Design & Implementation of Quantum Computing Immune Cryptography Processor

A thesis submitted in fulfillment of
the requirements for the degree of

Bachelor of Technology

by

Abhishek Agrawal (**150102002**)
Souradip Pal (**150102076**)

Under the guidance of
Dr. Gaurav Trivedi

Department of Electronics & Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
April 2019

Abstract

Recent advancements in the domain of Quantum Computing are posing a security threat to most of the classical cryptographic methods and algorithms. Most popular symmetric and asymmetric cryptosystems including RSA, ECC, DES, Diffie-Hellman etc. can be efficiently broken down by a quantum computer running Shor's Algorithm. There exist two methods to counter this threat: firstly using **Quantum Cryptography** which involves setting up a quantum channel, quantum computers and would require a complete overhaul of the data communication infrastructure. Second alternative called **Post-Quantum Cryptography**, can however be implemented on the existing infrastructure and is equally resistant to classical and quantum crypto-attacks. Post-Quantum Cryptographic algorithms are more immune to these attacks because they are designed on the weakness of quantum circuits and algorithms. Hash, Code, Lattice based and Multivariate Polynomial based cryptographic algorithms are the most popular class of such algorithms. *In this report, we present the design and hardware implementation of a chaos-based post-quantum cryptographic system derived from a mechanical model showing non-linear dynamics and show its resistance against quantum computing attacks.* We also provide in this report the details of the encryption-decryption algorithms and experimentations performed on the implementation of the system in *Artix FPGA* hardware platform.

Contents

List of Figures

Nomenclature

FPGA	Field Programmable Gate Array
DSA	Digital Signature Algorithm
RSA	Rivest-Shamir-Adleman
DES	Data Encryption Standard
PQC	Post-Quantum Cryptography
ECC	Elliptic Curve Cryptography
NTRU	Nth-Degree Truncated Polynomial Ring
ODE	Ordinary Differential Equation
CLB	Configurable Logic Block
ALU	Arithmetic Logic Unit
LUT	Look-Up Table
USB	Universal Serial Bus
FSM	Finte State Machine
HDL	Hardware Description Language
RTL	Resistor Transfer Level

Chapter 1

Introduction

1.1 Motivation

In this modern era, while transferring data from one computer to another, we almost everytime assume a secure connection. However, in a situation where this security is broken, the effects can be devastating and almost all the businesses and services relying on internet including banking and cloud services would be rendered useless. Such a situation may not be too distant in future. Advancements in quantum computing have posed a threat to the existing cryptographic methods on which the complete data communication infrastructure relies on, including the most popular public-key cryptography.

In 1994, a mathematician named Peter Shor, developed a quantum computing based algorithm which is able to break the security of key exchanges and digital signatures. Using this algorithm, a quantum computer would be able to crack the most sophisticated encryption in a matter of minutes. Quantum computers operate differently from traditional computing devices i.e. they work at atomic level. The essential units of a quantum computer is a qubit in contrast to bits used in traditional computers. A qubit is able to represent 0 and 1 simultaneously. Due to this property, few qubits can speed up certain types of computation by an enormous amount and hence quantum computers are more suitable for brute-force exhaustive searches. Although, this new technology is ideal for solving complex problems in astrophysics, weather forecasting and pharmaceuticals, it can also break the encryption and endanger our privacy and security.

1.2 Problem Definition

The objective of this project is to design and implement a chaos-based cryptographic system on hardware that is secure against Shor's Algorithm running on an ideal quantum computer.

This problem focuses on going beyond traditional cryptographic methods and implementing a new chaos based encryption-decryption technique in dedicated hardware like Field Programmable Gate Array (FPGA). Our aim is to design the system in such a way that it is simple enough to be implemented in practice, it is computationally efficient and it provides a reasonable degree of security. The system must also possess a number of fundamental features which are important for any cryptosystem in general. In order to achieve this goal some necessary steps to be performed along with the design and implementation are:

- Verification of Encryption-Decryption Algorithm
  - Validity of chaotic nature and quantum-safe properties of the scheme
- Hardware-level optimization of the Algorithm
  - Efficient pipelining of the algorithm for fast response in practical scenarios
- Key Generation & Management
  - Storage and maintenanace of valid keys and efficient key exchange strategies
- Security Analysis of the Cryptosystem
  - Analysis of randomness and complexity of the system

Chapter 2

Preliminaries

2.1 Present-day Cryptography

Cryptography is the study and practice of techniques for secure data transfer over insecure channels in presence of unauthorized users, usually called adversaries. Prior to modern age, cryptography was most exclusively referred to as encryption, i.e. the process of converting data from readable state to apparent noise.

Most of the present-day encryption algorithms are based on overlapping theory of mathematics and computer science. These are largely designed around 'trap-door functions', i.e. problems with sufficient computational hardness. These problems can be theoretically solved, but is is not possible to do within reasonable time and with the resources that are usually at disposal.

Advances in mathematics such as improved algorithms on the integer factorization problem and discrete logarithm problem and availability of more computational power require these methods to adapt with time. Most of the algorithms can, however, be made secure against these advancements just by increasing the key-length.

2.2 Shor's Algorithm

Published in 1995 and named after it's formulator, Shor's algorithm, is a quantum algorithm for finding the prime factors of any given integer N.

The importance of Shor's algorithm lies in it's ability to find the prime factors of an integer in polynomial time, in comparison to the most efficient classical algorithms known, such as general number field sieve, which takes sub-exponential time. This is possible to due to efficiency of quantum fourier transform and modular exponentiation by repeated squaring.

2.3 Shor's Algorithm & Present Encryption Methods

All of the popular cryptographic algorithms rely on one of the three hard problems - integer factorization problem, the discrete logarithm problem or the elliptic-curve discrete logarithm problem. These problems were viewed as important because many cryptography systems rely upon the difficulty of factoring large numbers. If an efficient method of factoring large numbers can be implemented, most of the encryption schemes would be next to worthless to protect their data.

With Shor's algorithm, these can be be solved in reasonable polynomial time. This implies that, with a quantum computer having sufficient number of qubits, Shor's algorithm can be used to break the public-key cryptography schemes including RSA. However, the experimental quantum computers available today succumb to noise and decoherence problems, hence, cannot be used to break current encryption schemes.

2.4 Post-Quantum Cryptography

Post-Quantum Cryptography refers to the study of classical encryption schemes that are considered to be secure against an attack by quantum computers. As discussed above, most of the popular encryption methods can be efficiently broken using a sufficiently powerful hypothetical quantum computer. Thus, scientists are preparing methods to secure data against attacks when quantum computing becomes much more powerful.

