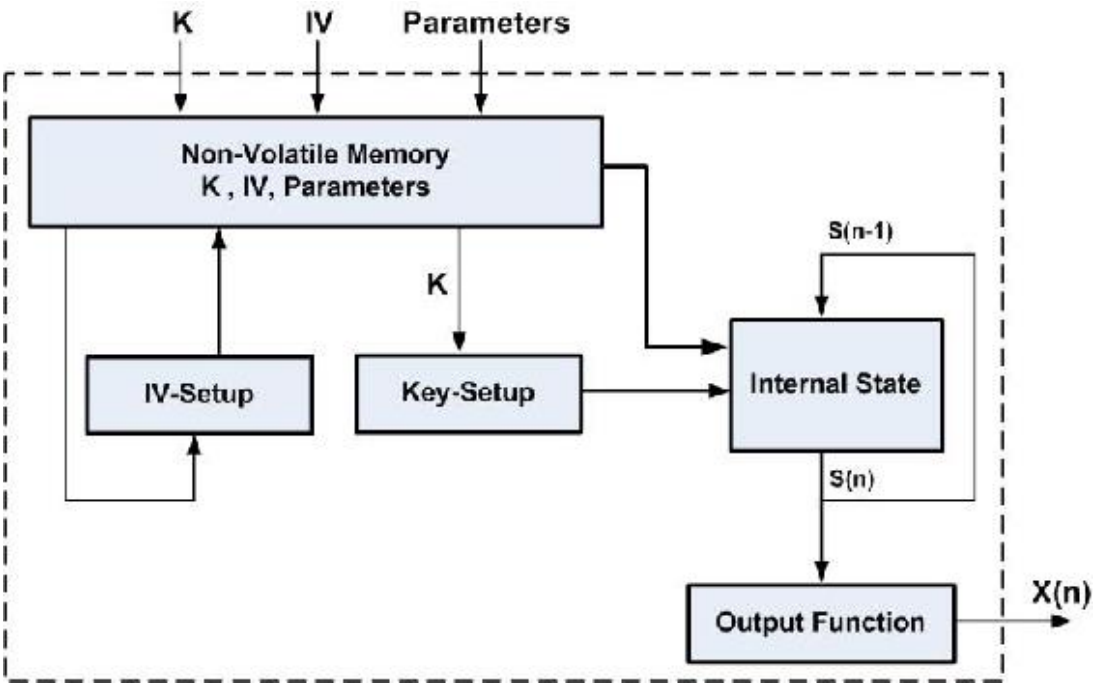


Figure A.1 – Structure générale des PCNGs proposés.

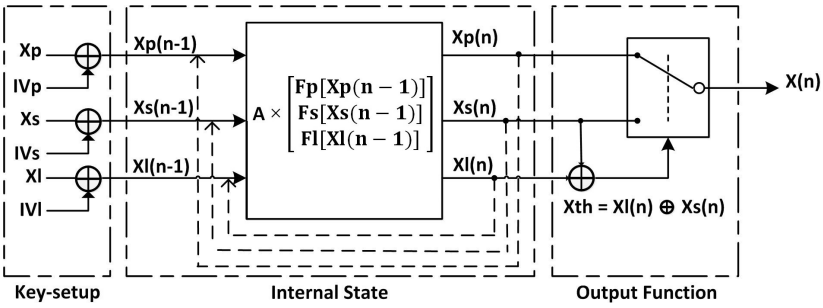


Figure A.2 – architecture du générateur proposé CM-PCNG.

La clé secrète du système est formée par:

— les conditions initiales Xp, Xs et Xl des trois cartes chaotiques: PWLCM, Skewtent et Logistique respectivement, allant de 1 à 2^N-1,

— le paramètres de contrôles Pp et Ps des cartes PWLCM et Skewtent, appartenant aux intervalles $[1, 2^{N-1} - 1]$ et $[1, 2^N - 1]$ respectivement,

— les parametres de la matrice de couplage **A**, ε_{ij}, allant de 1 à 2^k avec k$\leq$ 5.

Les valeurs initiales $Xp(0)$, $Xs(0)$ et $Xl(0)$ sont calculées comme suit:

$$\begin{cases} Xp(0) = Xp \oplus IVp \\ Xs(0) = Xs \oplus IVs \\ Xl(0) = Xl \oplus IVl \end{cases} \tag{A.1}$$

où

$$\begin{cases} IVp = lsb(IV) \\ IVs = L_{cir}[lsb(IV), 3] \\ IVl = L_{cir}[lsb(IV), 2] \end{cases} \tag{A.2}$$

Avec $\oplus$ désigne l'opérateur XOR, $lsb(IV)$ sont les 32 bits les moins significatifs de IV et $L_{cir}[S, q]$ effectue un déplacement circulaire gauche des q-bits sur la séquence binaire S.

La fonction d'état interne (internal state) effectue le couplage faible des cartes chaotiques et produit les futurs échantillons $Xp(n)$, $Xs(n)$ et $Xl(n)$,qui sont utulisés par la fonction de sortie en appliquant une technique de commutation chaotique, pour produire la séquence de sortie $X(n)$ (voir Fig. A.2).

Le système est gouverné par l'équation suivante:

$$\begin{bmatrix} Xp(n) \\ Xs(n) \\ Xl(n) \end{bmatrix} = \mathbf{A} \times \begin{bmatrix} Fp[Xp(n-1)] \\ Fs[Xs(n-1)] \\ Fl[Xl(n-1)] \end{bmatrix}. \tag{A.3}$$

Où **A** présente la matrice de couplage faible:

$$\mathbf{A} = \begin{bmatrix} (2^N - \varepsilon_{12} - \varepsilon_{13}) & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & (2^N - \varepsilon_{21} - \varepsilon_{23}) & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & (2^N - \varepsilon_{31} - \varepsilon_{32}) \end{bmatrix}. \tag{A.4}$$

Avec ε_{ij} sont les paramètres de couplage faible, et $Fp[Xp(n-1)]$, $Fs[Xs(n-1)]$ et $Fl[Xl(n-1)]$ sont les fonctions discrètes des cartes chaotiques PWLCM, Skew Tent et Logistic respectivement.

Les échantillons multiplexés obtenus de la séquence $X(n)$ sont contrôlés par l'échantillon chaotique $Xth(n)$ et un seuil T, comme c'est illustré dans Fig. A.2, et sont définis comme suit:

$$X(n) = \begin{cases} Xp(n), & \text{if } 0 < Xth(n) < T \\ Xs(n), & \text{otherwise} \end{cases} \tag{A.5}$$

avec $Xth(n) = Xl(n) \oplus Xs(n)$.

Architecture du DM-PCNG proposé

L'architecture du deuxième générateur proposé appelé DM-PCNG est présentée dans la Fig. A.3. Par rapport à l'architecture précédente, la principale différence réside dans la fonction d'état interne, qui repose sur une matrice de diffusion binaire **D**.

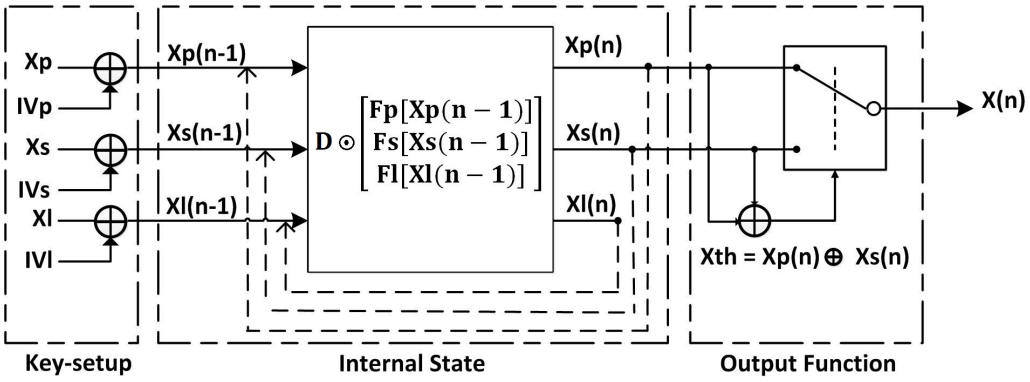


Figure A.3 – Architecture du DM-PCNG proposé.

L'équation du système est donnée par:

$$\begin{bmatrix} Xp(n) \\ Xs(n) \\ Xl(n) \end{bmatrix} = \mathbf{D} \odot \begin{bmatrix} Fp[Xp(n-1)] \\ Fs[Xs(n-1)] \\ Fl[Xl(n-1)] \end{bmatrix}. \tag{A.6}$$

Où D est la matrice de diffusion binaire:

$$\mathbf{D} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}. \tag{A.7}$$

Et $\odot$ est l'opérateur défini comme suit:

$$\begin{bmatrix} Xp(n) \\ Xs(n) \\ Xl(n) \end{bmatrix} = \begin{bmatrix} Fp[Xp(n-1)] \oplus Fs[Xs(n-1)] \\ Fs[Xs(n-1)] \oplus Fl[Xl(n-1)] \\ Fp[Xp(n-1)] \oplus Fl[Xl(n-1)] \end{bmatrix}. \tag{A.8}$$

Le choix des échantillons $X(n)$ est contrôlé, comme dans l'équation Eq.(A.5) par un seuil T et l'échantillon chaotique Xth, avec $Xth(n) = Xp(n) \oplus Xs(n)$.

Architecture du CS-PCNG proposé

L'architecture de CS-PCNG est présentée dans la Fig.A.4. Par rapport à l'architecture de CM-PCNG, l'architecture de CS-PCNG se distingue non seulement par la fonction d'état interne, mais aussi par la fonction de sortie. L'état interne utilise deux cartes chaotiques (PWLCM et SkewTent), et inclut des techniques de couplage et de permutation. La fonction de sortie est une opération XOR entre $Xp(n)$ et $Xs(n)$ samples.

