Politecnico di Torino · Computer Engineering

Candidate: Davide Arcolini
Academic Year: 2020/2021

Pseudo-Random Number Generator
Logistic Chaotic Map and Generalized Lorenz System

Teacher:	Igor Simone Stievano
Supervisor:	Fernando Corinto
Course:	Reti e sistemi complessi
Academic Year:	2020/2021
Candidate:	Davide Arcolini
Contact:	s256671@studenti.polito.it
Date of last revision	May 30, 2021

Contents

Introduction

During the last few years, we have witnessed a significant evolution of Information Security Systems. With the rapid development of the internet through our daily life, from managing our bank account to storing personal data, we have started to seriously consider the emerging CyberSecurity problems and challenges. Since there is an interesting relationship between chaos and cryptography, the aim of this thesis is to reproduce, simulate and analyze the results obtained from the implementation of two **chaos-based pseudo-random number generators**: a Logistic Chaotic System and a Generalized Lorenz System.
In particular, the thesis is based on the review of two recent-published papers: the 2019 paper from Entropy: "*Pseudo-Random Number Generator based on Logistic Chaotic System*" [1], and the 2015 paper from IFAC: "*Pseudo-Random Number Generator based on the Generalized Lorenz Chaotic System*" [2].

The thesis is structured in the following way:

- **Chapter: Introduction**. It gives an overview of the PRNG algorithms and introduces to the dynamical systems that lead to chaotic behavior. Chaos theory is introduced in order to justify the need of chaos-based PRNGs.
- **Chapter: Logistic Chaotic System**. The logistic map is presented and the algorithm for the generation of pseudo-random numbers is deeply analyzed. Results obtained are studied in order to demonstrate the effectiveness of the system.
- **Chapter: Generalized Lorenz System**. The generalized Lorenz system is presented and the algorithm for the generation of pseudo-random numbers is deeply analyzed. Results obtained are studied in order to demonstrate the effectiveness of the system.
- **Chapter: Conclusions**. It gives a final overview of the thesis, summarizing the results obtained.

All the codes used to simulate the chaotic systems in this thesis and to generate the diagrams can be found at the link in the **code section**. Indeed, all the codes have been uploaded with the MIT LICENSE on GITHUB and can be easily downloaded and run.

1.1 Pseudo-Random Number Generators

In theoretical computer science and cryptography, a **Pseudo-Random Number Generator** (**PRNG**), for a specific class of statistical tests, is a **deterministic algorithm** that translates a random seed to a longer string such that none of the statistical tests in the class are able to distinguish between the output of the generator and the uniform distribution.

Since the algorithm is fully deterministic, the PRNG-sequence generated is not completely random. The aim of the generators is to produce numbers that approximate the properties of sequences of true random number.

Indeed, a **True-Random Numbers Generator** (**TRNG**) is a piece of hardware capable of generating random numbers from a **physical process**, rather than by means of an algorithmic procedure. Such devices are often based on microscopic phenomena that generate low-level, statistically random ”noise” signals, such as thermal noise, photoelectric effect and other quantum phenomena. TRNG happens to be used to generate the random initial seed that will be fed into the PRNG algorithm.

However, the most common PRNG exhibits **artifacts** that does not allow them to success the pattern-detection statistical tests. These problems typically include:

- **Periods generation**: circular repetitions of specific output numbers in the generated sequence. When periods occur, the seed that generated that particular sequence is called a ”weak” seed.
- **Distribution**: lack of uniformity that occurs for large quantities of generated numbers: some sequences generated as output may exhibit with a relative frequency higher than others;
- **Correlation** between successive numbers. It should be computational impossible to infer the successive sequences given the current state of the algorithm.
- Poor dimensional distribution of the output sequence.

The mathematical formality adopted to statistically describe PRNGs is not proposed here since it is out of the scope of this thesis.

John von Neumann proposed the first algorithm to generate pseudo-random numbers. Although the procedure, in its original form, is of extremely poor quality and does not provide any **cryptographic security**, the algorithm is of interest for historical reasons and it is, therefore, presented here.

1.1.1 John von Nuemann: middle-square method

In a 1949 conference, the Hungarian-American mathematician, physicist and computer scientist, **John von Neumann**, recognize the usefulness of deterministic methods for producing pseudo-random numbers. Since a True-Random Number Generator would have been too slow to be used in his ENIAC work, von Neumann proposes the so called *middle-square method* stating, however, that: ”*anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin*”. [3]

The formalization of the method is straightforward and it is the following:

A n-digits number s_0, with n **even**, called **starting value** or **seed**, is given such as:

$$s_0 : \quad d_{n-1}d_{n-2}\cdots d_1 d_0$$

The number is squared obtaining a $2n$-digits number, s'_0, such that:

$$s'_0 = s_0^2 : \quad d'_{2n-1}d'_{2n-2}\cdots d'_1 d'_0$$

If the result has fewer than $2n$ digits, leading zeroes are added to compensate.

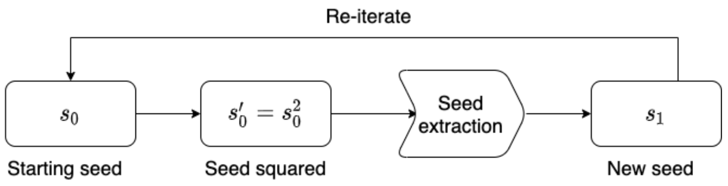
Then, the middle n digits of the squared number s'_0 are taken to form a new number of the sequence, s_1, such that:

$$s_1 = d'_{n+\frac{n}{2}-1}\cdots d'_{n-\frac{n}{2}}$$

Finally, s_1 is used as new *seed* and the process is repeated.

As an example, suppose to take as starting seed: $s_0 = 5913$. Then: $s'_0 = s_0^2 = 34\mathbf{9635}69$ from which the new value $s_1 = 9635$ is extracted and it will be used as new seed for the algorithm.

Figure 1.1: Flow chart of the middle-square method.



A possible Python implementation (the full version is presented in the code section) is the following:

```
seed_0 = int(input("Insert initial value: "))
sequence = set()

n = len(str(seed_0))
end = int(n + n/2)
start = int(n - n/2)

number = seed_0
while number not in sequence:
    sequence.add(number)
    number = int(str(number*number).zfill(2*n)[start:end])
```

The *middle-square method* is in practice no good: its **period** is usually very short, presenting some severe weakness. For a generator of n-digit numbers, the period can be no longer than $8n$. If the middle n digits are all zeroes, the generator then outputs zeroes forever. If the first half of a number in the sequence is zeroes, the subsequent numbers will be converge to zero. While these runs of zero are easy to detect, they occur too frequently for this method to be of practical use. The middle-squared method can also get stuck on a number other than zero.

1.2 Non-Linear Dynamics

In mathematics, a **non-linear system** is a system in which the change of the output is not **proportional** to the change of the input [4]. Non-linear dynamic systems are those systems that describe the change in variables over time. These systems are widely used to describe non-linear problem in biology, sociology, physics, and a lot more, since they may exhibit chaotic behavior and counter-intuitive dynamics, with respect to the linear systems.

There are two main types of dynamical systems: the former uses **differential equations** and the latter **difference equations** (**iterated map**) [5]. Differential equations describe the evolution of systems in continuous-time domain, whereas iterated map arise in problems where time is discrete.

1.2.1 Continuous-Time systems: differential equations

Differential equations were first described in 1671 in the work by **Newton**: "*Methodus fluxionum et Serierum Infinitarum*". **Leibniz** himself derived his form of calculus in 1673. A **non-linear differential equation** is a differential equation that is not a linear equation in the unknown function and its derivatives.