Daniel J. Bernstein[11] lists the following important classes of cryptographic methods beyond RSA, DSA and ECRSA -

Hash-based - This includes Merkle's hash-tree public-key signature system, building upon the one-message idea.

Code-based - An example is McEliece's hidden-Goppa-code public-key encryption system.

Lattice-based - The most popular example that garnered most of the interest is Hoffstein-Pipher-Silverman NTRU public-key-encryption system.

Multivariate quadratic equation based- One of the examples is Patarin's *HFEv* public-key-signature system originally proposed by Matsumoto and Imai.

Secret-key based - The prominent example is the DaemenRijmen *Rijndael* cipher which was later named AES (Advanced Encryption Standard).

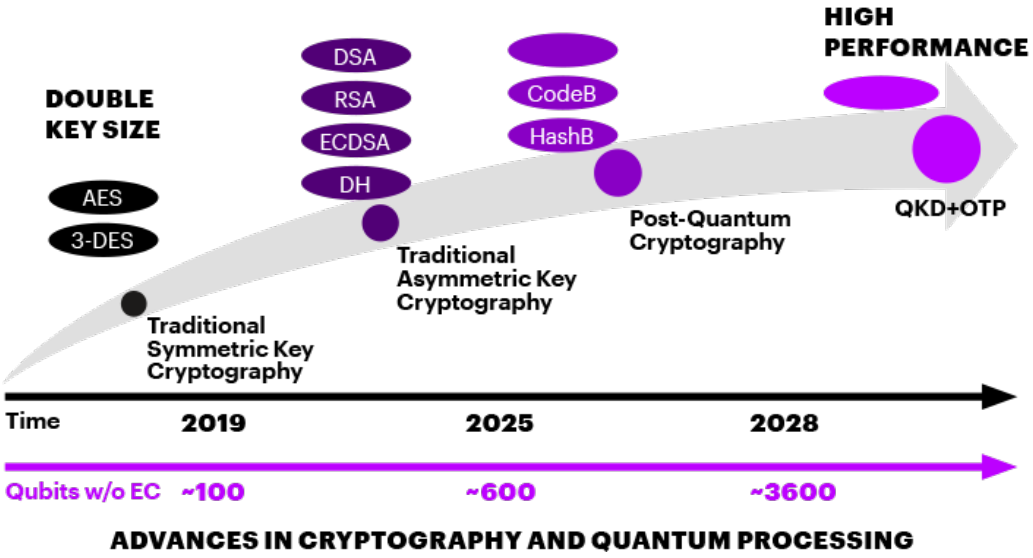
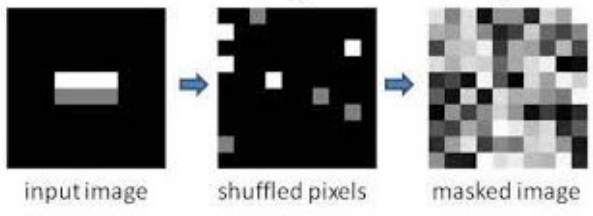


Figure 2.1: Timeline for Cryptography & Quantum Processing

2.5 Chaos based Cryptography

Apart from the popular post-quantum cryptographic methods, a new method of constructing cryptosystems utilising the non-predictability property of discrete chaotic systems has become somewhat noteworthy from practical perspectives. This type of systems are based on the characteristics of chaos, which are sensitivity of parameters, sensitivity of initial points, and randomness of sequences obtained by iterating a chaotic map. A ciphertext is obtained by the iteration of an inverse chaotic map from an initial point denoting a plaintext.

If the number of iterations is large enough, the randomness of the encryption and the decryption functions would be so large that attackers would be unable to break this cryptosystem by statistical characteristics. Hence, nonlinear dynamic systems with chaos are generally viewed to be good candidates for constructing such encryption-decryption algorithms. Most of these methods have been used in the development of symmetric ciphers for encryption of 2D images.



(a) Image Encryption using 1D Chaotic Map

(b) Chaos based Image Encryption

Figure 2.2: Chaos based Cryptography

2.6 Field Programmable Gate Array

A field programmable gate array is a set (array) of reconfigurable gates that can implement both sequential and combinational logic including multi-level logic functions.

FPGAs are built in form of an array of configurable logic blocks (CLBs). Each of these CLBs can be programmed to perform a logic function and then connected to each other through a hierarchy of interconnects (routing channels) to form a complex logic. FPGA also has a set of I/O elements for interfacing with external devices such as flash memories and network ports.

The structure of CLBs varies with model and manufacturers, but all of them share a lot of similarity. Figure 2.3 shows the structure of Cyclone IV CLB, consisting of a 4-input LUT and a flip-flop. By loading appropriate values, the LUT can be programmed to perform any 4-input function. Further, the choice of multiplexer select signals determines how the data is routed through LUT to the neighbouring LEs and IOEs. The LUT output either goes directly to the LE output for combinational logic, or it can be routed through the register for sequential logic.

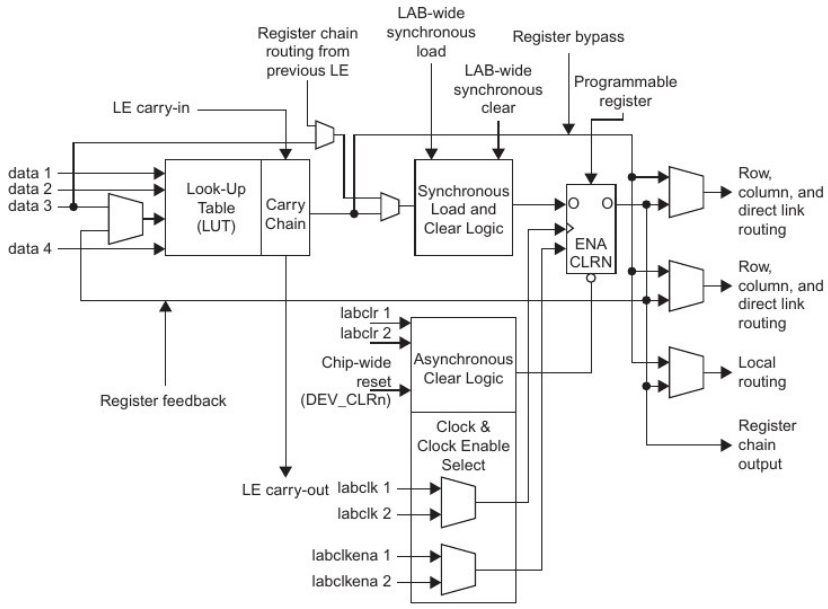


Figure 2.3: FPGA CLB Structure

An engineer programs the logic using Hardware Description Language (HDLs) such as VHDL or Verilog. A synthesis is then used to generate a gate-level netlist. An implementation tool then determines how the CLBs and routing channels should be configured to perform the specified function and generates a bitstream that can be loaded into a FPGA.