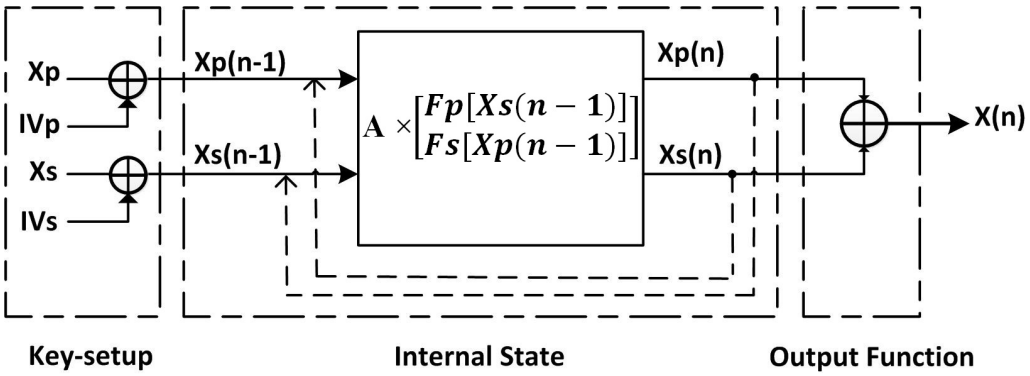


Figure A.4 – Architecture du CS-PCNG proposé.

La technique de couplage est basée sur l'utilisation de la matrice **A** lors du calcul des échantillons $Xp(n)$ et $Xs(n)$. La technique de permutation consiste à utiliser $Xp(n-1)$ comme une entrée de la fonction discrète de la carte SkewTent Fs et $Xs(n-1)$ comme entrée de la fonction discrète de la carte PWLCM Fp.

L'équation du systeme est donnée par:

$$\begin{bmatrix} Xp(n) \\ Xs(n) \end{bmatrix} = \mathbf{A} \times \begin{bmatrix} Fp[Xs(n-1)] \\ Fs[Xp(n-1)] \end{bmatrix}. \tag{A.9}$$

Avec:

$$\mathbf{A} = \begin{bmatrix} (2^N - \varepsilon_{11}) & \varepsilon_{12} \\ \varepsilon_{21} & (2^N - \varepsilon_{22}) \end{bmatrix} \tag{A.10}$$

Les échantillons de sortie $X(n)$ sont calculés à partir des échantillons $Xp(n)$ et $Xs(n)$ comme suit:

$$X(n) = Xp(n) \oplus Xs(n). \tag{A.11}$$

Réalisation de système de chiffrement par flux basé sur les générateurs proposés et étude de leurs performances cryptographiques

Nous avons réalisé et analysé trois systèmes de chiffrement/déchiffrement par flux nommé CM-SC, DM-SC et CS-SC basés sur les générateurs proposés CM-PCNG, DM-PCNG et CS-PCNG respectivement(voir Fig. A.5). En effet, la sécurité de tout système de chiffrement par flux dépend du caractère aléatoire de la clé (keystream) générée par le PCNG, donc de la robustesse du PCNG utilisé qui est l'élément principal de tout système de chiffrement par flux.

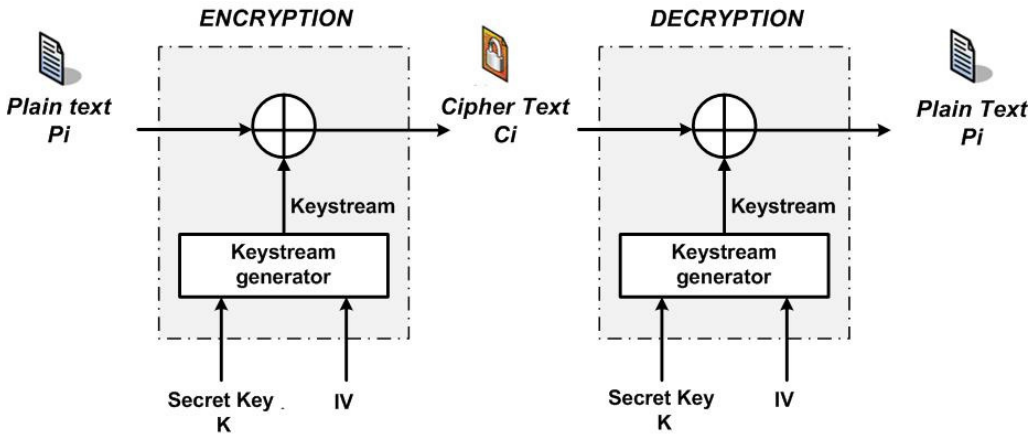


Figure A.5 – Système de chiffrement/déchiffrement par flux.

Nous avons analysé la sécurité des systèmes réalisés. Les résultats obtenus lors des différents tests expérimentaux appliqués montrent leur robustesse contre les attaques cryptographiques et statistiques connues, et peuvent être utilisés dans des applications pratiques. Ci-dessous, nous donnons quelques résultats obtenus pour le système de chiffrement par flux CM-SC proposé. L'histogramme de l'image chiffrée est uniforme et significativement différent de celui de l'image d'origine (voir Fig A.6). Ceci est prouvé par le test Chi2. Nous utilisons également l'un des tests standard les plus populaires pour enquêter sur le caractère aléatoire des données binaires, appelé NIST test. Il est composé de 188 tests statistiques et sous-tests groupés sur 15 tests. Nous montrons dans la Fig. A.7, un exemple de résultat obtenu par le test de NIST sur 100 séquences produites, chacune contenant 1 million de bits. Toutes les séquences passent le test de NIST.

Aussi, nous avons étudié les performances des systèmes de chiffrement par flux en termes de vitesses de calcul. Nous donnons dans le tableau A.1 le nombre de cycles nécessaires pour chiffrer un octet (NCpB), pour les 3 systèmes de chiffrements par flux proposés et ceux de Rabbit, HC-128 et AES-CTR. Ces résultats montrent que le système CS-SC est le plus rapide comparé aux autres systèmes proposés sauf celui de Rabbit.

Table A.1 – Perfermance en termes de vitesse de chiffrement des systèmes proposés.

Système de chiffrement par flux	NCpB
CM-SC	31.78
DM-SC	24.03
CS-SC	14
Rabbit	11.31
Hc-128	17.59
AES-CTR	21.2

A.2.2 2 ème contribution: Évaluation de la consommation d'énergie et mise en œuvre en temps réel des systèmes de chiffrement par flux proposés

Nous avons étudié la consommation d'énergie et les exigences en termes de mémoire RAM et la taille de code des deux systèmes de chiffrement proposés CM-SC et CS-SC. Pour effectuer ces mesures, nous avons

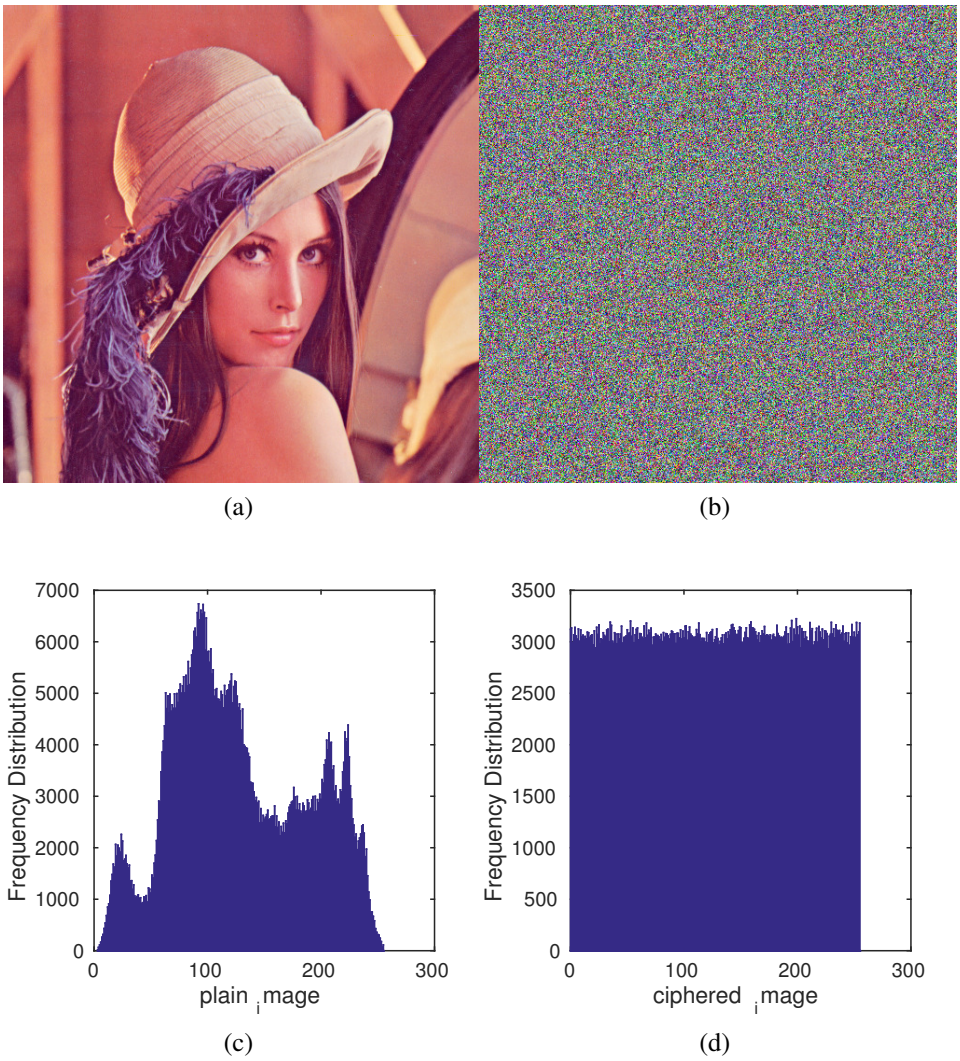


Figure A.6 – (a) Lena Image, (b) Image chiffrée de Lena , (c) tHistogramme de Lena Image et (d) histogramme de l'image chiffrée de Lena.

utilisé les outils logiciels appelés RAPL (Running Average Power Limit) pour mesurer la consommation d'énergie, et FELICS pour calculer la consommation de RAM et la taille de code. Les résultats obtenus, donnés dans le tableau A.2, indiquent que les systèmes de chiffrement par flux basés chaos peuvent être mis en oeuvre efficacement sur les dispositifs de l'IoT qui sont caractérisés par des fortes contraintes de consommation d'énergie et aussi de mémoire. Aussi, CS-SC a moins de consommation d'énergie par rapport aux algorithmes CM-SC et HC-128. En effet, CS-SC consomme environ 30% de l'énergie consommée par l'algorithme CM-SC et 60% de celle de l'algorithme HC-128. Aussi, CS-SC consomme environ deux fois plus d'énergie que Rabbit. Cependant, CM-SC est le plus efficace en termes de sécurité.