In continuous-time domain, the dynamic of a non-linear system can be described by a set of non-linear differential equations. By adopting (for the sake of simplicity) the Newton 's dot notation about derivatives:

$$\dot{x} = \frac{dx}{dt}$$

it is possible to define the system as:

$$\begin{aligned}
&\dot{x}_1 = f_1(x_1, x_2, \ldots, x_n)\\
&\dot{x}_2 = f_2(x_1, x_2, \ldots, x_n)\\
&\vdots\\
&\dot{x}_n = f_n(x_1, x_2, \ldots, x_n)
\end{aligned}$$

However, the solution of a set of differential equations is not trivial. The fundamental principle of existence, uniqueness, and extendability of solutions for non-linear differential equations is a hard problem and its resolution in special cases is considered to be a significant advance in the mathematical theory.

What matters in this thesis is that non-linear differential equations can exhibit very complicated behavior over extended time intervals, characteristic of **chaos**. The **Lorenz system** presented in this thesis is an example of non-linear differential equations.

1.2.2 Discrete-Time systems: iterated maps

The translations to discrete-time of the non-linear differential equations are called **iterated maps**. In general, a non-linear difference equation is defined as

$$x_{n+1} = f(x_n, x_{n-1}, \ldots)$$

where x_n is the value of x at the iteration n and the **recursion function** f depends on non-linear combinations of its arguments. A solution is a general formula relating x_n to the iteration n and to some specific initial values. However, in relatively few cases, a solution can be analytically obtained when the equation is non-linear [6].

Of particular interest are the **first-order difference equations**, defined as:

$$x_{n+1} = f(x_n)$$

in which the successive state of the system depends only on the current state and not all the previous states.

Some concepts are relevant when talking about iterated maps:

- **Steady-State**: a steady state solution $\bar{x}$ is defined to be the value that satisfies the equation

  $$x_{n+1} = x_n = \bar{x}$$

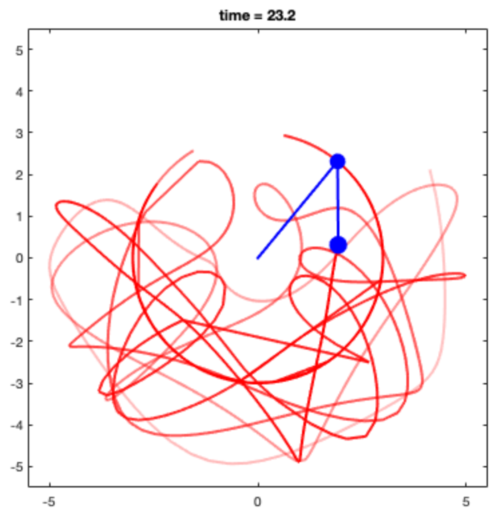
  so that no change occurs from iteration n to iteration $n+1$. The steady-state value is also called **fixed point** of the function f (a value that f leaves unchanged).
- **Stability**: a fixed point is said to be **stable** if neighboring states are attracted to it and unstable if the converse is true. The condition for stability is the following:

  $$\bar{x} \text{ is a stable fixed point of a first-order difference equation iff } \left.\frac{df}{dx}\right|_{\bar{x}<1}$$
- **Critical parameters**: sometimes, stability of a fixed point value is conditional on a parameter μ. If μ is greater or smaller than certain critical values (for instance, 3.5699 for the logistic equation) than the steady-state may or may not exhibits. Such critical parameter values, often called **bifurcation values**, are points of demarcation for abrupt changes in qualitative behavior of the equation or of the system that it models.

1.2.3 Chaotic behavior

"Simple dynamical system do not necessarily lead to simple dynamical behavior"[6]

Figure 1.2: Double pendulum simulation ran with MATLAB.



Dynamical system may lead to **chaotic behavio**r under certain conditions. Indeed, such dynamical systems are said to be strongly dependent on the **initial conditions**: even a slightly perturbation of the current state, not only leads to a complete different output, but also to a complete different dynamics. This behavior exhibits even though these systems are **fully deterministic**, i.e. their future behavior follows a unique evolution and is completely determined by their initial conditions, with no random elements involved.

The theory was first summarized by E. Lorenz and has found applications in various disciplines: from meteorology (for instance: Navier-Stocks equations) to computer science (as it can be found in cryptography); from economics (regarding market dynamics) to sociology, engineering and philosophy.

In order to define the predictability of a non-linear dynamical system, the Russian mathematician **Aleksandr Lyapunov** introduced the concept of **Lyapunov exponent** and **Lyapunov time** (named after him). The Lyapunov time by convention, is defined as the time for the distance between nearby trajectories of the system to increase by a factor of e (being the Nepero's value). Mathematically, it is defined as the inverse of a system's largest Lyapunov exponent. Quantitatively, two trajectories in phase space with initial separation vector $\delta\mathbf{Z}_0$ diverge (provided that the divergence can be treated within the linearized approximation) at a rate given by

$$|\delta\mathbf{Z}| \approx e^{\lambda t}\,|\delta\mathbf{Z}_0|$$

where λ is the Lyapunov exponent.

For discrete-time system (iterated maps) $x_{n+1} = f(x_n)$, for an orbit starting with x_0, this translates to:

$$\lambda(x_0) = \lim_{n\to\infty}\frac{1}{n}\sum_{i=0}^{n-1}\log(|f'(x_i)|)$$

Logistic Chaotic System

2.1 Introduction

In 1976, the biologist **Robert May** published an article on *Nature* called *"Simple mathematical models with very complicated behavior*" [6], in which he proposed an interpretation of particular behaviors that particular systems manifest if tuned properly. First-order differential equations arise in a lot of different context, from biological to economic and social environment, providing an explanation of the the dynamic behavior of several systems: even though simple and deterministic, they can exhibit a surprising array of peculiar situations, such as some "apparently" random fluctuations. **Logistic map** is introduced as the very beginning of *May*'s paper as the discrete-time variation of the continuous-time logistic equation and it is nowadays one of the most commonly used chaotic map in chaotic cryptography for it is one of the simplest and most studied non-linear systems. It has been widely used in block ciphers, stream ciphers and hash functions [1].

2.1.1 The logistic map equation

The logistic map provides a **discrete-time** version of the so-called logistic equation which was proposed by the Belgian mathematician Pierre François Verhulst in 1945 as a model of population growth (and then popularized by R. Pearl and L. Reed), defined as:

$$\frac{dX}{dt} = \mu X \left(1 - \frac{X}{X_{\max}}\right)$$

whose analytic solution is given as:

$$X(t) = \left(1 + \left(\frac{1}{X_0} - 1\right) e^{-\mu t}\right)^{-1}$$

Mathematically, the equation (which has been described by May as *arguably the simplest nonlinear difference equation*) is written as:

$$X_{n+1} = \mu X_n (1 - X_n), \qquad n = 0, 1, 2, 3, \ldots$$

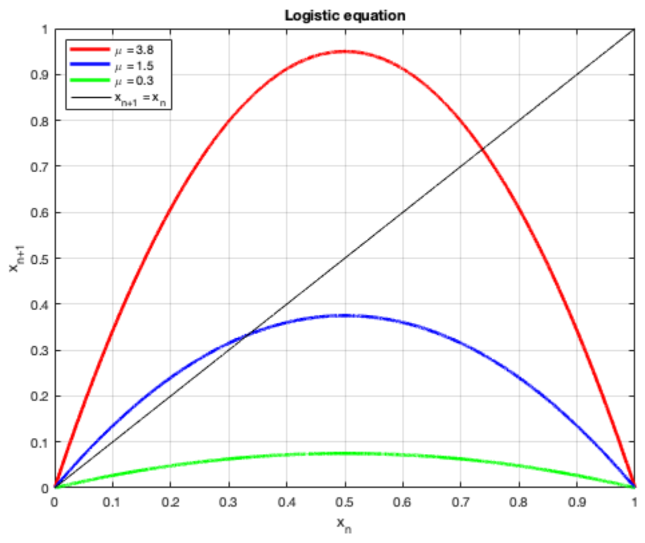
where:

- X_n is a number between 0 and 1 which represents, ideally, the ratio of existing population to the maximum possible population. X_0 represents the **initial condition** of the given system;
- X_{n+1} represents the ratio at the next iteration in time;

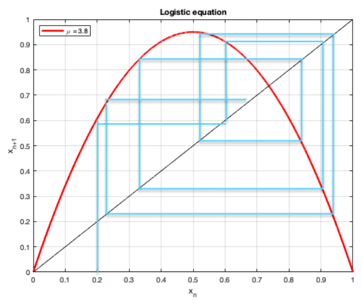
- μ is the **system parameter** of the logistic equation and runs between 0 and 4 in order to keep the ratio in the interested interval.