Chapter 3

Literature Survey

3.1 Chaos and Cryptography

Over last few years, there has been a great interest in understanding the working of chaotic systems. They are distinguished by their high sensitivity to initial conditions, statistical similarity to random signals and a continuous broad-band power spectrum. This has garnered interest from cryptanalysts and there have been several publications proposing various chaos-based cryptographic systems such as in [2], [4].

The chaos-based cryptosystems can be sub-divided into two classes. First one involves numerically computing a large number of iterations over time of a chaotic system, using message as the initial data. (see [1]). The second class amounts to scrambling a message with a chaotic dynamic function. This includes chaotic switching, additive masking, message embedding, etc.

The relevance and usefulness of chaos in these systems have been demonstrated through comparative studies between characteristics of chaotic systems and requirements of a strong cipher (see [1], [4]). Several properties of chaotic maps are similar to those of cryptographic maps : extreme sensitivity to initial conditions and parameters, unstable periodic orbits with large time-periods. Further, iterations of chaotic map spread the initial region over entire map, introducing diffusion which is an important requirement for a strong cipher.

3.2 Discrete Chaos

It must be noted that when chaotic systems are simulated on computers with limited precision, the sequences x_k generated are not exactly chaotic. Since, the cardinality of this set is finite, such sequences will always be a part of a loop of finite period. It can be expected that this period wouldn't be too short and will be greatly chaotic in nature.

Claiming such properties, however, requires some consideration [7]. Contributions made in this regard and discussion about discrete chaos can be found in [9]. However, some noteworthy takeaways are listed below -

- Through numerical experiments, it has been shown that mean cycle L of such a system is $O(2^{P/2})$, where P is the amount of precision in terms of number of bits. This serves as good reference while working with chaotic systems. However, it must be verified as there are no mathematical proofs to support it.
- The rounding error in computer systems poses another problem. The errors made in each iteration will culminate at a very fast rate due to high sensitivity of the system on the initial conditions. Thus, the actual trajectory and the calculated trajectory will be considerably different after a few iterations. However, a famous lemma called "Shadowing Lemma", guarantees that one can always find an actual trajectory that is arbitrarily near the calculated trajectory.

3.3 Chaos in Non-Linear Dynamic Systems

Many real world phenomena can be mathematically modelled as non-linear dynamic systems. Out of these phenomena, some exhibit significant degree of chaos. The unpredictability of these non-linear phenomena is due to the fact that the system passes through a series of *unstable states*. Also, these non-linear systems generally display very sensitive dependence on initial conditions which is the main reason for generating chaotic maps using non-linear dynamic systems. It must, however, be noted that not every complicated dynamic behavior can be considered chaotic. Chaotic systems differ from *noisy motion* in that their randomness is due to interaction of few simple laws. The quantitive description, however, lies in the concept of Lyapunov exponents which measures the exponential divergence of trajectories of the chaotic maps.

3.4 Baptista-type Cryptosystem

Proposed by M.S. Baptista, this is a chaotic cryptosystem based on the interval-partitioning of chaotic orbits of a 1D chaotic map called logistic map. This uses the ergodic property of chaos, which enables the construction of very fast and secure encryption-decryption schemes due to its simplicity and less complex structure. The main idea of this scheme is to first map the text characters to real values and then algorithms are applied to encrypt the message. Decryption is performed by iterating the chaotic map and then corresponding symbol for the real values are obtained by inverting the process.

Chapter 4

Proposed Approach

The proposed algorithm of encryption & decryption is based on multiple iterations of a certain dynamical chaotic system and using a Baptista-type encryption-decryption scheme from the chaotic map generated. After referring extensive literature and keeping in mind the resources at our disposal, we have decided to take the compound triple-pendulum model for creating the chaotic map in our cryptosystem. We assume that input to the system is a plain message. The system parameter(s) and additionally the initial conditions of the dynamical system are assumed to be the part of the secret key. After simulating the triple-pendulum, our next step would be to generate a valid set of keys for encryption and decryption. The entire mechanical model would then be implemented on *Artix FPGA* including the design and implementation of ALU and Control Unit, adhering to the performance specification of the problem. Additionally, an extra data transfer module can be added to the hardware for key exchanges.
The entire approach can be summarized using the following roadmap :-

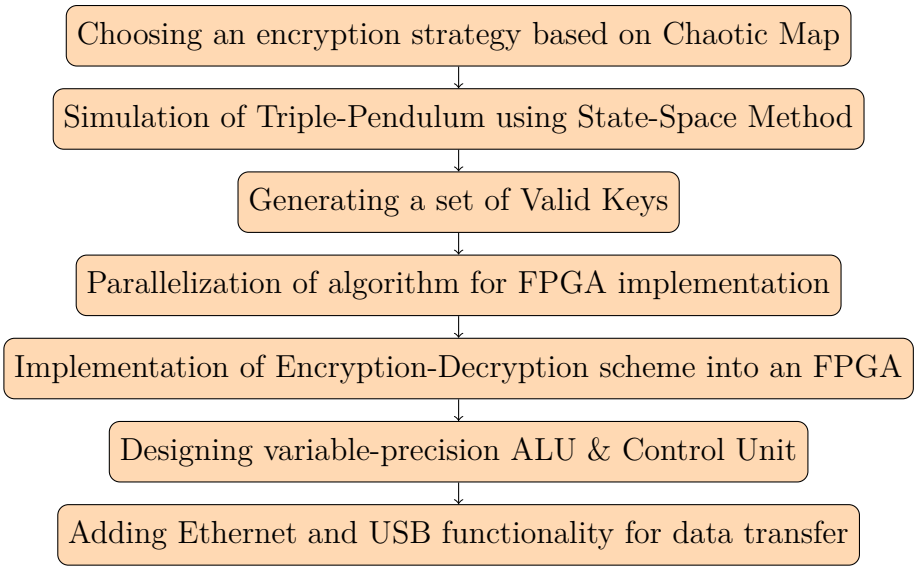


Figure 4.1: Implementation Steps

Chapter 5

Work Progress

5.1 Verifying the Algorithm in MATLAB

5.1.1 Model of Non-Linear Dynamic System

A schematic diagram of the compound triple-pendulum system is shown in Figure 5.1.The bars of the pendulum have significant mass so that it can be modeled as a compound pendulum. Damping factors were also included in the model for higher degree of chaos and non-linearity in the system.
Each bar i is defined by a set of four parameters:

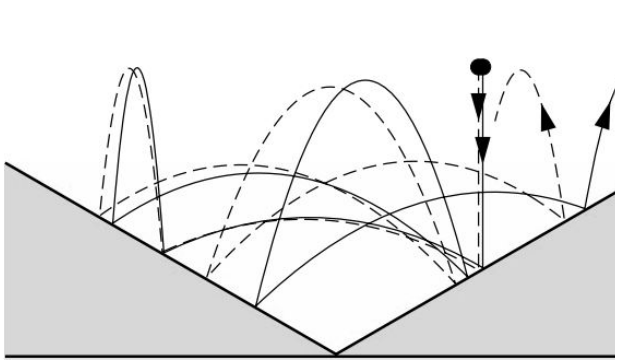
m_i, the mass of the bar,

l_i, the length of the bar,

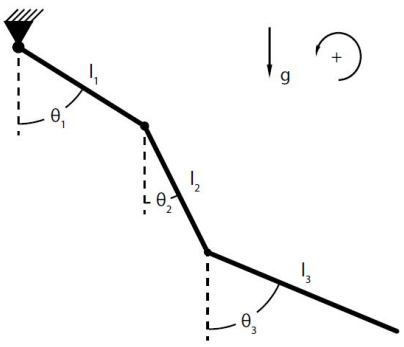
I_i, the moment of inertia of the bar and

k_i, the damping coefficient of the bar rotating about it's upper joint.

The position and velocity of the bars are defined by the six system state variables: θ_1, θ_2, θ_3, $\dot{\theta_1}$, $\dot{\theta_2}$, $\dot{\theta_3}$



(a) Chaotic motion of a bouncing ball



(b) Schematic Diagram of Triple Pendulum

Figure 5.1: Examples of Non-Linear Dynamic systems

Parameter	Value	Unit
m_1	0.2944	kg
m_2	0.1756	kg
m_3	0.0947	kg
l_1	0.508	m
l_2	0.254	m
l_3	0.127	m
I_1	9.526e-3	kg· m^2
I_2	1.625e-3	kg· m^2
I_3	1.848e-4	kg· m^2
k_1	5e-3	N·m·s/rad
k_2	0	N·m·s/rad
k_3	8e-4	N·m·s/rad

(a) Table 1

Condition	Value	Unit
θ_1	-0.4603	rad
θ_2	-1.2051	rad
θ_3	-1.5165	rad
$\dot{theta}_n$	0	rad/s

(b) Table 2

Figure 5.2: Parameters & Initial Conditions for the Initial Value Problem

5.1.2 Simulation of Compound Triple Pendulum

This compound triple-pendulum model has been simulated using MATLAB using approximate differential equations describing the random motions. The parameters and initial conditions of the ODEs are given in Table 1 and Table 2 respectively. For the simulation, numerical methods were used to solve the state space model of the differential equations and the values corresponding to the angular position of the bars were obtained within a certain duration of time with a predefined precision. This generates the mapping values for the encryption module.

$$\ddot{\theta}_1 = -(2((l_3^2 m_3^2 \sin(2\theta_1 - 2\theta_3)(4I_2 - l_2^2 m_2) + l_2^2 \sin(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2 m_3 l_3^2 + 4I_3(m_2 + 2m_3)))l_1^2 \dot{\theta_1}^2 + (l_2(\sin(\theta_1 - \theta_2)((m_2 m_3(m_2 + 3m_3)l_3^2 + 4I_3(m_2^2 + 6m_2 m_3 + 8m_3^2))l_2^2 + 4I_2(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3))) + l_3^2 m_3^2 \sin(\theta_1 + \theta_2 - 2\theta_3)(4I_2 - l_2^2 m_2))\dot{\theta_2}^2 - 4k_2 l_2(cos(\theta_1 - \theta_2)(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3)) - l_3^2 m_3^2 cos(\theta_1 + \theta_2 - 2\theta_3))\dot{\theta_2} + l_3 m_3(\sin(\theta_1 - \theta_3)(8I_3 m_3 l_2^2 + 4I_2 m_3 l_3^2 + 16I_2 I_3) + l_2^2 \sin(\theta_1 - 2\theta_2 + \theta_3)(m_2 m_3 l_3^2 + 4I_3(m_2 + 2m_3)))\dot{\theta_3}^2 - 4k_3 l_3 m_3(cos(\theta_1 - \theta_3)(2m_3 l_2^2 + 4I_2) - l_2^2 cos(\theta_1 - 2\theta_2 + \theta_3)(m_2 + 2m_3))\dot{\theta_3} - g(\sin(\theta_1)((m_3(m_1 m_2 + 2m_1 m_3 + 3m_2 m_3 + m_2^2)l_3^2 + 4I_3(m_2^2 + 6m_2 m_3 + m_1 m_2 + 4m_3^2 + 4m_1 m_3))l_2^2 + 4I_2(m_3(m_1 + 2m_2 + m_3)l_3^2 + 4I_3(m_1 + 2m_2 + 2m_3))) + l_3^2 m_3^2(\sin(\theta_1 - 2\theta_3)(4I_2 - l_2^2 m_2) - 2l_2^2 cos(2\theta_2 - 2\theta_3)\sin(\theta_1)(m_1 + m_2)) + l_2^2 \sin(\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2 m_3 l_3^2 + 4I_3(m_2 + 2m_3))))l_1 + 2k_1(4I_2(m_3 l_3^2 + 4I_3) + l_2^2(m_3(m_2 + 2m_3)l_3^2 + 4I_3(m_2 + 4m_3)) - 2l_2^2 l_3^2 m_3^2 cos(2\theta_2 - 2\theta_3))\dot{\theta_1}))/(64I_1 I_2 I_3 + 8I_3 l_1^2 l_2^2 m_2^2 + 8I_1 l_2^2 l_3^2 m_3^2 + 8I_2 l_1^2 l_3^2 m_3^2 + 32I_3 l_1^2 l_2^2 m_3^2 + 16I_2 I_3 l_1^2 m_1 + 16I_1 I_3 l_2^2 m_2 + 64I_2 I_3 l_1^2 m_2 + 16I_1 I_2 l_3^2 m_3 + 64I_1 I_3 l_2^2 m_3 + 64I_2 I_3 l_1^2 m_3 + 4I_3 l_1^2 l_2^2 m_1 m_2 + 4I_2 l_1^2 l_3^2 m_1 m_3 + 16I_3 l_1^2 l_2^2 m_1 m_3 + 4I_1 l_2^2 l_3^2 m_2 m_3 + 16I_2 l_1^2 l_3^2 m_2 m_3 + 48I_3 l_1^2 l_2^2 m_2 m_3 - 8I_1 l_2^2 l_3^2 m_3^2 cos(2\theta_2 - 2\theta_3) - 2l_1^2 l_2^2 cos(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2 m_3 l_3^2 + 4I_3(m_2 + 2m_3)) - 2l_1^2 l_3^2 m_3^2 cos(2\theta_1 - 2\theta_3)(-m_2 l_2^2 + 4I_2) + 2l_1^2 l_2^2 l_3^2 m_1 m_3^2 + 6l_1^2 l_2^2 l_3^2 m_2 m_3^2 + 2l_1^2 l_2^2 l_3^2 m_2^2 m_3 + l_1^2 l_2^2 l_3^2 m_1 m_2 m_3 - 2l_1^2 l_2^2 l_3^2 m_1 m_3^2 cos(2\theta_2 - 2\theta_3) - 4l_1^2 l_2^2 l_3^2 m_2 m_3^2 cos(2\theta_2 - 2\theta_3))$$