Nous avons intégré les deux crypto-systèmes CM-SC et CS-SC dans un système temps réel. Pour cela, nous avons utilisé le système d'exploitation temps réel Xenomai. Nous avons évalué les performances des crypto-systèmes sous RTOS Xenomai par rapport au système d'exploitation Linux. Les résultats obtenus prouvent que CS-SC a de meilleures performances en termes de vitesse que CM-SC comme mentionné précédemment. En outre, pour tous les algorithmes, le temps de chiffrement est plus grand lorsque le programme fonctionne sur Xenomai. Cela est dû à la préemption et le temps d'interruption pris en compte. Par conséquent, pour mieux évaluer le temps requis pour effectuer le chiffrement / déchiffrement, il faut tenir compte du système d'exploitation non temps réel dans lequel il n'y aura pas d'interférences de préemption et d'interruption.

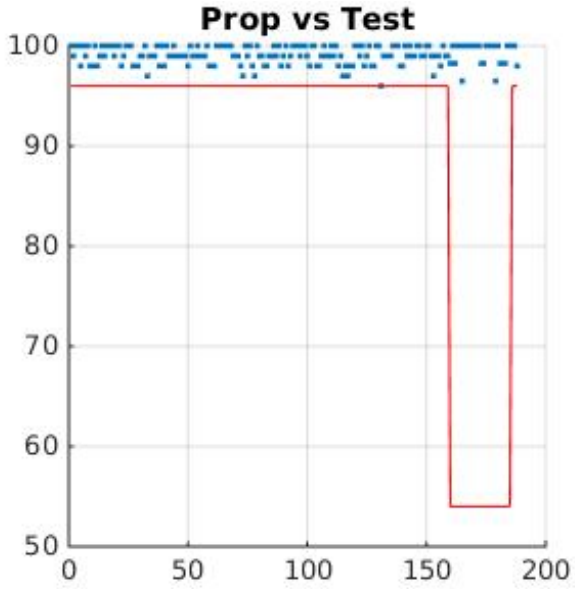


Figure A.7 – Résultat du test NIST du système CM-SC.

Table A.2 – Etude comparative des systèmes de chiffrement par flux proposés en termes de consommation d'énergie, taille de code et de mémoire RAM.

Systèmes de chiffrement par flux	énergie consommée (J)	taille de code (bytes)	mémoire RAM (bytes)
CM-SC	0.078613	7240	660
CS-SC	0.022672	6562	564
Rabbit	0.013855	1714	216
HC-128	0.038768	23100	4556

Mots clés: Internet des Objets, Sécurité des Données, Générateurs de nombres pseudo-chaotiques, Chiffrement par flux basé chaos, Analyse de la sécurité, Vitesse de chiffrement, Consommation d'énergie, Temps réel.

B

List of perturbation polynomials

List of perturbation polynomials for the Logistic map

P - PW - G1 : g1(x) =$x^{16} + x^{12} + x^3 + x + 1$, or [16, 12, 3, 1, 0])
P - PW - G3 : g3(x) =$x^{16} + x^{12} + x^7 + x^2 + 1$
P - PW - G5 : g5(x) =$x^{16} + x^9 + x^5 + x^2 + 1$
P - PW - G7 : g7(x) = $x^{16} + x^{15} + x^9 + x^4 + 1$
P - PW - G9 : g9(x) =$x^{16} + x^{12} + x^9 + x^6 + 1$
P - PW - G11 : g11(x) =$x^{16} + x^{10} + x^7 + x^6 + 1$
P - PW - G13 : g13(x) =$x^{16} + x^9 + x^4 + x^3 + 1$
P - PW - G2 : g2(x) = $x^{17} + x^3 + 1$
P - PW - G4 : g4(x) =$x^{17} + x^{16} + x^3 + x + 1$
P - PW - G6 : g6(x) =$x^{17} + x^8 + x^7 + x^6 + x^4 + x^3 + 1$
P - PW - G8 : g8(x) =$x^{17} + x^9 + x^8 + x^6 + x^4 + x + 1$
P - PW - G10 : g10(x) = $x^{17} + x^7 + x^4 + x^3 + 1$
P - PW - G12 : g12(x) = $x^{17} + x^{12} + x^6 + x^3 + x^2 + x + 1$
P - PW - G14 : g14(x) = $x^{17} + x^{11} + x^8 + x^6 + x^4 + x^2 + 1$
P - PW - G15 : g15(x) = $x^{19} + x^5 + x^2 + x + 1$
P - PW - G17 : g17(x) =$x^{19} + x^{12} + x^{10} + x^9 + x^7 + x^3 + 1$
P - PW - G19 : g19(x) =$x^{19} + x^{13} + x^8 + x^5 + x^4 + x^3 + 1$
P - PW - G21 : g21(x) = $x^{19} + x^{18} + x^{17} + x^{16} + x^{12} + x^7 + x^6 + x^5 + x^3 + x + 1$
P - PW - G23 : g23(x) = $x^{19} + x^9 + x^8 + x^7 + x^6 + x^3 + 1$
P - PW - G25 : g25(x) =$x^{19} + x^{16} + x^{15} + x^{13} + x^{12} + x^9 + x^5 + x^4 + x^2 + x + 1$
P - PW - G27 : g27(x) =$x^{19} + x^{18} + x^{15} + x^{14} + x^{11} + x^{10} + x^8 + x^5 + x^3 + x^2 + 1$
P - PW - G16 : g16(x) =$x^{23} + x^5 + 1$
P - PW - G18 : g18(x) =$x^{23} + x^{12} + x^5 + x^4 + 1$
P - PW - G20 : g20(x) =$x^{23} + x^{11} + x^{10} + x^7 + x^6 + x^5 + 1$
P - PW - G22 : g22(x) =$x^{23} + x^{17} + x^{11} + x^5 + 1$
P - PW - G24 : g24(x) =$x^{23} + x^{21} + x^7 + x^5 + 1$
P - PW - G26 : g26(x) =$x^{23} + x^5 + x^4 + x + 1$
P - PW - G28 : g28(x) = $x^{23} + x^{16} + x^{13} + x^6 + x^5 + x^3 + 1$

List of perturbation polynomials for the Skew Tent map

P - SK - G1 : g1(x) = $x^{15} + x^{13} + x^{10} + x + 1$, or [15, 13, 10, 1, 0])
P - SK - G3 : g3(x) = $x^{19} + x^5 + x^2 + x + 1$
P - SK - G5 : g5(x) = $x^{21} + x^2 + 1$
P - SK - G7 : g7(x) =$x^{15} + x^9 + x^4 + x + 1$
P - SK - G9 : g9(x) = $x^{21} + x^{14} + x^7 + x^2 + 1$
P - SK - G11 : g11(x) = $x^{17} + x^{16} + x^3 + x + 1$
P - SK - G13 : g13(x) =$x^{15} + x^{14} + x^{12} + x^2 + 1$
P - SK - G2 : g2(x) = $x^{17} + x^3 + x^2 + x + 1$
P - SK - G4 : g4(x) = $x^{23} + x^{12} + x^5 + x^4 + 1$
P - SK - G6 : g6(x) = $x^{17} + x^7 + x^4 + x^3 + 1$
P - SK - G8 : g8(x) =$x^{19} + x^9 + x^8 + x^7 + x^6 + x^3 + 1$
P - SK - G10 : g10(x) =$x^{15} + x^7 + x^4 + x + 1$
P - SK - G12 : g12(x) = $x^{21} + x^{13} + x^5 + x^2 + 1$
P - SK - G14 : g14(x) =$x^{15} + x^{13} + x^{10} + x^9 + 1$
P - SK - G15 : g15(x) =$x^{23} + x^5 + x^4 + x + 1$
P - SK - G17 : g17(x) =$x^{15} + x^{13} + x^9 + x^6 + 1$
P - SK - G19 : g19(x) =$x^{21} + x^{10} + x^6 + x^4 + x^3 + x^2 + 1$
P - SK - G21 : g21(x) =$x^{15} + x^{14} + x^9 + x^2 + 1$
P - SK - G23 : g23(x) =$x^{15} + x^{13} + x^{12} + x^{10} + 1$
P - SK - G25 : g25(x) = $x^{15} + x^{12} + x^3 + x + 1$
P - SK - G27 : g27(x) =$x^{19} + x^{13} + x^8 + x^5 + x^4 + x^3 + 1$
P - SK - G16 : g16(x) = $x^{17} + x^{12} + x^6 + x^3 + x^2 + x + 1$
P - SK - G18 : g18(x) = $x^{21} + x^{14} + x^7 + x^6 + x^3 + x^2 + 1$
P - SK - G20 : g20(x) =$x^{23} + x^{17} + x^{11} + x^5 + 1$
P - SK - G22 : g22(x) =$x^{17} + x^9 + x^8 + x^6 + x^4 + x + 1$
P - SK - G24 : g24(x) =$x^{21} + x^8 + x^7 + x^4 + x^3 + x^2 + 1$
P - SK - G26 : g26(x) =$x^{17} + x^8 + x^7 + x^6 + x^4 + x^3 + 1$
P - SK - G28 : g28(x) =$x^{15} + x^{13} + x^7 + x^4 + 1$