The parameter μ, which originally indicated the rate of maximum population growth, can be tuned to obtain different equations that leads to different behaviors. The following figures show that feature.

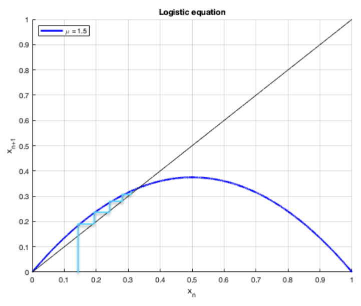
Figure 2.1: Three different logistic equations.



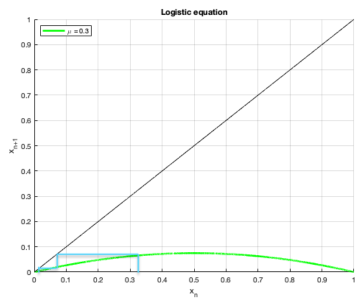
By iterating the state of the equation over the maps, it can be easily seen that different choices of μ may lead to different behavior of the system:



(a) $\mu = 3.8$ (chaos)



(b) $\mu = 1.5$ (determinism)



(c) $\mu = 0.3$ (death)

Figure 2.2: Different behavior of the logistic map when μ is changed.

As already stated, what leads to chaotic behavior is the choice of the system parameter μ:

- $\mu \in [0, 1]$: the output will eventually die (i.e. the state of the system will go to zero), independent of the initial condition.
- $\mu \in [1, 3]$: the output will approach the value $\frac{\mu-1}{\mu}$, independent of the initial condition.
- $\mu \in [3, 1 + \sqrt{6}] \approx [3, 3.4494]$: the output will approach permanent oscillations between two values for almost all initial condition, dependently on the system parameter.

- $\mu \in [3.4494, 3.5699]$: the output oscillates in period of 4, 8, 16, 32 and so on for almost all initial condition, dependently on the system parameter.

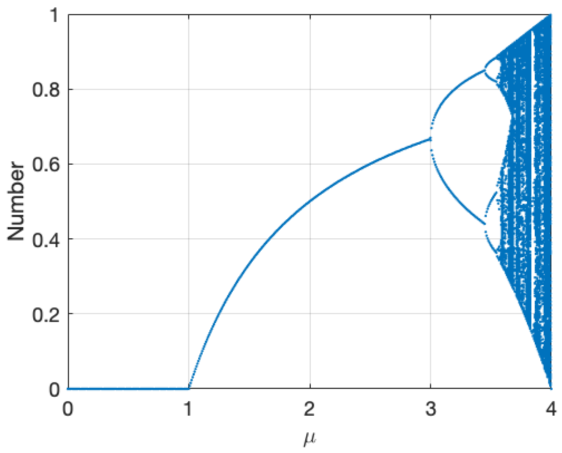
If μ is set to a value in the range

$$\mu \in [3.5699, 4]$$

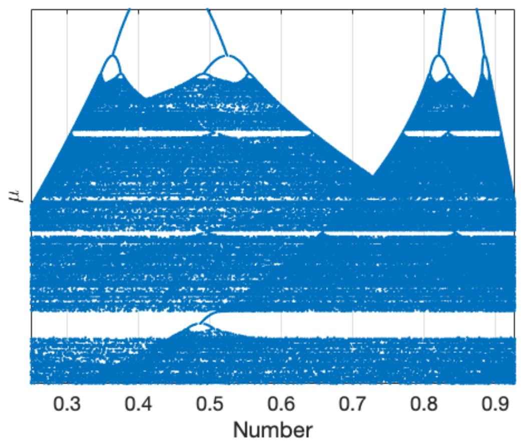
the **Lyapunov exponent** is positive and therefore the system enters into a chaotic state in which slightly modification of the *initial condition* leads to complete different outputs. However, there are certain isolated ranges of μ that show non-chaotic behavior which are called *islands of stability* [6].

The following figures show the evolution of the system for different values of the system parameter μ. From the figure, the **bifurcations** can be easily detected. A bifurcation represents the sudden appearance of a qualitatively different solution for a nonlinear system as some parameter is varied [7]. There are four basic kind of bifurcations:

- **Period-Doubling bifurcation** (**flip**): as it occurs at $\mu = 3$, when the system becomes unstable and a 2-cycle appears;
- **Trans-critical bifurcation**: where a fixed point exists for all values of a parameter and is never destroyed. However, such a fixed point interchanges its stability with another fixed point as the parameter is varied as it occurs at $\mu = 1$, when the previous equilibrium point becomes unstable and a new one appears [5].
- **Pitchfork bifurcation**: where the system transitions from one fixed point to three fixed points.
- **Saddle-Node bifurcation** (**fold**): where two fixed points collide and annihilate each other.



(a) Logistic map



(b) Zoom of the logistic map

Figure 2.3: Representation of the Logistic Map with μ that runs from zero to four.

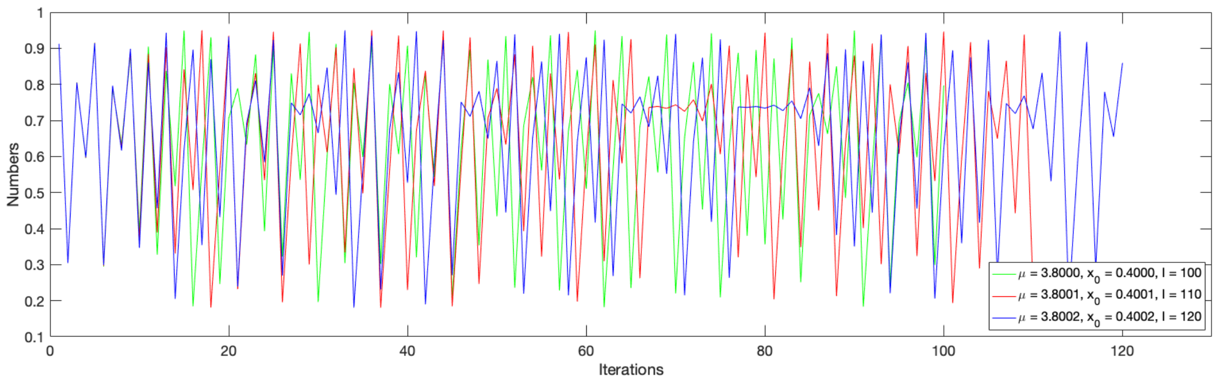
Logistic chaotic system is therefore very sensitive to initial values. By analyzing the results obtained out of the simulation of the logistic map, it is possible to better understand this peculiar characteristic of the system. The following table shows that a very slight change of the initial value can lead to a great difference of the output and the sensitivity of the logistic mapping to the initial value is very significant. The system has been tuned with parameter $\mu = 3.8115$ (arbitrarily chosen) and the results are reported with (only) four decimal digits (for the sake of simplicity). Five different simulations have been run for every different initial value ($I = 1$ to $I = 1000$):

Table 2.1: Logistic mapping values with $\mu = 3.8115$.

Initial value	Iterations				
	$I = 1$	$I = 2$	$I = 50$	$I = 100$	$I = 1000$
$x_0 = 0.4000$	0.9148	0.2972	0.9146	0.3550	0.2678
$x_0 = 0.4001$	0.9148	0.2970	0.3333	0.6754	0.8373
$x_0 = 0.4002$	0.9149	0.2967	0.6093	0.8433	0.6352

Under the different system parameter μ, initial value X_0 and the iteration numbers I, also the transient behaviors (i.e. the various states the system assume) greatly differ from each other. The following picture highlights this behavior.