Figure 5.3: Differential Equation for θ_1

$$\ddot{\theta_2} = (2((l_1^2\sin(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2m_3l_3^2 + 4I_3(m_2 + 2m_3)) - l_3^2m_3^2\sin(2\theta_2 - 2\theta_3)((m_1 + 2m_2)l_1^2 + 4I_1))l_2^2\dot{\theta_2}^2 + l_1(\sin(\theta_1 - \theta_2)((m_3(m_1(m_2+m_3)+2m_2(2m_2+3m_3))l_3^2 + 4I_3(m_2+2m_3)(m_1 + 4m_2 + 4m_3))l_1^2 + 4I_1(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3))) - l_3^2m_3^2\sin(\theta_1 + \theta_2 - 2\theta_3)((m_1 + 2m_2)l_1^2 + 4I_1))l_2\dot{\theta_1}^2 + 4k_1l_1(cos(\theta_1 - \theta_2)(m_3(m_2 + m_3)l_3^2 + 4I_3(m_2 + 2m_3)) - l_3^2m_3^2cos(\theta_1 + \theta_2 - 2\theta_3))l_2\dot{\theta_1} + (-l_3m_3(\sin(\theta_2-\theta_3)((m_3(m_1+3m_2)l_3^2+4I_3(m_1+3m_2+2m_3))l_1^2+4I_1(m_3l_3^2+4I_3)) - l_1^2\sin(2\theta_1-\theta_2-\theta_3)(m_2m_3l_3^2 + 4I_3(m_2 + 2m_3)))\dot{\theta_3}^2 + 4k_3l_3m_3(cos(\theta_2 - \theta_3)((m_1 + 3m_2 + 2m_3)l_1^2 + 4I_1) - l_1^2cos(2\theta_1 - \theta_2 - \theta_3)(m_2+2m_3))\theta_3 + g(\sin(\theta_2)((m_2m_3(2m_2+3m_3)l_3^2+8I_3(m_2^2+3m_2m_3+2m_3^2))l_1^2+4I_1(m_3(m_2+m_3)l_3^2 + 4I_3(m_2 + 2m_3))) - l_1^2\sin(2\theta_1 - \theta_2)(m_3(m_1(m_2 + m_3) + m_2(2m_2 + 3m_3))l_3^2 + 4I_3(m_2 + 2m_3)(m_1 + 2m_2 + 2m_3)) + l_3^2m_3^2(\sin(\theta_2 - 2\theta_3)(m_2l_1^2 + 4I_1) + l_1^2\sin(2\theta_1 + \theta_2 - 2\theta_3)(m_1 + m_2))))l_2 - 2k_2(4I_1(m_3l_3^2 + 4I_3) + l_1^2(m_3(m_1 + 4m_2 + 2m_3)l_3^2 + 4I_3(m_1 + 4m_2 + 4m_3)) - 2l_1^2l_3^2m_3^2cos(2\theta_1 - 2\theta_3))\dot{\theta_2}))/(64I_1I_2I_3 + 8I_3l_1^2l_2^2m_2^2 + 8I_1l_2^2l_3^2m_3^2 + 8I_2l_1^2l_3^2m_3^2 + 32I_3l_1^2l_2^2m_3^2 + 16I_2I_3l_1^2m_1 + 16I_1I_3l_2^2m_2 + 64I_2I_3l_1^2m_2 + 16I_1I_2l_3^2m_3 + 64I_1I_3l_2^2m_3 + 64I_2I_3l_1^2m_3 + 4I_3l_1^2l_2^2m_1m_2 + 4I_2l_1^2l_3^2m_1m_3 + 16I_3l_1^2l_2^2m_1m_3 + 4I_1l_2^2l_3^2m_2m_3 + 16I_2l_1^2l_3^2m_2m_3 + 48I_3l_1^2l_2^2m_2m_3 - 8I_1l_2^2l_3^2m_3^2cos(2\theta_2 - 2\theta_3) - 2l_1^2l_2^2cos(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2m_3l_3^2+4I_3(m_2+2m_3)) - 2l_1^2l_3^2m_3^2cos(2\theta_1-2\theta_3)(-m_2l_2^2+4I_2) + 2l_1^2l_2^2l_3^2m_1m_3^2 + 6l_1^2l_2^2l_3^2m_2m_3^2 + 2l_1^2l_2^2l_3^2m_2^2m_3 + l_1^2l_2^2l_3^2m_1m_2m_3 - 2l_1^2l_2^2l_3^2m_1m_3^2cos(2\theta_2 - 2\theta_3) - 4l_1^2l_2^2l_3^2m_2m_3^2cos(2\theta_2 - 2\theta_3))$$