List of perturbation polynomials for the PWLCM map

P - PW - G1 : g1(x) =$x^{16} + x^{12} + x^3 + x + 1$, or [16, 12, 3, 1, 0])
P - PW - G3 : g3(x) =$x^{16} + x^{12} + x^7 + x^2 + 1$
P - PW - G5 : g5(x) =$x^{16} + x^9 + x^5 + x^2 + 1$
P - PW - G7 : g7(x) = $x^{16} + x^{15} + x^9 + x^4 + 1$
P - PW - G9 : g9(x) =$x^{16} + x^{12} + x^9 + x^6 + 1$
P - PW - G11 : g11(x) =$x^{16} + x^{10} + x^7 + x^6 + 1$
P - PW - G13 : g13(x) =$x^{16} + x^9 + x^4 + x^3 + 1$
P - PW - G2 : g2(x) = $x^{17} + x^3 + 1$
P - PW - G4 : g4(x) =$x^{17} + x^{16} + x^3 + x + 1$
P - PW - G6 : g6(x) =$x^{17} + x^8 + x^7 + x^6 + x^4 + x^3 + 1$
P - PW - G8 : g8(x) =$x^{17} + x^9 + x^8 + x^6 + x^4 + x + 1$
P - PW - G10 : g10(x) = $x^{17} + x^7 + x^4 + x^3 + 1$
P - PW - G12 : g12(x) = $x^{17} + x^{12} + x^6 + x^3 + x^2 + x + 1$
P - PW - G14 : g14(x) = $x^{17} + x^{11} + x^8 + x^6 + x^4 + x^2 + 1$
P - PW - G15 : g15(x) = $x^{19} + x^5 + x^2 + x + 1$
P - PW - G17 : g17(x) =$x^{19} + x^{12} + x^{10} + x^9 + x^7 + x^3 + 1$

P - PW - G19 : g19(x) =$x^{19} + x^{13} + x^{8} + x^{5} + x^{4} + x^{3} + 1$
P - PW - G21 : g21(x) = $x^{19} + x^{18} + x^{17} + x^{16} + x^{12} + x^{7} + x^{6} + x^{5} + x^{3} + x + 1$
P - PW - G23 : g23(x) = $x^{19} + x^{9} + x^{8} + x^{7} + x^{6} + x^{3} + 1$
P - PW - G25 : g25(x) =$x^{19} + x^{16} + x^{15} + x^{13} + x^{12} + x^{9} + x^{5} + x^{4} + x^{2} + x + 1$
P - PW - G27 : g27(x) =$x^{19} + x^{18} + x^{15} + x^{14} + x^{11} + x^{10} + x^{8} + x^{5} + x^{3} + x^{2} + 1$
P - PW - G16 : g16(x) =$x^{23} + x^{5} + 1$
P - PW - G18 : g18(x) =$x^{23} + x^{12} + x^{5} + x^{4} + 1$
P - PW - G20 : g20(x) =$x^{23} + x^{11} + x^{10} + x^{7} + x^{6} + x^{5} + 1$
P - PW - G22 : g22(x) =$x^{23} + x^{17} + x^{11} + x^{5} + 1$
P - PW - G24 : g24(x) =$x^{23} + x^{21} + x^{7} + x^{5} + 1$
P - PW - G26 : g26(x) =$x^{23} + x^{5} + x^{4} + x + 1$
P - PW - G28 : g28(x) = $x^{23} + x^{16} + x^{13} + x^{6} + x^{5} + x^{3} + 1$

Bibliography

[1] Clang Static Analyzer. 94

[2] DRD: a thread error detector. 94

[3] The estream portfolio. 27

[4] The estream project-hc128. 29

[5] GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF). 94

[6] GDB: The GNU Project Debugger). 94

[7] LeakTracer - trace and analyze memory leaks in C++ programs). 94

[8] memset(3) - Linux manual page. 95

[9] mlock(2): lock/unlock memory - Linux man page. 95

[10] munlock(2): lock/unlock memory - Linux man page. 95

[11] Rössler attractor - wikipedia. 36

[12] The tent map - wikipedia. 9, 33

[13] Valgrind Home. 94

[14] Xenomai:real-time framework for linux. 16, 136, 137

[15] Powertop | 01.org, 2016. 128

[16] T. Addabbo, M. Alioto, A. Fort, S. Rocchi, and V. Vignoli. Long period pseudo random bit generators derived from a discretized chaotic map. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 892–895. IEEE, 2005. 41

[17] H. E.-d. H. Ahmed, H. M. Kalash, and O. S. F. Allah. An efficient chaos-based feedback stream cipher (ecbfsc) for image encryption and decryption. *Informatica*, 31(1), 2007. 42, 121

[18] A. Akhavan, A. Samsudin, and A. Akhshani. A symmetric image encryption scheme based on combination of nonlinear chaotic maps. *Journal of the Franklin Institute*, 348(8):1797–1813, 2011. 31

[19] A. Akhshani, A. Akhavan, A. Mobaraki, S.-C. Lim, and Z. Hassan. Pseudo random number generator based on quantum chaotic map. *Communications in Nonlinear Science and Numerical Simulation*, 19(1):101–111, 2014. 107, 111

[20] C. Alippi, A. Bogdanov, and F. Regazzoni. Lightweight cryptography for constrained devices. In *Integrated Circuits (ISIC), 2014 14th International Symposium on*, pages 144–147. IEEE, 2014. 15

[21] K. T. Alligood, T. D. Sauer, and J. A. Yorke. *Chaos: an introduction to dynamical systems*. Springer Science & Business Media, 2006. 32

[22] E. Alvarez, A. Fernandez, P. Garcıa, J. Jiménez, and A. Marcano. New approach to chaotic encryption. *Physics Letters A*, 263(4):373–375, 1999. 40

[23] G. Alvarez and S. Li. Some basic cryptographic requirements for chaos-based cryptosystems. *International Journal of Bifurcation and Chaos*, 16(08):2129–2151, 2006. 46

[24] G. Alvarez, F. Montoya, M. Romera, and G. Pastor. Cryptanalysis of an ergodic chaotic cipher. *Physics Letters A*, 311(2):172–179, 2003. 38

[25] G. Alvarez, F. Montoya, M. Romera, and G. Pastor. Keystream cryptanalysis of a chaotic cryptographic method. *Computer Physics Communications*, 156(2):205–207, 2004. 38

[26] P. Amato, D. Mascolo, I. Pedaci, and D. Ruggiero. Method of generating successions of pseudorandom bits or numbers, May 3 2006. US Patent App. 11/381,474. 87

[27] S. Ansari, N. Gupta, and S. Agrawal. A review on chaotic map based cryptography. 31

[28] A. Arlicot. Sequences generator based on chaotic maps. Technical report, IETR, Rennes, France, 2014. 120

[29] D. Arroyo, G. Alvarez, J. M. Amigó, and S. Li. Cryptanalysis of a family of self-synchronizing chaotic stream ciphers. *Communications in Nonlinear Science and Numerical Simulation*, 16(2):805–813, 2011. 43

[30] K. Ashton. That 'internet of things' thing. *RFiD Journal*, 22(7), 2011. 14

[31] J.-P. Aumasson. On a bias of rabbit. In *State of the Art of Stream Ciphers Workshop (SASC 2007), eSTREAM, ECRYPT Stream Cipher Project, Report*, 2007. 29

[32] G. L. Baker and J. P. Gollub. *Chaotic dynamics: an introduction*. Cambridge University Press, 1996. 32

[33] D. Bandyopadhyay and J. Sen. Internet of things: Applications and challenges in technology and standardization. *Wireless Personal Communications*, 58(1):49–69, 2011. 14

[34] M. Baptista. Cryptography with chaos. *Physics letters A*, 240(1-2):50–54, 1998. 37

[35] B. Barney. Posix threads programming. *National Laboratory. Disponível em:< https://computing. llnl. gov/tutorials/pthreads/> Acesso em*, 5, 2009. 92

[36] R. W. Batterman. Defining chaos. *Philosophy of Science*, 60(1):43–66, 1993. 31

[37] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield. Powermon: Fine-grained and integrated power monitoring for commodity computer systems. In *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, pages 479–484. IEEE, 2010. 125

[38] S. Behnia, A. Akhshani, H. Mahmodi, and A. Akhavan. A novel algorithm for image encryption based on mixture of chaotic maps. *Chaos, Solitons & Fractals*, 35(2):408–419, 2008. 31

[39] B. Beizer. *Software testing techniques*. Dreamtech Press, 2003. 93

[40] M. Bellare and P. Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:207, 2005. 23

[41] F. Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, pages 37–42. ACM, 2000. 126

[42] G. Bernat, A. Burns, and A. Liamosi. Weakly hard real-time systems. *IEEE transactions on Computers*, 50(4):308–321, 2001. 132

[43] C. S. Bertuglia and F. Vaio. *Nonlinearity, chaos, and complexity: the dynamics of natural and social systems*. Oxford University Press on Demand, 2005. 31

[44] D. Bhattacharjee and A. Chattopadhyay. Hardware accelerator for stream cipher spritz. 2016. 26

[45] E. Biham and O. Dunkelman. Cryptanalysis of the a5/1 gsm stream cipher. In *International Conference on Cryptology in India*, pages 43–51. Springer, 2000. 27

[46] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of a5/1 on a pc. In *International Workshop on Fast Software Encryption*, pages 1–18. Springer, 2000. 27

[47] J. Blazewicz. Deadline scheduling of tasks-a survey. *Foundations of Control Engineering*, 1(4):203–216, 1976. 132

[48] G. Boeing. Visual analysis of nonlinear dynamical systems: Chaos, fractals, self-similarity and the limits of prediction. *Systems*, 4(4):37, 2016. 50

[49] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius. Rabbit: A new high-performance stream cipher. In *International Workshop on Fast Software Encryption*, pages 307–329. Springer, 2003. 27

[50] C. Bonnet and I. Demeure. *Introduction aux systèmes temps réel*. Hermes Science, 1999. 131

[51] C. Bormann, M. Ersue, and A. Keranen. Terminology for constrained-node networks. Technical report, 2014. 14

[52] M. Bougouin. Time performance analysis of estream stream ciphers. Technical report, IETR-Polytech Nantes,France, 2015. 120

[53] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier. Powerapi: A software library to monitor the energy consumed at the process-level. *ERCIM News*, 2013(92), 2013. 126

[54] M. Briceno. A pedagogical implementation of a5/1. *http://www. scard. org*, 1995. 26

[55] M. Briceno, I. Goldberg, and D. Wagner. A pedagogical implementation of the gsm a5/1 and a5/2 "voice privacy" encryption algorithms. *Originally published at http://www. scard. org, mirror at http://cryptome. org/gsm-a512. htm*, 1999. 26

[56] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *The international journal of high performance computing applications*, 14(3):189–204, 2000. 126

[57] A. Burns and A. J. Wellings. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001. 131

[58] G. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011. 131

[59] D. Buttlar, J. Farrell, and B. Nichols. *Pthreads programming: A POSIX standard for better multiprocessing*. " O'Reilly Media, Inc.", 1996. 92