Figure 2.4: Dynamic behavior under different seed parameters.



2.1.2 Computation of the Lyapunov exponent

The **Lyapunov exponent** of a dynamical system is a quantity that characterizes the rate of separation of infinitesimally close trajectories. It is, in other words, an indicator of chaos. It is defined, in one-dimension as following:

consider two points x_0 and $x_0 + \varepsilon$, mapped by the function

$$f : I \to I$$

where $I \in \mathbb{R}$ is some bounded interval on the real axis. For n iterations of this map, the Lyapunov exponent $\lambda(x_0)$ satisfies the equations

$$|\varepsilon| e^{n\lambda(x_0)} = |f^n(x_0 + \varepsilon) - f^n(x_0)|$$

Dividing by ε and taking the limit as $\varepsilon \to 0$:

$$\begin{aligned}\lim_{\varepsilon \to 0} e^{n\lambda(x_0)} &= \lim_{\varepsilon \to 0} \frac{|f^n(x_0 + \varepsilon) - f^n(x_0)|}{|\varepsilon|} \\ &= \left.\frac{df^n(x)}{dx}\right|_{x_0}\end{aligned}$$

Then, taking the limit as $n \to \infty$, the definition of the Lyapunov exponent is the following:

$$\lambda(x_0) = \lim_{n\to\infty} \frac{1}{n} \log \left(\left. \frac{df^n(x)}{dx} \right|_{x_0} \right)$$

Moreover, since

$$f^n(x_0) = f(f(\cdots(f(x_0))\cdots))$$

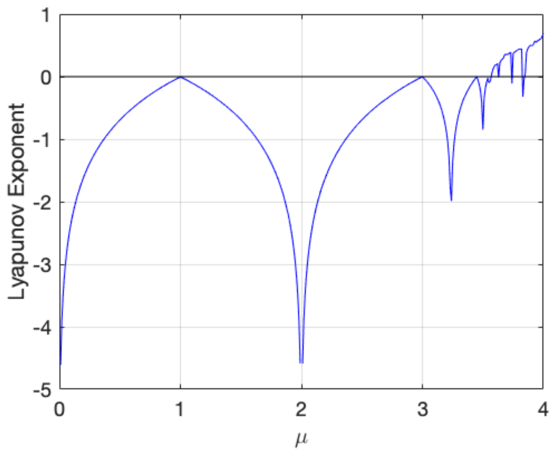
themn

$$\left. \frac{df^n(x)}{dx} \right|_{x_0} = \left. \frac{df(x)}{dx} \right|_{x_{n-1}} \cdots \left. \frac{df(x)}{dx} \right|_{x_1} \left. \frac{df(x)}{dx} \right|_{x_0} = \\ = f'(x_{n-1}) \cdots f'(x_1) f'(x_0)$$

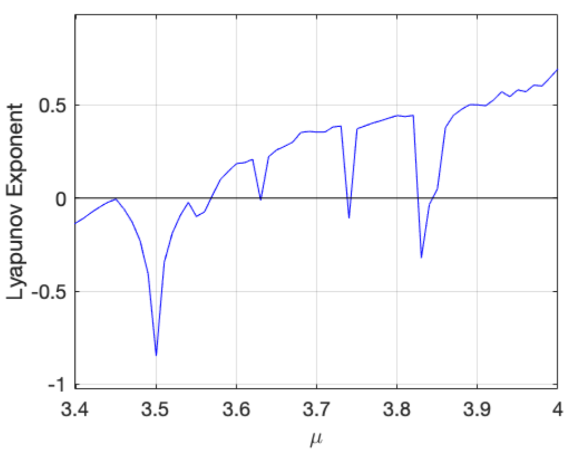
and finally

$$\lambda(x_0) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \log |f'(x_i)|$$

The following pictures represent the Lyapunov exponent of the logistic map when μ runs from 0 to 4. Notice that, for $\mu = 3.5669$ the exponent becomes positive, leading to chaotic behavior.



(a) Lyapunov exponent



(b) Zoom of the Lyapunov exponent

Figure 2.5: Diagram of the Lyapunov exponent with μ that runs from zero to four.

2.2 Algorithm

A **Pseudo-Random Number Generator** (**PRNG**) is defined as an algorithm that can produce random number sequences whose main advantage is the rapidity and repeatability of a generating process. The Logistic map has been exploited by **L. Wang** and **H. Cheng** in 2019 in the article called *Pseudo-Random Number Generator Based on Logistic Chaotic System.* "In order to improve the complexity and randomness of modified PRNGs, the chaotic system parameter denoted by floating point numbers generated by the chaotic system is confused and rearranged to increase its key space and reduce the possibility of an exhaustive attack" [1].

The algorithm is divided into two states:

- **Initial state**: the **seed** parameters need to be tuned by hand;
- **Normal state**: the **seed** parameters are automatically tuned based on the generation of antecedent pseudo-random sequences.

In the following section the algorithm is reproduced in the **C language** and the results obtained are analyzed in the **MATLAB environment**.

2.2.1 Initial State

The system is defined by three parameters which, collectively, formed the **seed** S of the system. The initial seed S_1 is therefore defined as

$$S_1 = (\mu_1, x_{0_1}, I_1)$$

where:

- $\mu_1 \in [3.5699, 4]$ is the initial system parameter;
- $x_{0_1} \in [0.0, 0.5]$ is the initial value of the initial system;
- $I_1 > 1000$ is the initial number of iterations.

According to the logistic equation, it is possible to obtain a **16 floating point** number y_1 by iterating for I_1 times. Discarding the integer part of y_1, the 15-decimal number is re-arranged in order to tune the system for the generation of the next number in the sequence.

The seed structure is defined as follow:

```
typedef struct seed_s {
    double system_param;
    double initial_value;
    int N_iterations;
} seed_t;
```

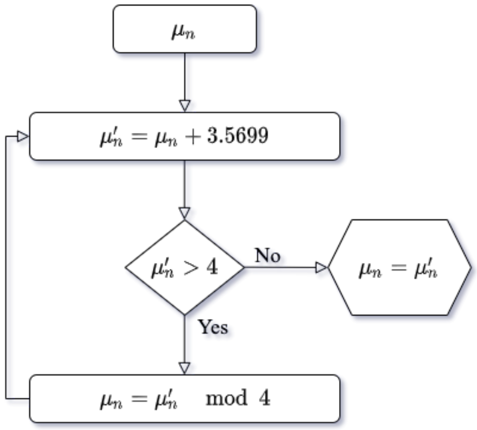
2.2.2 Normal State

After the generation of the output y_{n-1}, generated at the $(n-1)$th iterations, retaining the fractional part of the floating point number it is possible to rearrange it in the following way:

- The 1st, 5th, 9th and 13th numbers are taken to form the decimal part with the integer part equals to 0 as the initial value x_{0_n} of the new chaotic system;
- The 2nd, 6th, 10th and 14th digits are taken as the hypothetically new system parameter μ_n of the new chaotic system. The flow chart of the generation of μ_n is shown in the following figure, along with the snippet of C code used in the simulation:

```
double generate_new_param(double old_param) {
    double param_tmp = old_param + (double)3.5699;
    while (param_tmp > 4) {
        old_param = fmod(param_tmp, (double)4));
        param_tmp = old_param + (double)3.5699;
    }
    return param_tmp;
}
```

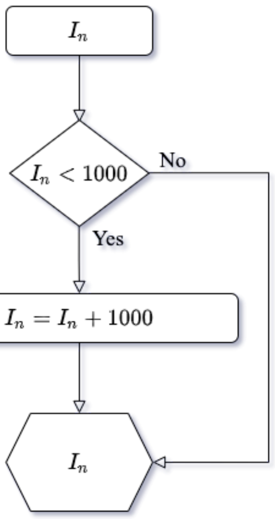
Figure 2.6: Flow chart of the system parameter generation.