Figure 5.4: Differential Equation for θ_2

$$\ddot{\theta_3} = -(2(32I_1I_2k_3\dot{\theta_3} - l_2l_3m_3\dot{\theta_2}^2(\sin(\theta_2 - \theta_3)((l_2^2(m_1m_2 + 4m_1m_3 + 6m_2m_3 + m_2^2) + 4I_2(m_1 + 3m_2 + 2m_3))l_1^2 + 4I_1(4I_2 + l_2^2(m_2 + 4m_3))) + l_1^2\sin(2\theta_1 - \theta_2 - \theta_3)(m_2 + 2m_3)(4I_2 - m_2l_2^2)) - l_1l_3m_3\dot{\theta_1}^2(\sin(\theta_1 - \theta_3)(8I_1(m_3l_2^2 + 2I_2) + 2l_1^2((m_1m_3 - m_2^2)l_2^2 + 2I_2(m_1 + 4m_2 + 4m_3))) - l_2^2\sin(\theta_1 - 2\theta_2 + \theta_3)(m_2 + 2m_3)((m_1 + 2m_2)l_1^2 + 4I_1)) + 4k_3l_1^2l_2^2m_2^2\dot{\theta_3} + 16k_3l_1^2l_2^2m_3^2\dot{\theta_3} + 8I_2k_3l_1^2m_1\dot{\theta_3} + 8I_1k_3l_2^2m_2\dot{\theta_3} + 32I_2k_3l_1^2m_2\dot{\theta_3} + 32I_1k_3l_2^2m_3\dot{\theta_3} + 32I_2k_3l_1^2m_3\dot{\theta_3} - 4k_1l_1l_3m_3\dot{\theta_1}(cos(\theta_1 - \theta_3)(2m_3l_2^2 + 4I_2) - l_2^2cos(\theta_1 - 2\theta_2 + \theta_3)(m_2 + 2m_3)) - 16I_1I_2gl_3m_3\sin(\theta_3) - 4I_2l_1^2l_3^2m_3^2\dot{\theta_3}^2\sin(2\theta_1 - 2\theta_3) - 4I_1l_2^2l_3^2m_3^2\dot{\theta_3}^2\sin(2\theta_2 - 2\theta_3) + 8I_2gl_1^2l_3m_3^2\sin(2\theta_1 - \theta_3) + 8I_1gl_2^2l_3m_3^2\sin(2\theta_2 - \theta_3) + 2k_3l_1^2l_2^2m_1m_2\dot{\theta_3} + 8k_3l_1^2l_2^2m_1m_3\dot{\theta_3} + 24k_3l_1^2l_2^2m_2m_3\dot{\theta_3} - 4k_2l_2l_3m_3\dot{\theta_2}(cos(\theta_2 - \theta_3)((m_1 + 3m_2 + 2m_3)l_1^2 + 4I_1) - l_1^2cos(2\theta_1 - \theta_2 - \theta_3)(m_2 + 2m_3)) - 8I_1gl_2^2l_3m_3^2\sin(\theta_3) - 8I_2gl_1^2l_3m_3^2\sin(\theta_3) - 4k_3l_1^2l_2^2m_2^2\dot{\theta_3}cos(2\theta_1 - 2\theta_2) - 16k_3l_1^2l_2^2m_3^2\dot{\theta_3}cos(2\theta_1 - 2\theta_2) - 8I_2gl_1^2l_3m_2m_3\sin(\theta_3) - 16k_3l_1^2l_2^2m_2m_3\dot{\theta_3}cos(2\theta_1 - 2\theta_2) - l_1^2l_2^2l_3^2m_1m_3^2\dot{\theta_3}^2\sin(2\theta_2 - 2\theta_3) + l_1^2l_2^2l_3^2m_2m_3^2\dot{\theta_3}^2\sin(2\theta_1 - 2\theta_3) - 2l_1^2l_2^2l_3^2m_2m_3^2\dot{\theta_3}^2\sin(2\theta_2 - 2\theta_3) + 2gl_1^2l_2^2l_3m_1m_3^2\sin(2\theta_1 - \theta_3) - gl_1^2l_2^2l_3m_2^2m_3\sin(2\theta_1 - \theta_3) + 2gl_1^2l_2^2l_3m_2m_3^2\sin(2\theta_2 - \theta_3) + gl_1^2l_2^2l_3m_2^2m_3\sin(2\theta_2 - \theta_3) + 4I_2gl_1^2l_3m_1m_3\sin(2\theta_1 - \theta_3) + 8I_2gl_1^2l_3m_2m_3\sin(2\theta_1 - \theta_3) + 4I_1gl_2^2l_3m_2m_3\sin(2\theta_2 - \theta_3) - 2gl_1^2l_2^2l_3m_1m_3^2\sin(2\theta_1 - 2\theta_2 + \theta_3) - 2gl_1^2l_2^2l_3m_2m_3^2\sin(2\theta_1 - 2\theta_2 + \theta_3) - gl_1^2l_2^2l_3m_2^2m_3\sin(2\theta_1 - 2\theta_2 + \theta_3) + gl_1^2l_2^2l_3m_2^2m_3\sin(\theta_3) - gl_1^2l_2^2l_3m_1m_2m_3\sin(2\theta_1 - 2\theta_2 + \theta_3)))/(64I_1I_2I_3 + 8I_3l_1^2l_2^2m_2^2 + 8I_1l_2^2l_3^2m_3^2 + 8I_2l_1^2l_3^2m_3^2 + 32I_3l_1^2l_2^2m_3^2 + 16I_2I_3l_1^2m_1 + 16I_1I_3l_2^2m_2 + 64I_2I_3l_1^2m_2 + 16I_1I_2l_3^2m_3 + 64I_1I_3l_2^2m_3 + 64I_2I_3l_1^2m_3 + 4I_3l_1^2l_2^2m_1m_2 + 4I_2l_1^2l_3^2m_1m_3 + 16I_3l_1^2l_2^2m_1m_3 + 4I_1l_2^2l_3^2m_2m_3 + 16I_2l_1^2l_3^2m_2m_3 + 48I_3l_1^2l_2^2m_2m_3 - 8I_1l_2^2l_3^2m_3^2cos(2\theta_2 - 2\theta_3) - 2l_1^2l_2^2cos(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2m_3l_3^2+4I_3(m_2+2m_3)) - 2l_1^2l_3^2m_3^2cos(2\theta_1 - 2\theta_3)(4I_2 - m_2l_2^2) + 2l_1^2l_2^2l_3^2m_1m_3^2 + 6l_1^2l_2^2l_3^2m_2m_3^2 + 2l_1^2l_2^2l_3^2m_2^2m_3 + l_1^2l_2^2l_3^2m_1m_2m_3 - 2l_1^2l_2^2l_3^2m_1m_3^2cos(2\theta_2 - 2\theta_3) - 4l_1^2l_2^2l_3^2m_2m_3^2cos(2\theta_2 - 2\theta_3))$$