[60] A. Cabrera, F. Almeida, J. Arteaga, and V. Blanco. Energy measurement library (eml) usage and overhead analysis. In *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on*, pages 554–558. IEEE, 2015. 126

[61] C.-C. Chang, M.-S. Hwang, and T.-S. Chen. A new encryption algorithm for image cryptosystems. *Journal of Systems and Software*, 58(2):83–91, 2001. 15, 86

[62] S. Charlwood and P. James-Roxby. Evaluation of the xc6200-series architecture for cryptographic applications. In *International Workshop on Field Programmable Logic and Applications*, pages 218–227. Springer, 1998. 21

[63] H. Cheng, C. Huang, Q. Ding, and S.-C. Chu. An efficient image encryption scheme based on zuc stream cipher and chaotic logistic map. In *Intelligent Data analysis and its Applications, Volume II*, pages 301–310. Springer, 2014. 43

[64] H. Cheng and X. Li. Partial encryption of compressed images and videos. *IEEE Transactions on signal processing*, 48(8):2439–2451, 2000. 15, 86

[65] D. Christin, A. Reinhardt, P. S. Mogre, R. Steinmetz, et al. Wireless sensor networks and the internet of things: selected challenges. *Proceedings of the 8th GI/ITG KuVS Fachgespräch Drahtlose sensornetze*, pages 31–34, 2009. 14

[66] P. Cloutier, P. Mantegazza, S. Papacharalambous, I. Soanes, S. Hughes, and K. Yaghmour. Diapm-rtai position paper, nov 2000. In *Proceedings of the Real-Time Systems Symposium*, 2000. 136

[67] R. M. Corless, G. W. Frank, and J. Graham. Chaos and continued fractions. *Physica D: Nonlinear Phenomena*, 46(2):241–253, 1990. 32

[68] I. Corportation. Ia-32 intel architecture software developer's manual. *Intel Corportation*, 2001. 127

[69] T. W. Cusick and P. Stanica. *Cryptographic Boolean functions and applications*. Academic Press, 2009. 22

[70] J. Daemen and V. Rijmen. The rijndael block cipher: Aes proposal. In *First Candidate Conference (AES1)*, pages 343–348, 1999. 23

[71] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998. 92

[72] N. Developer. Nvidia management library (nvml). 126

[73] D. Dinu, A. Biryukov, J. Großschädl, D. Khovratovich, Y. Corre, and L. Perrin. Felics–fair evaluation of lightweight cryptographic systems. In *NIST Workshop on Lightweight Cryptography*, 2015. 128

[74] T. Eisenbarth and S. Kumar. A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers*, 24(6), 2007. 15, 20, 124

[75] P. Ekdahl and T. Johansson. Snow-a new stream cipher. In *Proceedings of First Open NESSIE Workshop, KU-Leuven*, pages 167–168, 2000. 31

[76] P. Ekdahl and T. Johansson. Another attack on a5/1. *IEEE transactions on information theory*, 49(1):284–289, 2003. 27

[77] S. El Assad and M. Farajallah. A new chaos-based image encryption system. *Signal Processing: Image Communication*, 41:144–157, 2016. 115

[78] S. El Assad and H. Noura. Generator of chaotic sequences and corresponding generating system. WO2011121218 A1 Extension to : Europe EP-2553567 A1, February 2013 ; China : CN-103124955 A, May 2013 ; Japan : JP-2013524271 A, June 2013 ; United states : US-20130170641, July 2013., 10 2011. 60, 62

[79] S. El Assad and H. Noura. Generator of chaotic sequences and corresponding generating system, July 15 2014. US Patent 8,781,116. 87

[80] S. El Assad, H. Noura, and I. Taralova. Design and analyses of efficient chaotic generators for cryptosystems. In *World Congress on Engineering and Computer Science 2008, WCECS'08. Advances in Electrical and Electronics Engineering-IAENG Special Edition of the*, pages 3–12. IEEE, 2008. 62, 87

[81] B. Elaine and K. John. Recommendation for random number generation using deterministic random bit generators. Technical report, NIST SP 800-90 Rev A, 2012. 48, 104

[82] L. Ericsson. More than 50 billion connected devices. *White Paper*, 2011. 14, 124

[83] O. Eriksson. Evaluation of static time analysis for cc systems. *Dept. of Computer Science an Engineering, Märlardalen University, Box*, 883, 2005. 141

[84] R. E. Fairley. An experimental program-testing facility. *IEEE Transactions on Software Engineering*, (4):350–357, 1975. 94

[85] M. Farajallah, S. El Assad, and O. Deforges. Fast and secure chaos-based cryptosystem for images. *International Journal of Bifurcation and Chaos*, 26(02):1650021, 2016. 39, 115

[86] M. A. Feki, F. Kawsar, M. Boussard, and L. Trappeniers. The internet of things: the next technological revolution. *Computer*, 46(2):24–25, 2013. 14, 86, 124

[87] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*, pages 2–10. IEEE, 1999. 125

[88] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of rc4. In *International Workshop on Selected Areas in Cryptography*, pages 1–24. Springer, 2001. 25

[89] M. François, T. Grosges, D. Barchiesi, and R. Erra. Image encryption algorithm based on a chaotic iterative process. *Applied Mathematics*, 3(12):1910, 2012. 86

[90] M. François, T. Grosges, D. Barchiesi, and R. Erra. A new image encryption scheme based on a chaotic function. *Signal Processing: Image Communication*, 27(3):249–259, 2012. 86

[91] M. François, T. Grosges, D. Barchiesi, and R. Erra. A novel pseudo random number generator based on two plasmonic maps. *Applied Mathematics*, 3(11):1664, 2012. 86

[92] M. Francois, T. Grosges, D. Barchiesi, and R. Erra. A new pseudo-random number generator based on two chaotic maps. *Informatica*, 24(2):181–197, 2013. 86

[93] J. Fridrich. Image encryption based on chaotic maps. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, volume 2, pages 1105–1110. IEEE, 1997. 37

[94] J. Fridrich. Symmetric ciphers based on two-dimensional chaotic maps. *International Journal of Bifurcation and chaos*, 8(06):1259–1284, 1998. 36, 37, 46, 48

[95] C. Fu, Z.-c. Zhang, Y. Chen, and X.-w. Wang. An improved chaos-based image encryption scheme. *Computational Science–ICCS 2007*, pages 575–582, 2007. 42

[96] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual International Cryptology Conference*, pages 537–554. Springer, 1999. 14, 20

[97] H. Gao, Y. Zhang, S. Liang, and D. Li. A new chaotic algorithm for image encryption. *Chaos, Solitons & Fractals*, 29(2):393–399, 2006. 15

[98] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, 2010. 125

[99] P. Gerum. Xenomai-implementing a rtos emulation framework on gnu/linux. *White Paper, Xenomai*, page 81, 2004. 136

[100] T. Good and M. Benaissa. A low-frequency rfid to challenge security and privacy concerns. In *Mobile Adhoc and Sensor Systems, 2009. MASS'09. IEEE 6th International Conference on*, pages 856–863. IEEE, 2009. 20, 124

[101] D. E. Goumidi and F. Hachouf. Hybrid chaos-based image encryption approach using block and stream ciphers. In *Systems, Signal Processing and their Applications (WoSSPA), 2013 8th International Workshop on*, pages 139–144. IEEE, 2013. 43

[102] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979. 132

[103] P. Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Proceedings of the Sixth USENIX Security Symposium, San Jose, CA*, volume 14, pages 77–89, 1996. 95

[104] Z. Gutterman, B. Pinkas, and T. Reinman. Analysis of the linux random number generator. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006. 88

[105] T. Hardjono and L. R. Dondeti. *Security in Wireless LANS and MANS (Artech House Computer Security)*. Artech House, Inc., 2005. 21

[106] M. Hénon. A two-dimensional mapping with a strange attractor. In *The Theory of Chaotic Attractors*, pages 94–102. Springer, 1976. 33

[107] J. Holler, V. Tsiatsis, C. Mulligan, S. Avesand, S. Karnouskos, and D. Boyle. *From Machine-to-machine to the Internet of Things: Introduction to a New Age of Intelligence*. Academic Press, 2014. 14

[108] W. Horn. Some simple scheduling algorithms. *Naval Research Logistics (NRL)*, 21(1):177–185, 1974. 132

[109] A. Hubler. Adaptive-control of chaotic systems. *Helvetica Physica Acta*, 62(2-3):343–346, 1989. 20, 31

[110] B. S. C. L. C. J. Tayeb, K. Bross and S. Rogers. Intel energy checker software development kit user guide. 2010-12-15. 126

[111] G. Jakimoski and L. Kocarev. Analysis of some recently proposed chaos-based encryption algorithms. *Physics Letters A*, 291(6):381–384, 2001. 38

[112] O. Jallouli, M. Abutaha, S. El Assad, M. Chetto, A. Queudet, and O. Deforges. Comparative study of two pseudo chaotic number generators for securing the iot. In *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on*, pages 1340–1344. IEEE, 2016. 43, 44, 111

[113] O. Jallouli, S. El Assad, and M. Chetto. Robust chaos-based stream-cipher for secure public communication channels. In *International Conference on Internet Technology and Secured Transactions*, 2016. 127, 144

[114] O. Jallouli, S. El Assad, M. Chetto, and R. Lozi. Design and analysis of two stream ciphers based on chaotic coupling and multiplexing techniques. *Multimedia tools and applications, accepted with minor corrections*, 2017. 127, 144

[115] A. Jolfaei and A. Mirghadri. Image encryption using chaos and block cipher. *Computer and Information Science*, 4(1):172, 2010. 15

[116] R. R. Jueneman. Analysis of certain aspects of output feedback mode. In *Advances in Cryptology*, pages 99–127. Springer, 1983. 26

[117] D. Kahn. *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet*. Simon and Schuster, 1996. 14, 20

[118] K. Kaneko. Pattern dynamics in spatiotemporal chaos: Pattern selection, diffusion of defect and pattern competition intermettency. *Physica D: Nonlinear Phenomena*, 34(1-2):1–41, 1989. 38

[119] H. Karimi, S. Morteza Hosseini, and M. Vafaei Jahan. On the combination of self-organized systems to generate pseudo-random numbers. *Information Sciences*, 221:371–388, 2013. 46