- The 3rd, 7th, 11th and 15th digits are taken as the hypothetically new system iterations I_n of the new chaotic system. The flow chart of the generation of I_n is shown in the following figure, along with the snippet of C code used in the simulation:

```
int generate_new_iteration(int old_iteration) {
    if (old_iteration < 1000) {
        return old_iteration + 1000;
    }
    return old_iteration;
}
```

Figure 2.7: Flow chart of the system iteration generation.



2.2.3 random_generator(seed)

In order to provide a correct behavior of the system architecture when working with decimal numbers, a certain precision with floating-points value is required. The following directive provides the architecture information in order not to lose any precision during the operations in the algorithm:

```
#define FILENAME "output_sequence.txt"
#define DIGITS 16
#undef  FLT_ROUNDS
#undef  DBL_DIG
#undef  DBL_EPSILON
#define FLT_ROUNDS -1
#define DBL_DIG 17
#define DBL_EPSILON 1E-17
```

The algorithm used to generate only one output (i.e. only the output of the first chaotic system, before the re-arrangement of the seed parameter) is implemented in **C-code** language as follows. The rest of the algorithm can be found in the CODE SECTION:

```
double random_generator(seed_t seed) {

    double old_value = seed.initial_value;
    double new_value = 0.0;
    for (int i = 0; i < seed.iterations; i++) {
        new_value = seed.system_parameter*old_value*(1 - old_value);
        old_value = new_value;
    }
    return new_value;
}
```

2.3 Results

The results of the two algorithms have been analyzed using the test suite **NIST SP 800-22** (Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, 2014). The suite can be downloaded here: link to download. The results of the test are shown in this section.

The **statistical analysis** is run on MATLAB. The **security analysis** is taken from the article [1].

2.3.1 Statistical Analysis

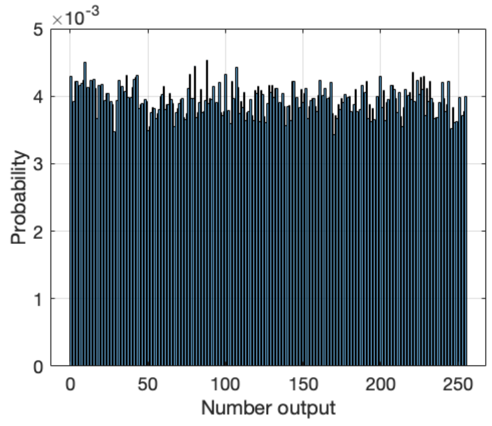
The algorithm has been run under two different initial conditions.
The first seed is the following:

$$\begin{cases} \mu = 3.8000 \\ x_0 = 0.5000 \\ I = 1000 \end{cases}$$

The second seed is the following:

$$\begin{cases} \mu = 3.7166 \\ x_0 = 0.1674 \\ I = 7348 \end{cases}$$

The distribution set is plotted in the range $(0, 256]$ and the results are the following:


(a) Distribution with the first seed choice.

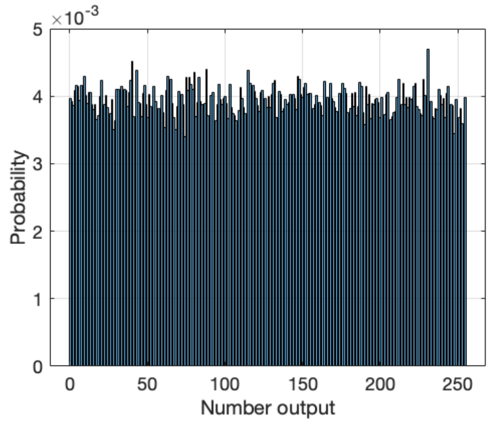
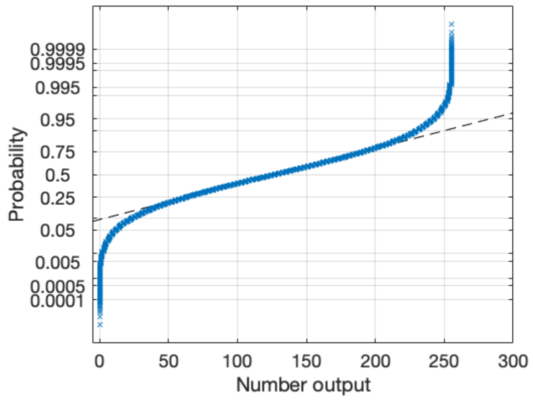

(b) Distribution with the second seed choice.

Figure 2.8: Simulation of the Logistic Map.


(a) Distribution with the first seed choice.

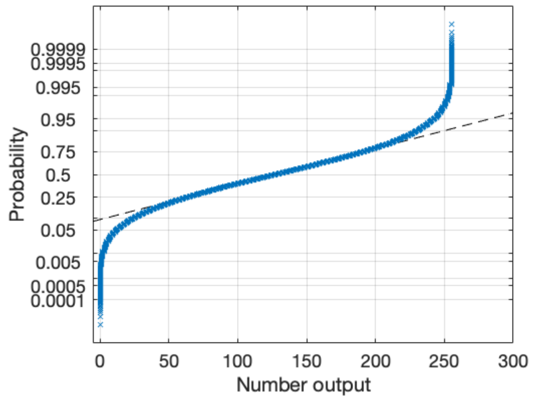

(b) Distribution with the second seed choice.

Figure 2.9: Simulation of the Logistic Map.

As expected, the data distribution is almost uniform in the range, which demonstrate the effectiveness of the algorithm to generate almost true random sequence.

2.3.2 Security Analysis

In order to ensure the security of the cryptosystem, a good cryptosystem must be sensitive to the keys. Let p be the difference between the previous output number and the current output number. Then, let q be the length of the sequence: the ratio

$$r = \frac{difference\_output}{sequence\_length} = \frac{p}{q}$$

is calculated. It can be concluded from the data in the table that the generated sequence is sensitive to seed parameters.

$(\mu, x_0, I)_1$	$(\mu, x_0, I)_2$	$r/\%$
$(3.8001, 0.4000, 1000)$	$(3.8002, 0.4001, 2000)$	51.9231
$(3.8002, 0.4001, 2000)$	$(3.8003, 0.4002, 1000)$	42.3077
$(3.8003, 0.4003, 1000)$	$(3.8004, 0.4003, 1000)$	51.9231
$(3.8004, 0.4003, 1000)$	$(3.8004, 0.4004, 1000)$	43.2692
$(3.8004, 0.40041000)$	$(3.8004, 0.4004, 1000)$	44.2308

A necessary condition for an encryption scheme to be secure is that the key space is large enough so as to frustrate brute-force attacks. The analysis can be performed by taking into account the following parameters:

- Initial value: it has been shown in [8] that a difference of the order of 10^{-30} in the initial value leads to different values after only 99 iterations. Thus, for this sensitivity order it is possible to have 10^{30} possible initial values between 0 and 1. Therefore, increasing the number of decimal places to be supported results in increasing the key space of the desired system and thus increasing its safety.
- Iterations: in order to improve security, it is necessity to improve the key space; thus the number of iterations of each chaotic system is guaranteed to be 1000 at least;
- Parameter extraction: according to the algorithm, the probability of the generation of the integer value sequence is 3.55×10^{12}, the integer is transformed into binary sequence after module 256. When normalizing, the sequence of 15 integer values obtained last time is rearranged, and there are 1.3×10^{12} possibilities. The process has a total of $1.3 \times 10^{12} \times 3.55 \times 10^{12} = 4.615 \times 10^{24}$ possibilities, and the second time has the potential of $\left(4.615 \times 10^{24}\right)^2 > 2^{128}$. On this basis, the seed parameters (μ, x_0, I) of the next chaotic system are extracted. It not only expands the key space, but also realizes the independence of pseudo-random sequences.

2.3.3 Performance Analysis

The following table shows the results of the **NIST test** for the first algorithm, after the conversion of the integer output sequence to the **binary domain**.

Test index	**p value**	**Result**
Frequency	0.556298	SUCCESS
Block Frequency	0.578344	SUCCESS
Cumulative Sums	0.691934	SUCCESS
Runs	0.510265	SUCCESS
Longest Run	0.084999	SUCCESS
Rank	0.670342	SUCCESS
Non-overlapping Template	0.457732	SUCCESS
FFT	0.402675	SUCCESS
Overlapping template matching	0.210308	SUCCESS
Universal	0.784275	SUCCESS
Approximate Entropy	0.287458	SUCCESS
Random Excursions	0.347548	SUCCESS
Random Excursions Variant	0.219526	SUCCESS
Serial	0.512756	SUCCESS
Linear Complexity	0.651363	SUCCESS

Generalized Lorenz System

3.1 Introduction

In 1963, **Edward Lorenz** developed a simplified mathematical model for atmospheric convection. He was trying to simulate the atmospheric condition evolution over time with a set of 14 equations and 14 parameters (such as: temperature, pressure, humidity, etc...) input into a numerical calculator, when he noticed that the outputs of the simulation were different, even when the initial conditions were the same. Due to rounding errors, the system went into chaotic state even if the equations were completely deterministic. In his article "*Deterministic Nonperiodic Flow*" [9], he introduced a **system of three ordinary differential equations**, simplified from the original 14 equations studied, now known as the **Lorenz equations**.

3.1.1 The Lorenz equations

The Lorenz dynamical system is a set of three differential equations described as:

$$\dot{s} = f(t, s, p)$$

which expands to:

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x) \\[2ex] \dfrac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y \\[2ex] \dfrac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z \end{cases}$$

where:

$$\dot{s} = \left[\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt}\right]$$

is the state vector containing the three-dimensional coordinates of the system with respect to time t and

$$p = [\sigma, \rho, \beta] > 0$$

is the vector containing the system parameters, assumed to be positive.

When ρ is set to be smaller than 1, then there is only one equilibrium point, which is at the origin. This point corresponds to no convection. All orbits converge to the origin, which is a **global attractor**. At $\rho = 1$, a **pitchfork bifurcation** occurs and two additional critical point appear at:

$$(\sqrt{\beta(\rho - 1)}, \sqrt{\beta(\rho - 1)}, \rho - 1)$$

and

$$(-\sqrt{\beta(\rho-1)}, -\sqrt{\beta(\rho-1)}, \rho-1)$$

which correspond to steady convection. This pair of equilibrium points is stable only if

$$\rho < \sigma\frac{\sigma+\beta+3}{\sigma-\beta-1}$$

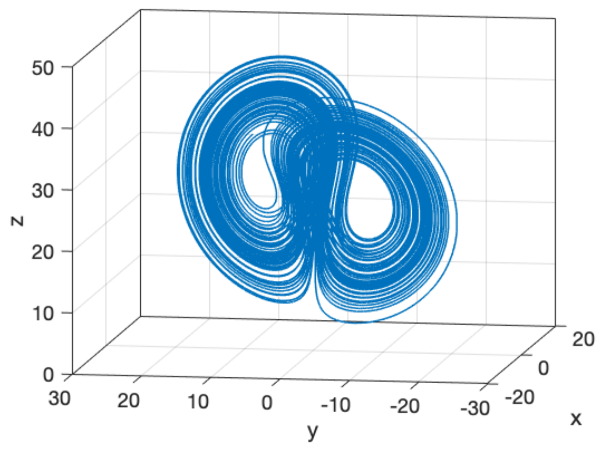
which can hold only for positive ρ if $\sigma > \beta+1$. At the critical value, both equilibrium points lose stability through a subcritical **Hopf bifurcation** [10].

3.1.2 Chaotic state of the Lorenz system

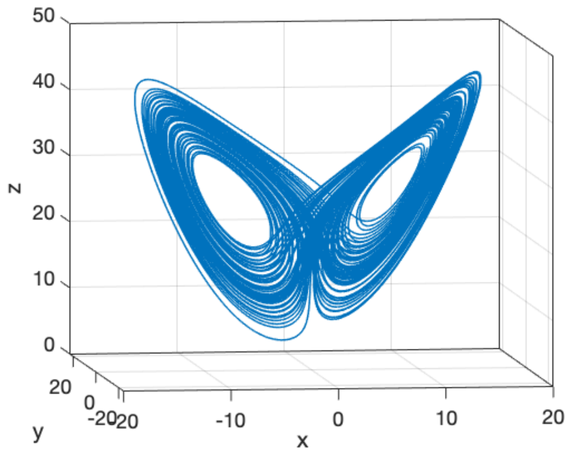
In the original work, Lorenz used the following set of values to tune the system into chaotic state:

$$\sigma = 10$$
$$\rho = 28$$
$$\beta = 8/3$$

Even if from a technical standpoint, the Lorenz system is nonlinear, non-periodic, three-dimensional and **deterministic**, it exhibits chaotic behavior for these (and nearby) values. Almost all initial points will tend to an invariant set (the Lorenz attractor) a strange attractor, a fractal, and a self-excited attractor with respect to all three equilibria.



(a) Lorenz chaotic system (y view)



(b) Lorenz chaotic system (x view)

Figure 3.1: Simulation of the Lorenz chaotic system.

3.1.3 Computation of the Lyapunov exponent

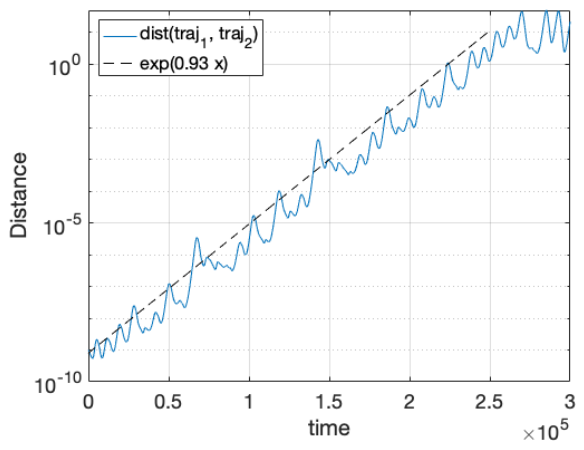
For a continuous-time dynamical system, the **maximal Lyapunov exponent** is defined as follows:

Consider a trajectory $x(t)$, $\quad t \geq 0$ in phase space and a nearby trajectory $x(t)+\delta(t)$ where $\delta(t)$ is a vector with infinitesimal initial length. The maximal Lyapunov exponent of the system is the number λ, if it exists, such that

$$|\delta(t)| \approx |\delta(0)|e^{\lambda t}$$

However, dynamical systems don't just have a single Lyapunov exponent. Rather, every dynamical system has a spectrum of Lyapunov exponents, one for each dimension of its phase space. Like the largest eigenvalue of a matrix, the largest Lyapunov exponent is responsible for the dominant behavior of a system. Therefore the use of "**maximal**" is justified.

Figure 3.2: Lyapunov exponent as distance between two trajectory.

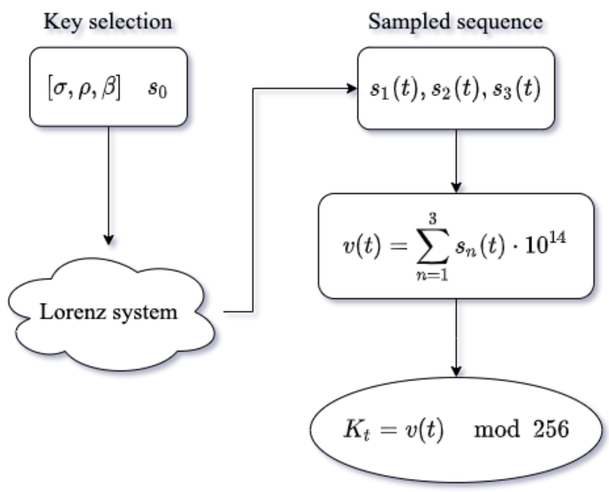


3.2 Algorithm

As already stated, a **Pseudo-Random Number Generator** (**PRNG**) is defined as an algorithm that can produce random number sequences whose main advantage is the rapidity and repeatability of a generating process. The Lorenz chaotic system has been exploited by **V. Lynnyk**, **N. Sakamoto** and **S. Čelikovsky** in 2015 in the article called *Pseudo-Random Number Generator Based on the generalized Lorenz chaotic system.* [2].