[120] J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2014. 25

[121] A. Kerckhoffs. *La cryptographie militaire*. University Microfilms, 1978. 21

[122] Y. S. Kim and W. W. Zachary. *The Physics of Phase Space: Nonlinear Dynamics and Chaos, Geometric Quantization, and Wigner Function*, volume 278. Springer, 2006. 47

[123] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976. 94

[124] A. Klein. Attacks on the rc4 stream cipher. *Designs, Codes and Cryptography*, 48(3):269–286, 2008. 25

[125] L. R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis methods for (alleged) rc4. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 327–341. Springer, 1998. 25

[126] L. Kocarev and G. Jakimoski. Pseudorandom bits generated by chaotic maps. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 50(1):123–126, 2003. 40, 41, 48

[127] L. Kocarev and S. Lian. *Chaos-based cryptography: Theory, algorithms and applications*, volume 354. Springer, 2011. 15, 20, 46

[128] P. Koopman. Embedded system security. *Computer*, 37(7):95–97, 2004. 124

[129] H. Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011. 131

[130] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2010. 14

[131] Z. Kotulski and J. Szczepanski. Discrete chaotic cryptography (dcc). In *Proc NEEDS*, volume 97, 1997. 46

[132] Y. Kumar, R. Munjal, and H. Sharma. Comparison of symmetric and asymmetric cryptography with existing vulnerabilities and countermeasures. *International Journal of Computer Science and Management Studies*, 11(03), 2011. 14, 20

[133] A. P. Kurian and S. Puthusserypady. Self-synchronizing chaotic stream ciphers. *Signal Processing*, 88(10):2442–2452, 2008. 42

[134] H. Kwok and W. K. Tang. A fast image encryption system based on chaotic maps with finite precision representation. *Chaos, solitons & fractals*, 32(4):1518–1529, 2007. 42

[135] P. A. Laplante et al. *Real-time systems design and analysis*. 131

[136] P. Larsson. Energy-efficient software guidelines. *Intel Software Solutions Group, Tech. Rep*, 2011. 128

[137] E. Lawler. Recent results in the theory of machine scheduling. In *Mathematical programming the state of the art*, pages 202–234. Springer, 1983. 132

[138] P.-H. Lee, S.-C. Pei, and Y.-Y. Chen. Generating chaotic stream ciphers using chaotic systems. *Chinese Journal of physics*, 41(6):559–581, 2003. 41

[139] H. K. Leung and L. White. Insights into regression testing (software testing). In *Software Maintenance, 1989., Proceedings., Conference on*, pages 60–69. IEEE, 1989. 93

[140] C. Li, S. Li, G. Alvarez, G. Chen, and K.-T. Lo. Cryptanalysis of two chaotic encryption schemes based on circular bit shift and xor operations. *Physics Letters A*, 369(1):23–30, 2007. 42

[141] C. Li, S. Li, and K.-T. Lo. Breaking a modified substitution–diffusion image cipher based on chaotic standard and logistic maps. *Communications in Nonlinear Science and Numerical Simulation*, 16(2):837–843, 2011. 43

[142] C.-g. Li, Z.-z. Han, and H.-R. Zhang. Image encryption techniques: A survey [j]. *Journal of Computer Research and Development*, 10:023, 2002. 15

[143] S. Li, G. Chen, and X. Mou. On the dynamical degradation of digital piecewise linear chaotic maps. *International Journal of Bifurcation and Chaos*, 15(10):3119–3151, 2005. 60

[144] S. Li, G. Chen, K.-W. Wong, X. Mou, and Y. Cai. Baptista-type chaotic cryptosystems: problems and countermeasures. *Physics Letters A*, 332(5):368–375, 2004. 38

[145] S. Li, X. Mou, Z. Ji, J. Zhang, and Y. Cai. Performance analysis of jakimoski–kocarev attack on a class of chaotic cryptosystems. *Physics Letters A*, 307(1):22–28, 2003. 38

[146] S. Lian, J. Sun, J. Wang, and Z. Wang. A chaotic stream cipher and the usage in video protection. *Chaos, Solitons & Fractals*, 34(3):851–859, 2007. 59

[147] S. Lian, J. Sun, and Z. Wang. A block cipher based on a suitable use of the chaotic standard map. *Chaos, Solitons & Fractals*, 26(1):117–129, 2005. 38

[148] S. Lian, J. Sun, and Z. Wang. Security analysis of a chaos-based image encryption algorithm. *Physica A: Statistical Mechanics and its Applications*, 351(2):645–661, 2005. 37

[149] H. Lipmaa, D. Wagner, and P. Rogaway. Comments to nist concerning aes modes of operation: Ctr-mode encryption. 2000. 26

[150] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973. 132

[151] H. Liu and X. Wang. Color image encryption based on one-time keys and robust chaotic maps. *Computers & Mathematics with Applications*, 59(10):3320–3327, 2010. 43, 121

[152] J. W. Liu. Real-time systems. 2000. 131

[153] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141, 1963. 34

[154] R. Lozi. Un attracteur étrange (?) du type attracteur de hénon. *Le Journal de Physique Colloques*, 39(C5):C5–9, 1978. 33

[155] R. Lozi. New enhanced chaotic number generators. *Indian Journal of Industrial and Applied Mathematics*, 1(1):1–23, 2007. 48, 87

[156] R. Lozi. Emergence of randomness from chaos. *International Journal of Bifurcation and Chaos*, 22(02):1250021–1 –1250021–15, 2012. 87, 99

[157] R. Lozi and E. Cherrier. Noise-resisting ciphering based on a chaotic multi-stream pseudo-random number generator. In *Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*, pages 91–96. IEEE, 2011. 48

[158] F. Maleki, A. Mohades, S. M. Hashemi, and M. E. Shiri. An image encryption system by cellular automata with memory. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 1266–1271. IEEE, 2008. 113

[159] R. O. Manansala, D. J. A. Quino, J. R. I. Pedrasa, and M. A. A. Pedrasa. Design and implementation of a web-based smart power distribution unit. In *Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), 2014 International Conference on*, pages 1–7. IEEE, 2014. 125

[160] C. Manchein and M. W. Beims. Gauss map and lyapunov exponents of interacting particles in a billiard. *Chaos, Solitons & Fractals*, 39(5):2041–2047, 2009. 32

[161] S. S. Maniccam and N. G. Bourbakis. Image and video encryption using scan patterns. *Pattern Recognition*, 37(4):725–737, 2004. 15

[162] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and Y. Papaefstathiou. A survey of lightweight stream ciphers for embedded systems. *Security and Communication Networks*, 2015. 27

[163] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and K. Rantos. Lightweight cryptography for embedded systems–a comparative analysis. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 333–349. Springer, 2014. 130

[164] P. Mantegazza, E. Dozio, and S. Papacharalambous. Rtai: Real time application interface. *Linux Journal*, 2000(72es):10, 2000. 135

[165] W. Mao. *Modern cryptography: theory and practice*. Prentice Hall Professional Technical Reference, 2003. 14

[166] Y. Mao, G. Chen, and S. Lian. A novel fast image encryption scheme based on 3d chaotic baker maps. *International Journal of Bifurcation and chaos*, 14(10):3613–3624, 2004. 38

[167] M. Maqableh, A. B. Samsudin, and M. A. Alia. New hash function based on chaos theory (cha-1). *International Journal of Computer Science and Network Security*, 8(2):20–27, 2008. 20

[168] J. Martínez-Ñonthe, A. Castañeda-Solís, A. Díaz-Méndez, M. Cruz-Irisson, and R. Vazquez-Medina. Chaotic block cryptosystem using high precision approaches to tent map. *Microelectronic Engineering*, 90:168–172, 2012. 40

[169] P. Marwedel. *Embedded system design: Embedded systems foundations of cyber-physical systems*. Springer Science & Business Media, 2010. 130

[170] A. Massoudi, F. Lefebvre, C. De Vleeschouwer, B. Macq, and J.-J. Quisquater. Overview on selective encryption of image and video: challenges and perspectives. *EURASIP Journal on Information Security*, 2008(1):179290, 2008. 15

[171] N. Masuda and K. Aihara. Cryptosystems with discretized chaotic maps. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(1):28–40, 2002. 46

[172] M. Matsui. New block encryption algorithm misty. In *International Workshop on Fast Software Encryption*, pages 54–68. Springer, 1997. 23

[173] R. Matthews. On the derivation of a “chaotic” encryption algorithm. *Cryptologia*, 13(1):29–42, 1989. 15, 40, 86

[174] R. M. May et al. Simple mathematical models with very complicated dynamics. *Nature*, 261(5560):459–467, 1976. 49

[175] S. Mazloom and A. M. Eftekhari-Moghadam. Color image encryption based on coupled nonlinear chaotic map. *Chaos, Solitons & Fractals*, 42(3):1745–1754, 2009. 15

[176] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Chapter 6: Stream Ciphers, "Handbook of applied cryptography"*. CRC press, 1996. 24

[177] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Chapter 7: Block Ciphers, "Handbook of applied cryptography"*. CRC press, 1996. 14, 22, 23

[178] M. Misiurewicz. Strange attractors for the lozi mappings. *Annals of the New York Academy of Sciences*, 357(1):348–358, 1980. 34

[179] S. Mister and S. E. Tavares. Cryptanalysis of rc4-like ciphers. In *International Workshop on Selected Areas in Cryptography*, pages 131–143. Springer, 1998. 25

[180] E. Moglen. Rtai and the rt-linux patent, 2001. 135

[181] P. Mukherjee. An overview of estream ciphers. In *[Online], http://cs.au.dk/ pratyay/eSTREAM.pdf*, last accessed 2017, June. 30

[182] N. Nethercote and J. Seward. Valgrind: A program supervision framework. *Electronic notes in theoretical computer science*, 89(2):44–66, 2003. 94

[183] N. Nissanke. *Realtime systems*. 1997. 132

[184] L. Pace. Chi-square tests. In *Beginning R*, pages 217–228. Springer, 2012. 47

[185] P. Pacheco. *An introduction to parallel programming*. Elsevier, 2011. 92

[186] R. Pakshwar, V. K. Trivedi, and V. Richhariya. A survey on different image encryption and decryption techniques. *International journal of computer science and information technologies*, 4(1):113–116, 2013. 15

[187] A. Palacios and H. Juarez. Cryptography with cycling chaos. *Physics Letters A*, 303(5):345–351, 2002. 38

[188] S. Papadimitriou, T. Bountis, S. Mavroudi, and A. Bezerianos. A probabilistic symmetric encryption scheme for very fast secure communication based on chaotic systems of difference equations. *International Journal of Bifurcation and Chaos*, 11(12):3107–3115, 2001. 59

[189] H. Pastijn. Chaotic growth with the logistic model of p.-f. verhulst. In *The Logistic Map and the Route to Chaos*, pages 3–11. Springer, 2006. 49

[190] V. Patidar, N. Pareek, G. Purohit, and K. Sud. Modified substitution–diffusion image cipher using chaotic standard and logistic maps. *Communications in Nonlinear Science and Numerical Simulation*, 15(10):2755–2765, 2010. 43

[191] V. Patidar, N. Pareek, and K. Sud. A new substitution–diffusion based image cipher using chaotic standard and logistic maps. *Communications in Nonlinear Science and Numerical Simulation*, 14(7):3056–3075, 2009. 43

[192] V. Patidar, K. K. Sud, and N. K. Pareek. A pseudo random bit generator based on chaotic logistic map and its statistical testing. *Informatica (slovenia)*, 33(4):441–452, 2009. 86

[193] G. Paul and S. Maitra. *RC4 stream cipher and its variants*. CRC press, 2011. 25

[194] G. Paul, S. Maitra, and S. Raizada. A theoretical analysis of the structure of hc-128. In *International Workshop on Security*, pages 161–177. Springer, 2011. 29

[195] K. Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110, 1894. 47

[196] J. Peng, M. You, Z. Yang, and S. Jin. Research on a block encryption cipher based on chaotic dynamical system. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 5, pages 744–748. IEEE, 2007. 50

[197] H. Petersen, E. Baccelli, and M. Wählisch. Interoperable services on constrained devices in the internet of things. In *W3C Workshop on the Web of Things*, 2014. 14

[198] M. V. Petersen and H. M. B. Sørensen. Method of generating pseudo-random numbers in an electronic device, and a method of encrypting and decrypting electronic data, Jan. 30 2007. US Patent 7,170,997. 87

[199] S. Phatak and S. S. Rao. Logistic map: A possible random-number generator. *Physical review E*, 51(4):3670, 1995. 48

[200] N. S. Philip and K. B. Joseph. Chaos for stream cipher. *arXiv preprint cs/0102012*, 2001. 40, 41

[201] A. Popov. Prohibiting rc4 cipher suites. *Computer Science*, 2355:152–164, 2015. 25

[202] A. Y. Poschmann. Lightweight cryptography: cryptographic engineering for a pervasive world. In *PH. D. THESIS*. Citeseer, 2009. 15

[203] M. J. Quinn. *Parallel computing: theory and practice*. McGraw-Hill, Inc., 1994. 92

[204] D. C. Ranasinghe and P. H. Cole. Confronting security and privacy threats in modern rfid systems. In *Signals, Systems and Computers, 2006. ACSSC'06. Fortieth Asilomar Conference on*, pages 2058–2064. IEEE, 2006. 15

[205] D. C. Ranasinghe, D. W. Engels, and P. H. Cole. Low cost rfid systems: confronting security and privacy. *Paper Auto-ID Labs White Paper Journal*, 1, 2005. 15

[206] L. Rauchwerger, F. Arzu, and K. Ouchi. Standard templates adaptive parallel library (stapl). In *International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, pages 402–409. Springer, 1998. 92

[207] N. Recipes. Wh press, bp flannery, sa teukolsky, wt vetterling, 1989. 47

[208] R. Rhouma and S. Belghith. A piecewice linear chaotic map for baptista-type cryptosystem. *Proc. of SSD'07 Tunisia*, 2007. 38

[209] R. Rhouma, E. Solak, and S. Belghith. Cryptanalysis of a new substitution–diffusion based image cipher. *Communications in Nonlinear Science and Numerical Simulation*, 15(7):1887–1892, 2010. 43

[210] H. Richter. The generalized henon maps: examples for higher-dimensional chaos. *International Journal of Bifurcation and Chaos*, 12(06):1371–1384, 2002. 33

[211] R. L. Rivest. The rc5 encryption algorithm. In *International Workshop on Fast Software Encryption*, pages 86–96. Springer, 1994. 23

[212] R. L. Rivest. The rc4 encryption algorithm, 1992. *RSA Data Security Inc*, 2016. 25

[213] R. L. Rivest, M. Robshaw, R. Sidney, and Y. L. Yin. The rc6tm block cipher. In *First Advanced Encryption Standard (AES) Conference*, page 16, 1998. 23

[214] R. L. Rivest and J. C. Schuldt. Spritz—a spongy rc4-like stream cipher and hash function. *Proceedings of the Charles River Crypto Day, Palo Alto, CA, USA*, 24:10, 2014. 26

[215] M. Robshaw. Stream ciphers, rsa laboratories technical report tr-701. *Redwood City, CA*, 1995. 25

[216] M. Robshaw. The estream project. In *New Stream Cipher Designs*, pages 1–6. Springer, 2008. 27

[217] P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm. *Journal of Cryptology*, 11(4):273–287, 1998. 31

[218] R. Roman, C. Alcaraz, and J. Lopez. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. *Mobile Networks and Applications*, 12(4):231–244, 2007. 14

[219] R. Roman, P. Najera, and J. Lopez. Securing the internet of things. *Computer*, 44(9):51–58, 2011. 86

[220] O. Rossler. An equation for hyperchaos. *Physics Letters A*, 71(2-3):155–157, 1979. 35

[221] O. E. Rössler. An equation for continuous chaos. *Physics Letters A*, 57(5):397–398, 1976. 35

[222] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. Power-management architecture of the intel microarchitecture code-named sandy bridge. *Ieee micro*, 32(2):20–27, 2012. 127

[223] R. A. Rueppel. *Analysis and design of stream ciphers*. Springer Science & Business Media, 2012. 24

[224] A. L. Rukhin, J. Soto, J. R. Nechvatal, M. Smid, E. B. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, NIST SP 800-22 Rev 1, 2008. 48, 104

[225] S. Ryffel. Lea2p-the linux energy attribution and accounting platform. *Master's thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland*, 2009. 126

[226] S. Ryffel, T. Stathopoulos, D. McIntire, W. Kaiser, and L. Thiele. Accurate energy attribution and accounting for multi-core systems. *Center for Embedded Network Sensing*, 2009. 126

[227] C. Sakamoto, T. Miyazaki, M. Kuwayama, K. Saisho, and A. Fukuda. Design and implementation of a parallel pthread library (ppl) with parallelism and portability. *Systems and Computers in Japan*, 29(2):28–35, 1998. 92

[228] J. Scharinger. Fast encryption of image data using chaotic kolmogorov flows. *Journal of Electronic Imaging*, 7(2):318–325, 1998. 86

[229] B. Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast software encryption*, pages 191–204. Springer, 1994. 23

[230] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: A 128-bit block cipher. *NIST AES Proposal*, 15, 1998. 23

[231] A. Sehgal, V. Perelman, S. Kuryla, and J. Schonwalder. Management of resource constrained devices in the internet of things. *IEEE Communications Magazine*, 50(12), 2012. 14

[232] O. Serlin. Scheduling of time critical processes. In *Proceedings of the May 16-18, 1972, spring joint computer conference*, pages 925–932. ACM, 1972. 132

[233] J. Seward and N. Nethercote. Using valgrind to detect undefined value errors with bit-precision. In *USENIX Annual Technical Conference, General Track*, pages 17–30, 2005. 94

[234] J. Shah and V. Saxena. Performance study on image encryption schemes. *arXiv preprint arXiv:1112.0836*, 2011. 15

[235] C. Shannon. A mathematical theory of communication, bell system technical journal 27: 379-423 and 623–656. *Mathematical Reviews (MathSciNet): MR10, 133e*, 1948. 20

[236] C. E. Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4):656–715, 1949. 22, 31

[237] D. Shin, H. Shim, Y. Joo, H.-S. Yun, J. Kim, and N. Chang. Energy-monitoring tool for low-power embedded programs. *IEEE Design & Test of Computers*, 19(4):7–17, 2002. 126

[238] C. Shuai and X.-X. Zhong. Chaotic block iterating method for pseudo-random sequence generator. *The Journal of China Universities of Posts and Telecommunications*, 14(1):45–48, 2007. 40

[239] L. Shujun, M. Xuanqin, and C. Yuanlong. Pseudo-random bit generator based on couple chaotic systems and its applications in stream-cipher cryptography. In *Progress in Cryptology—INDOCRYPT 2001*, pages 316–329. Springer, 2001. 40, 48, 60, 86

[240] L. Shujuna, M. Xuanqinb, and C. Yuanlongc. Pseudo-random bit generator based on couple chaotic systems and its applications in stream-cipher cryptography. In *International Conference on Cryptology in India (INDOCRYPT*, volume 16, page 20, 2001. 40, 41

[241] M. Silly-Chetto. Sur la problématique de l'ordonnancement dans les systèmes informatiques temps réel. *Rapport d'HDR, Université de Nantes*, 21, 1993. 133

[242] G. J. Simmons. Symmetric and asymmetric encryption. *ACM Computing Surveys (CSUR)*, 11(4):305–330, 1979. 14, 20

[243] S. Singh. *The code book: the science of secrecy from ancient Egypt to quantum cryptography*. Anchor, 2000. 20

[244] A. Skrobek. Cryptanalysis of chaotic stream cipher. *Physics Letters A*, 363(1):84–90, 2007. 41

[245] E. Solak, C. Çokal, O. T. Yildiz, and T. BIYIKOĞLU. Cryptanalysis of fridrich's chaotic image encryption. *International Journal of Bifurcation and Chaos*, 20(05):1405–1413, 2010. 37

[246] C. Sparrow. *The Lorenz equations: bifurcations, chaos, and strange attractors*, volume 41. Springer Science & Business Media, 2012. 34

[247] W. Stallings. *Cryptography and network security: principles and practices*. Pearson Education India, 2006. 21

[248] W. Stallings. *Cryptography and Network Security: Principles and Practice, International Edition: Principles and Practice*. Pearson Higher Ed, 2014. 109