The flow chart of the first implementation is presented here:

Figure 3.3: Flow chart of the PRNG Lorenz algorithm.



In the following section, the two algorithms proposed in the article are implemented, run and analyzed in **MATLAB**. In particular, MATLAB-SIMULINK ODE45() RUNGE-KUTTA PROCEDURE is used instead of the ODE4() PROCEDURE implemented in the article. Both of the algorithms use a combination of the three coordinates of the chaotic orbit. The first 100 pseudo-random number generated by PRNG are not taken into account due to the **transient state** of the chaotic system.

3.2.1 First implementation

The first algorithm is implemented by the following equations:

$$v(t) = \sum_{i=1}^{3} s_i(t) \cdot 10^{14}$$

$$k_t = v(t) \mod 256$$

where $s_1(t)$, $s_2(j)$ and $s_3(j)$ are the samples of the generalized Lorenz system at the iteration t; K_t is the one dimensional integer sequence of the generated numbers, such that:

$$K_t \in (0, 256]$$

The MATLAB implementation of the algorithm is the following. Notice that SIGMA, RHO, BETA, X0, Y0, Z0 are arbitrary selected in inputs. N is the length of the sequence, chosen arbitrary as input.

```
%% DEFINING DATA STRUCTURE
% ------------------------------------------------------------------------
offset = 99;                          % Delete the transient part
N = N + offset;                       % Due to transient removal
p = [sigma; rho; beta];               % [sigma, rho, beta]
s0 = [x0; y0; z0];                    % Initial state vector
dt = 0.0001;                          % Time step
t_start = dt;                         % Time start
t_end = N*dt;                         % Time end
tspan = t_start:dt:t_end;             % Time interval


%% RUNNING ODE SIMULATION
% ------------------------------------------------------------------------
options = odeset('RelTol', 1e-12, 'AbsTol', 1e-12*ones(1, 3));
[t, s] = ode45(@(t, x)lorenz(t, x, p), tspan, s0, options);

%% GENERATION OF NUMBER SEQUENCE
% ------------------------------------------------------------------------
v = zeros(1, length(s) - offset);
k = zeros(1, length(s) - offset);
for i = 100:length(s)
    v(i-offset) = sum(s(i, :)) * 1e14;
    k(i-offset) = mod(v(i-offset), 256);
end
```

3.2.2 Second implementation

The second implementation of the algorithm is given by the following set of equation:

$$\begin{cases} w(3t) = s_1(t) \cdot 10^{14} \\ w(3t+1) = s_2(t) \cdot 10^{14} \\ w(3t+2) = s_3(t) \cdot 10^{14} \end{cases}$$

where $s_1(t)$, $s_2(j)$ and $s_3(j)$ are the samples of the generalized Lorenz system at the iteration t. The result is spanned in the interval $(0, 256]$ as before:

$$Q_t = w(t) \mod 256$$

The MATLAB implementation of the algorithm is the following. Notice that SIGMA, RHO, BETA, X0, Y0, Z0 are arbitrary selected in inputs. N is the length of the sequence, chosen arbitrary as input.

```
%% DEFINING DATA STRUCTURE
% ------------------------------------------------------------------------
offset = 99;                          % Delete the transient part
N = ceil(N/3) + offset;               % Due to transient removal
p = [sigma; rho; beta];               % [sigma, rho, beta]
s0 = [x0; y0; z0];                    % Initial state vector
dt = 0.0001;                          % Time step
t_start = dt;                         % Time start
t_end = N*dt;                         % Time end
tspan = t_start:dt:t_end;             % Time interval


%% RUNNING ODE SIMULATION
% ------------------------------------------------------------------------
options = odeset('RelTol', 1e-12, 'AbsTol', 1e-12*ones(1, 3));
[t, s] = ode45(@(t, x)lorenz(t, x, p), tspan, s0, options);

%% GENERATION OF NUMBER SEQUENCE
% ------------------------------------------------------------------------
w = zeros(1, length(s) - offset);
q = zeros(1, length(s) - offset);
for i = 100:length(s)
    w(3*(i-offset) - 2) = s(i, 1) * 1e14;
    w(3*(i-offset) - 1) = s(i, 2) * 1e14;
    w(3*(i-offset)) = s(i, 3) * 1e14;
    q(3*(i-offset) - 2) = mod(w(3*(i-offset) - 2), 256);
    q(3*(i-offset) - 1) = mod(w(3*(i-offset) - 1), 256);
    q(3*(i-offset)) = mod(w(3*(i-offset)), 256);
end
```

3.3 Results

The results of the two algorithms have been analyzed using the test suite **NIST SP 800-22** (Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, 2014). The suite can be downloaded here: link to download. The results of the test are shown in this section.

The **statistical analysis** is run on MATLAB. The **security analysis** is taken from the article [2].

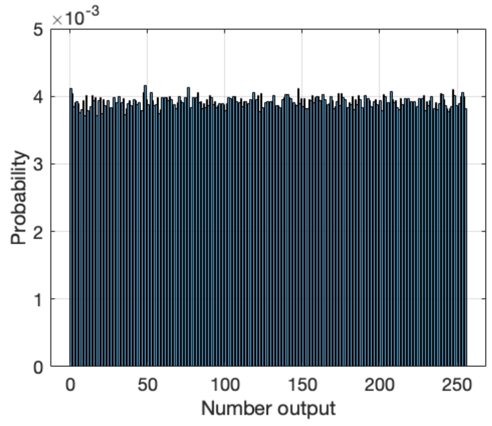
3.3.1 Statistical Analysis

The two algorithm have been run under the following conditions:

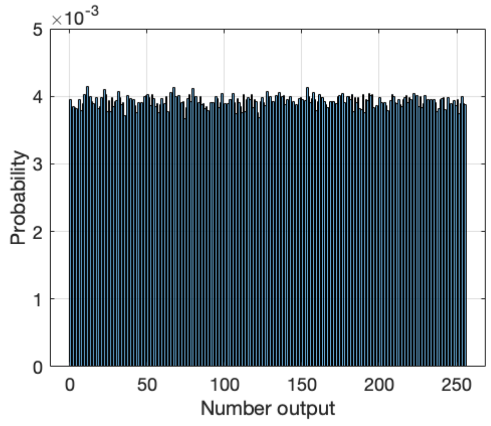
$$s_0 = \begin{cases} x_0 = 1.012417782395261 \\ y_0 = 0.022341888942009 \\ z_0 = 6.010786276501001 \end{cases}$$

$$p = \begin{cases} \sigma = 10 \\ \rho = 28 \\ \beta = \frac{8}{3} \end{cases}$$

The number of values in the output sequence is $N = 500000$. The distribution set is plotted in the range $(0, 256]$ and the results are the following:

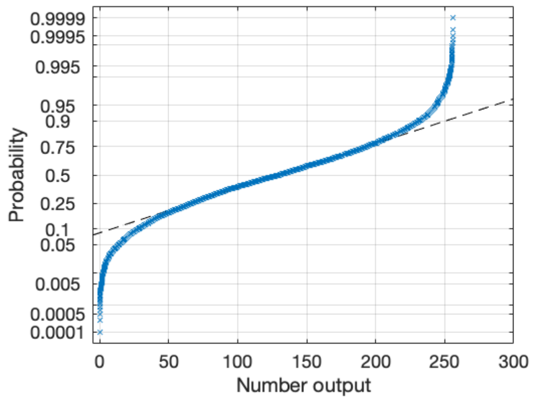


(a) Distribution of the first implementation

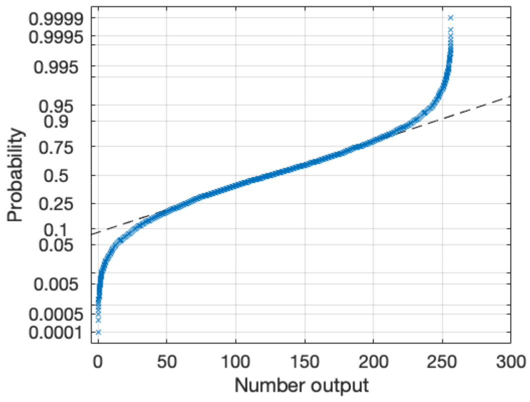


(b) Distribution of the second implementation

Figure 3.4: Simulation of the Lorenz chaotic system.