[249] M. Stamp and R. M. Low. *Applied cryptanalysis: breaking ciphers in the real world*. John Wiley & Sons, 2007. 36

[250] D. E. Standard et al. Federal information processing standards publication 46. *National Bureau of Standards, US Department of Commerce*, 1977. 23

[251] J. A. Stankovic. Real-time computing system: The next generation. 1988. 124

[252] T. Stathopoulos, D. McIntire, and W. J. Kaiser. The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes. In *Proceedings of the 7th international conference on Information processing in sensor networks*, pages 383–394. IEEE Computer Society, 2008. 125

[253] D. Stepner, N. Rajan, and D. Hui. Embedded application design using a real-time os. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 151–156. IEEE, 1999. 130

[254] D. B. Stewart. Twenty-five most common mistakes with real-time software development. In *Proceedings of the 1999 Embedded Systems Conference (ESC'99)*, 1999. 141

[255] D. B. Stewart. Measuring execution time and real-time performance. In *Embedded Systems Conference (ESC)*, 2001. 141

[256] T. Stojanovski and L. Kocarev. Chaos-based random number generators-part i: analysis [cryptography]. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48(3):281–288, 2001. 40

[257] S. H. Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. Westview press, 2014. 49

[258] B. Sunar, W. J. Martin, and D. R. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Transactions on computers*, 56(1):109–119, 2007. 46

[259] X.-j. Tong, M.-g. Cui, and W. Jiang. The production algorithm of pseudo-random number generator based on compound non-linear chaos system. In *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP'06. International Conference on*, pages 685–688. IEEE, 2006. 40

[260] W. Trappe, R. Howard, and R. S. Moore. Low-energy security: Limits and opportunities in the internet of things. *IEEE Security & Privacy*, 13(1):14–21, 2015. 14

[261] J. Treibig, G. Hager, and G. Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 207–216. IEEE, 2010. 126

[262] A. Ukil, J. Sen, and S. Koilakonda. Embedded security for internet of things. In *Emerging Trends and Applications in Computer Science (NCETACS), 2011 2nd National Conference on*, pages 1–6. IEEE, 2011. 86

[263] L. university. Cryptolux > felics. In *https://www.cryptolux.org/index.php/FELICS*. 2016. 128, 129

[264] P. F. Verhulst. Recherches mathématiques sur la loi d'accroissement de la population. *Nouveaux mémoires de l'académie royale des sciences et belles-lettres de Bruxelles*, 18:14–54, 1845. 49

[265] G. Vidal, M. S. Baptista, and H. Mancini. A fast and light stream cipher for smartphones. *The European Physical Journal Special Topics*, 223(8):1601–1610, 2014. 43, 120, 121

[266] X. Wang, J. Zhang, Y. Fan, and W. Zhang. Chaotic pseudorandom bit generator using n-dimensional nonlinear digital filter. In *Communication Technology, 2006. ICCT'06. International Conference on*, pages 1–4. IEEE, 2006. 42

[267] Y. Wang, Z. Liu, J. Ma, and H. He. A pseudorandom number generator based on piecewise logistic map. *Nonlinear Dynamics*, 83(4):2373–2391, 2016. 107, 111

[268] Y. Wang, K.-W. Wong, X. Liao, and G. Chen. A new chaos-based fast image encryption algorithm. *Applied soft computing*, 11(1):514–522, 2011. 38, 39

[269] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 262–268. IEEE, 2012. 126

[270] J. Weidendorfer. Cache performance analysis with callgrind and kcachegrind. In *8 VI-HPS Tuning Workshop*. 94

[271] M. Weiser. The computer for the 21st century. *Mobile Computing and Communications Review*, 3(3):3–11, 1999. 148

[272] E. W. Weisstein. "hénon map." from mathworld–a wolfram web resource. 34

[273] E. W. Weisstein. "lozi map." from mathworld–a wolfram web resource. 34

[274] E. Welbourne, L. Battle, G. Cole, K. Gould, K. Rector, S. Raymer, M. Balazinska, and G. Borriello. Building the internet of things using rfid: the rfid ecosystem experience. *IEEE Internet computing*, 13(3), 2009. 14

[275] D. D. Wheeler. Problems with chaotic cryptosystems. *Cryptologia*, 13(3):243–250, 1989. 40

[276] S. Wolfram. Cryptography with cellular automata. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 429–432. Springer, 1985. 40

[277] K.-W. Wong. A fast chaotic cryptographic scheme with dynamic look-up table. *Physics Letters A*, 298(4):238–242, 2002. 38

[278] K.-W. Wong. A combined chaotic cryptographic and hashing scheme. *Physics letters A*, 307(5):292–298, 2003. 38

[279] W.-k. Wong, L.-p. Lee, and K.-w. Wong. A modified chaotic cryptographic method. In *Communications and Multimedia Security Issues of the New Century*, pages 123–126. Springer, 2001. 38

[280] H. Wu. A new stream cipher hc-256. In *Fast Software Encryption*, pages 226–244. Springer, 2004. 29

[281] Y. Wu, J. P. Noonan, and S. Agaian. Npcr and uaci randomness tests for image encryption. *Cyber journals: multidisciplinary journals in science and technology, Journal of Selected Areas in Telecommunications (JSAT)*, pages 31–38, 2011. 113

[282] K. L. Wuensch. Chi-square tests. In *International Encyclopedia of Statistical Science*, pages 252–253. Springer, 2011. 47

[283] V. Yodaiken, C. Dougan, M. Barabanov, et al. Rtlinux/rtcore dual kernel real-time operating system. *FSMLabs, White Paper*, 2003. 135

[284] V. Yodaiken et al. The rtlinux manifesto. In *Proc. of the 5th Linux Expo*, 1999. 135

[285] T. Yoshida, H. Mori, and H. Shigematsu. Analytic study of chaos of the tent map: band structures, power spectra, and critical behaviors. *Journal of statistical physics*, 31(2):279–308, 1983. 33

[286] W. Yu and J. Cao. Cryptography based on delayed chaotic neural networks. *Physics Letters A*, 356(4):333–338, 2006. 42

[287] X. Zeng, R. A. Pielke, and R. Eykholt. Chaos theory and its applications to the atmosphere. *Bulletin of the American Meteorological Society*, 74(4):631–644, 1993. 20, 31

[288] E. Zeraoulia. Lozi mappings: Theory and applications. 2013. 34

[289] W. Zhang, K.-w. Wong, H. Yu, and Z.-l. Zhu. An image encryption scheme using reverse 2-dimensional chaotic map and dependent diffusion. *Communications in Nonlinear Science and Numerical Simulation*, 18(8):2066–2080, 2013. 39

[290] F. Zheng, X.-j. Tian, J.-y. Song, and X.-Y. Li. Pseudo-random sequence generator based on the generalized henon map. *The Journal of China Universities of Posts and Telecommunications*, 15(3):64–68, 2008. 40

[291] H. Zhou. A design methodology of chaotic stream ciphers and the realization problems in finite precision. *Department of Electrical Engineering, Fudan University, Shanghai, China*, 1996. 59

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Ons JALLOULI

Sécurité basée Chaos sous contraintes temps réel et d'énergie pour l'Internet des Objets

Chaos-based security under real-time and energy constraints for the Internet of Things

Résumé
De nos jours, la croissance rapide des technologies de l'Internet des Objets (IoT) rend la protection des données transmises un enjeu important. Les dispositifs de l'IoT sont intrinsèquement contraints à la mémoire, à la puissance de traitement et à l'énergie disponible. Ceci implique que la conception de techniques cryptographiques sécurisées, efficaces et légères est cruciale. Dans cette thèse, nous avons étudié la problématique de la sécurité de l'information basée chaos sous contraintes temps réel et d'énergie. À ce sujet, nous avons conçu et implémenté dans un premier temps, trois générateurs de nombres pseudo-chaotiques (PCNGs). Ces PCNGs utilisent une matrice de couplage faible ou une matrice de couplage binaire à forte diffusion entre des cartes chaotiques, et une technique de multiplexage chaotique. Puis, nous avons réalisé trois systèmes de chiffrement/déchiffrement par flux basés sur les PCNGs proposés. L'analyse cryptographique des systèmes chaotiques réalisés a montré leur robustesse contre des attaques connues. La performance obtenue en complexité de calcul met bien en évidence leur utilisation dans des applications temps réel. Dans un second temps, nous avons intégré ces systèmes de chiffrement/déchiffrement chaotiques au sein du système d'exploitation temps réel Xenomai. Enfin, nous avons mesuré la consommation d'énergie et de puissance des trois systèmes chaotiques réalisés ainsi que le temps moyen de chiffrement/déchiffrement.

Mots clés
Internet des Objets, Sécurité des Données, Générateurs de nombres pseudo-chaotiques, Chiffrement par flux basé chaos, Analyse de la sécurité, Vitesse de chiffrement, Consommation d'énergie, Temps réel.

Abstract
Nowadays, due to the rapid growth of Internet of Things (IoT) towards technologies, the protection of transmitted data becomes an important challenge. The devices of the IoT are very constrained resource in terms of computing capabilities, energy and memory capacities. Thus, the design of secure, efficient and lightweight crypto-systems becomes more and more crucial. In this thesis, we have studied the problem of chaos based data security under real-time and energy constraints. First, we have designed and implemented three pseudo-chaotic number generators (PCNGs). These PCNGs use a weak coupling matrix or a high diffusion binary coupling matrix between chaotic maps and a chaotic multiplexing technique. Then, we have realized three stream ciphers based on the proposed PCNGs. Security performance of the proposed stream ciphers were analysed and several cryptanalytic and statistical tests were applied. Experimental results highlight robustness as well as efficiency in terms of computation time. The performance obtained in computational complexity indicates their use in real-time applications. Then, we integrated these chaotic stream ciphers within the real-time operating system Xenomai. Finally, we have measured the energy and power consumption of the three proposed chaotic systems, and the average computing performance. The obtained results show that the proposed stream ciphers can be used in practical IoT applications.

Key Words
Internet of Things, Chaos-based Data Security, Pseudo-Chaotic Number Generators, Chaos-Based Stream Cipher, Security Analysis, Computing Performance, Power Consumption, Real-Time.

L'UNIVERSITÉ BRETAGNE LOIRE