(a) Distribution of the first implementation



(b) Distribution of the second implementation

Figure 3.5: Simulation of the Lorenz chaotic system.

As expected, the data distribution is almost uniform in the range, which demonstrate the effectiveness of the algorithm to generate almost true random sequence.

3.3.2 Performance Analysis

The following table shows the results of the **NIST test** for the first algorithm, after the conversion of the integer output sequence to the **binary domain**.

Test index	**p value**	**Result**
Frequency	0.467717	SUCCESS
Block frequency	0.270637	SUCCESS
Comulative sums	0.414979, 0.738513	SUCCESS
Runs	0.814540	SUCCESS
Longest run	0.882674	SUCCESS
Rank	0.439098	SUCCESS
Non-overlapping template	0.010412 . . .	SUCCESS
FFT	0.827409	SUCCESS
Overlapping template matching	0.206259	SUCCESS
Universal	0.853922	SUCCESS
Approximate entropy	0.831525	SUCCESS
Random excursions	0.187714 . . .	SUCCESS
Random excursion variant	0.108153 . . .	SUCCESS
Serial	0.020636, 0.051270	SUCCESS
Linear complexity	0.841081	SUCCESS

The following table shows the results of the **NIST test** for the second algorithm, after the conversion of the integer output sequence to the **binary domain**.

Test index	**p value**	**Result**
Frequency	0.411607	SUCCESS
Block frequency	0.459375	SUCCESS
Comulative sums	0.305161, 0.662097	SUCCESS
Runs	0.071376	SUCCESS
Longest run	0.561224	SUCCESS
Rank	0.147370	SUCCESS
Non-overlapping template	0.012808 . . .	SUCCESS
FFT	0.237078	SUCCESS
Overlapping template matching	0.616101	SUCCESS
Universal	0.622942	SUCCESS
Approximate entropy	0.130159	SUCCESS
Random excursions	0.192581 . . .	SUCCESS
Random excursion variant	0.103418 . . .	SUCCESS
Serial	0.823813, 0.947866	SUCCESS
Linear complexity	0.084808	SUCCESS

3.3.3 Security Analysis

The algorithm is run with two slightly different initial condition and it is shown that the two dataset obtained as output sequence are not correlated to each other. Retaining the same condition used during the statistical analysis, the algorithm is run first with

$$x_{0_1} = 1.012417782395261$$

and the with

$$x_{0_2} = 1.012417782395266$$

The correlation coefficient obtained is:

$$0.0003$$

This experiment shows that the PRNG based on the generalized Lorenz chaotic system is extremely sensitive to the key. This feature is extremely relevant in the security context. Being able to decrypt the output sequence to obtain the initial key leads to extremely poor quality of the system that uses the algorithm. An almost null correlation between the keys, even with a difference of 1×10^{-16} decimal places, results in a sufficient and efficient procedure in encryption of data.

Having (very) little statistical correlation between two generated sequences means that no statistical characteristics can be easily inferred. A potential malicious attack may happens by means of

- Attack the initial values parameter: for this purpose, a sufficiently large key space is guaranteed in the proposed PRNG for practical applications under this condition. Indeed, three initial coordinates need to be chosen with a magnitude of precision of 1×10^{-16} decimal places.
- Attack the sequence generated by the chaotic system: as already stated, the sequence generated by the chaotic system has no statistical properties which means it cannot be attacked.

Conclusions

The deep correlation between chaos and cryptography has led to several and different models of procedure to generate random numbers. The aim of this essay has been to reproduce and simulate two algorithms for generating pseudo-random sequences based on non-linear dynamical systems: the Lorenz System and the Logistic map. Several papers have been published over the course of the years: this essay took into account, in particular, the work by **L. Wang** and **H. Chang**: "*Pseudo-Random Number Generator Based on Logistic Chaotic System*"; and the work by **V. Lynnyk**, **N. Sakamoto** and **S. Čelikovsky**: *Pseudo-Random Number Generator Based on the generalized Lorenz chaotic system.*

First, an introduction to non-linear dynamical systems have been provided, showing that such systems may lead to chaotic behavior. Differential equations in the continuous-time domain and Difference equations in the discrete-time domain have been introduced. Then, a general understanding of Pseudo-Random Number Generator has been given in the first chapter. In the second and third chapters, a preliminary part to the Lorenz system and the Logistic Map has been presented, focusing on the visualization of such systems and the computation of the Lyapunov exponents. As for the computer engineering part, the algorithms have been written and published on *GitHub* and the results obtained have been analyzed on the MATLAB ENVIRONMENT. Finally, statistical and security tests have been performed on the data obtained, showing that such procedure are cryptographically secure, even though those are not the state-of-the-art for any crypto-systems.

4.1 Code Section

All the codes used to simulate the chaotic systems in this thesis and to generate the diagrams can be found at the following link:

github/DavideArcolini

All the flow charts have been realized using the online framework at the following link:

app.diagrams

References

[1] Luyao Wang and Hai Cheng. "Pseudo-Random Number Generator Based on Logistic Chaotic System". In: *Entropy* 21.10 (2019). ISSN: 1099-4300. DOI: 10.3390/e21100960. URL: https://www.mdpi.com/1099-4300/21/10/960.

[2] Noboru Sakamoto Volodymyr Lynnyk and Sergej Čelikovský. "Pseudo random number generator based on the generalized Lorenz chaotic system". In: *IFAC-PapersOnLine* 48.18 (2015). 4th IFAC Conference on Analysis and Control of Chaotic Systems CHAOS 2015, pp. 257–261. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.11.046. URL: https://www.sciencedirect.com/science/article/pii/S2405896315023046.

[3] John von Neumann. "Various Techniques Used in Connection with Random Digits". In: *Monte Carlo Method*. Ed. by A. S. Householder, G. E. Forsythe, and H. H. Germond. Vol. 12. National Bureau of Standards Applied Mathematics Series. Washington, DC: US Government Printing Office, 1951. Chap. 13, pp. 36–38.

[4] Larry Hardesty. *Explained: Linear and nonlinear systems*. Feb. 2010. URL: https://news.mit.edu/2010/explained-linear-0226.

[5] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*. Westview Press, 2000.

[6] Robert May. "Simple Mathematical Models With Very Complicated Dynamics". In: *Nature* 26 (July 1976), p. 457. DOI: 10.1038/261459a0.

[7] Henri Poincaré. "Sur l'équilibre d'une masse fluide animée d'un mouvement de rotation." In: *Acta mathematica* 7 (Sept. 1885), pp. 259–380. URL: http://henripoincarepapers.univ-lorraine.fr/chp/hp-pdf/hp1885ama.pdf.

[8] D. Bose and Amitabh Banerjee. "IMPLEMENTING SYMMETRIC CRYPTOGRAPHY USING CHAOS FUNCTIONS". In: 2002.

[9] E. Lorenz. "Deterministic Nonperiodic Flow". In: *Journal of Atmospheric Sciences* 20.2 (1963), pp. 130–141. DOI: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2. URL: https://journals.ametsoc.org/view/journals/atsc/20/2/1520-0469_1963_020_0130_dnf_2_0_co_2.xml.

[10] Hirsch Morris W., Smale Stephen, and Devaney Robert. *Differential Equations, Dynamical Systems, An Introduction to Chaos (Second ed.)* Academic Press, 2003.