Figure 5.5: Differential Equation for θ_3

The above equations can be simlified as follows:

$$\begin{aligned}
\ddot{\theta_1}^{(t+1)} &= f_1(\theta_1^{(t)}, \theta_2^{(t)}, \theta_3^{(t)}, \dot{\theta_1}^{(t)}, \dot{\theta_2}^{(t)}, \dot{\theta_3}^{(t)}) \\
\ddot{\theta_2}^{(t+1)} &= f_2(\theta_1^{(t)}, \theta_2^{(t)}, \theta_3^{(t)}, \dot{\theta_1}^{(t)}, \dot{\theta_2}^{(t)}, \dot{\theta_3}^{(t)}) \\
\ddot{\theta_3}^{(t+1)} &= f_3(\theta_1^{(t)}, \theta_2^{(t)}, \theta_3^{(t)}, \dot{\theta_1}^{(t)}, \dot{\theta_2}^{(t)}, \dot{\theta_3}^{(t)})
\end{aligned} \tag{5.1}$$

The three values $\ddot{\theta}_1, \ddot{\theta}_2, \ddot{\theta}_3$ are calulated using the above equations. The differential equations are solved numerically using the following equations by iterating for t = 0 to N where $N = \frac{T}{\Delta t}$:

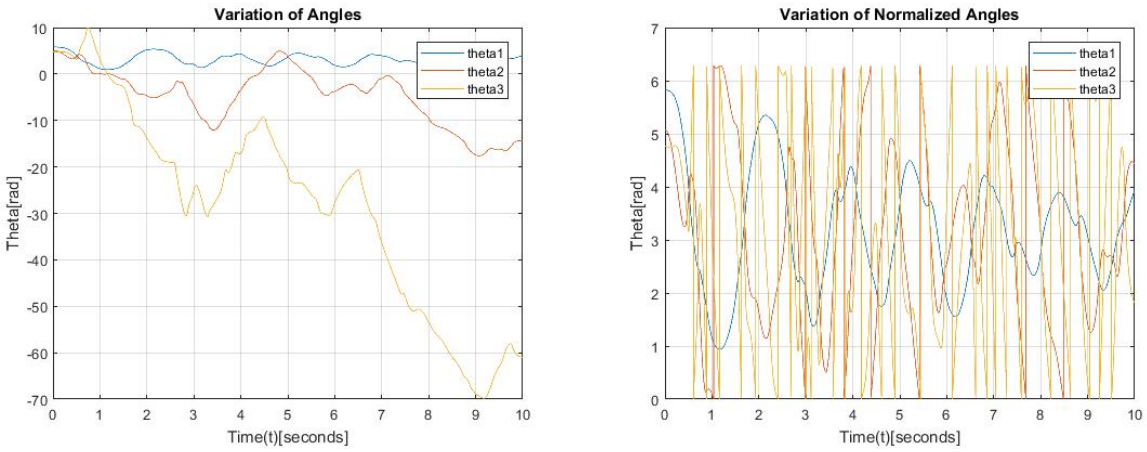
$$\begin{aligned}
\theta_1^{(t+1)} &\leftarrow \theta_1^{(t)} + \dot{\theta_1}^{(t+1)} \Delta t \\
\theta_1^{(t+1)} &\leftarrow \theta_1^{(t)} + \dot{\theta_1}^{(t+1)} \Delta t \\
\theta_1^{(t+1)} &\leftarrow \theta_1^{(t)} + \dot{\theta_1}^{(t+1)} \Delta t
\end{aligned} \tag{5.2}$$

$$\begin{aligned}
\dot{\theta_1}^{(t+1)} &\leftarrow \dot{\theta_1}^{(t)} + \ddot{\theta_1}^{(t+1)} \Delta t \\
\dot{\theta_2}^{(t+1)} &\leftarrow \dot{\theta_2}^{(t)} + \ddot{\theta_2}^{(t+1)} \Delta t \\
\dot{\theta_3}^{(t+1)} &\leftarrow \dot{\theta_3}^{(t)} + \ddot{\theta_3}^{(t+1)} \Delta t
\end{aligned} \tag{5.3}$$

Store the values of θ in a vector y such that
$\mathbf{y} = \{\theta_i^{(0)}, \theta_i^{(1)}, \theta_i^{(2)}, ..., \theta_i^{(N-1)}\}$ for $i = 1, 2, 3$ where $N = \frac{T}{\Delta t}$
Normalize the values of y in the range $[0, 2\pi]$ (or $[-\pi, \pi]$) to get
$\hat{\mathbf{y}}^i = \{\hat{\theta}_i^{(0)}, \hat{\theta}_i^{(1)}, \hat{\theta}_i^{(2)}, ..., \hat{\theta}_i^{(N-1)}\}$ for $i = 1, 2, 3$ where $\hat{\theta}_i^{(t)} = \theta_i^{(t)} - \lfloor \frac{\theta_i^{(t)}}{2\pi} \rfloor \times 2\pi$
The three state variable value can be combined using a one-one function F, such that $\hat{\mathbf{y}} = F(\hat{\mathbf{y}}^1, \hat{\mathbf{y}}^2, \hat{\mathbf{y}}^3)$ which is also normalized. For simplicity, we use $\hat{\mathbf{y}} = \hat{\mathbf{y}}^3$ here.

Simulation Results

These are some of the observations from the simulation of the triple-pendulum model for a duration of t = 0 to t = 10 seconds with Δt = 0.001 :
(i) Initial conditions are same as given in Table 1 & 2:



(a) Plot of θ vs Time (b) Plot of Normalized θ vs Time

Figure 5.6: Motion of Triple-Pendulum for t = 0 to t = 10 sec

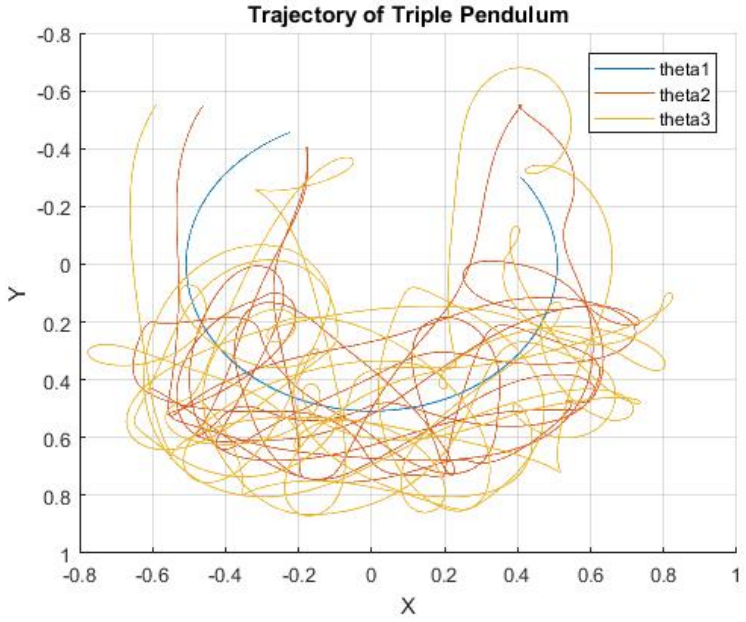


Figure 5.7: Plot of Motion of Triple-Pendulum

5.1.3 Encryption - Decryption

Our approach is to convert the plain-text of M characters into ascii format and map the values to the intervals partitioned from the range of state variable values of the triple-pendulum motion simulated within a specific duration of time for a particular set of parameters and initial conditions. The initial conditions and parameters of the different equation forms a part of the private key. The differential equations were numerically integrated to get the state variable values $\theta_1, \theta_2, \theta_3$ followed by normalization of the values in the range $[0, 2\pi]$ or $[-\pi, \pi]$. Then using Baptista-type method, the entire range of the normalized chaotic function $[\hat{y}_{min}, \hat{y}_{max}]$ was partitioned into a number of intervals equal to the number of characters(N_c). Each character in the plain-text is then mapped to a specific interval(k) and then to a time point (i) randomly selecting from that interval. We have used a permutation map $\pi(k)$ for mapping interval id to the character's ascii value.

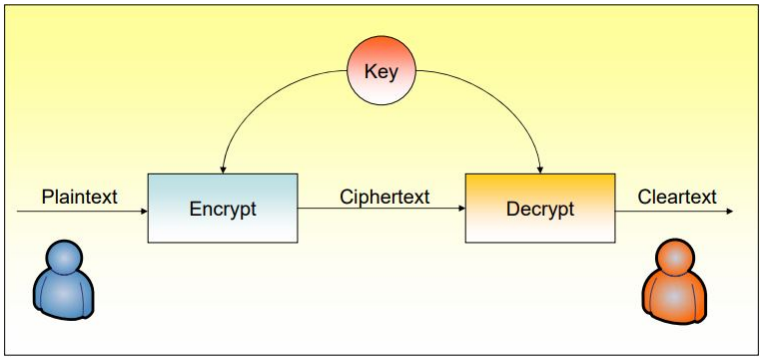


Figure 5.8: Working Principle of a Symmetric Cryptosystem

Algorithm 1: Encryption

Input: $\hat{\mathbf{y}}$, $\mathbf{P}$, N_c, η

Output: $\mathbf{C}$

Initialize D = {}

Find $\hat{y}_{max} = \max_{1 \leq i \leq N}(\hat{y}_i)$, $\hat{y}_{min} = \min_{1 \leq i \leq N}(\hat{y}_i)$

Calculate $\epsilon = \frac{(\hat{y}_{max} - \hat{y}_{min})}{N_c}$

For each $\hat{y}_i$ in $\hat{\mathbf{y}}$

  Calculate $k = \lfloor \frac{\hat{y}_i}{\epsilon} \rfloor$

  $D(\pi(k)) \leftarrow D(\pi(k)) \cup \{i\}$

For $m = 1$ to M

  Initialize $j = 1$, $p = \mathbf{P}(m)$

  Calculate $\hat{p} = ASCII(p)$, $l = size(D(\pi(\hat{p})))$

  Generate random number r

  While $r \leq \eta$, do $j = (j+1)\%l$ else $\mathbf{C}(m) = D(\pi(\hat{p}), j)$

Return $\mathbf{C}$

On the decryption module, the interval in which the encrypted value lies is computed from the generated motion of the triple-pendulum for the same key. Applying inverse permutation map ($\pi^{-1}(k)$) the corresponding index is calculated which would then refer to the ascii converted clear-text($\tilde{\mathbf{P}}$). Converting them into characters, the message can be decoded.

Algorithm 2: Decryption

Input: $\hat{\mathbf{y}}$,$\mathbf{C}$

Output: $\tilde{\mathbf{P}}$

Find $\hat{y}_{max} = \max_{1 \leq i \leq N}(\hat{y}_i)$, $\hat{y}_{min} = \min_{1 \leq i \leq N}(\hat{y}_i)$

Calculate $\epsilon = \frac{(\hat{y}_{max} - \hat{y}_{min})}{N_c}$

For $m = 1$ to M

  Initialize $c = \mathbf{C}(m)$

  Calculate $\hat{k} = \lfloor \frac{\hat{y}_c}{\epsilon} \rfloor$

  Compute $k = \pi^{-1}(\hat{k})$

  $\tilde{\mathbf{P}}(m) \leftarrow CHAR(k)$

Return $\tilde{\mathbf{P}}$

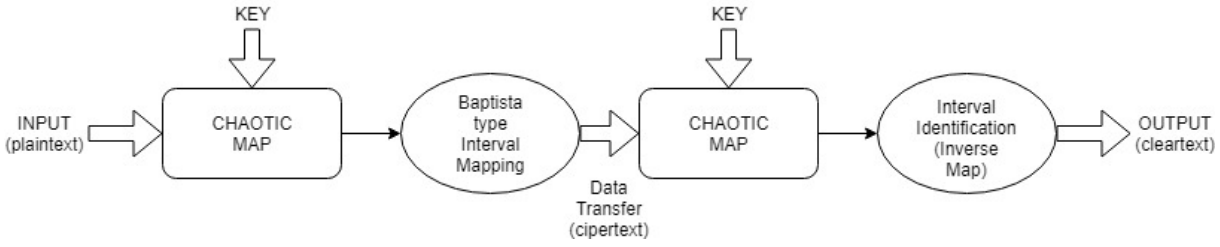


Figure 5.9: Encryption-Decryption Strategy

5.2 Key Generation

It is observed that for certain specific parameters or initial conditions, the motion of the bars of the triple-pendulum shows periodic nature after a certain span of time. Hence there is a need to eliminate those parameters or initial conditions for which the motion is periodic as the periodic nature breaks the chaotic behavior of the system. For that a test for periodicity was employed to extract the prominent period of the signal using statistical analysis on the spectrum of the signal. The test used is known as ***Fisher's g-statistic test*** (see [11]). This method is based on the test of significance of the periodic components of the signal derived from its periodogram.

A periodogram is used to identify the hidden periods (or frequencies) of a time series. This is a helpful tool for identifying the dominant periodic behavior in a series. The periodogram provides a measure of the relative importance of possible frequency values that can explain the oscillation pattern of the observed data.

The periodogram estimate of the Power Spectral Density of a signal $x_L(n)$ of length L is defined as

$$P_{xx}(f) = \frac{1}{LF_s} \mid \sum_{n=0}^{L-1} x_L(n) e^{-j2\pi fn/F_s} \mid^2 \tag{5.4}$$

where F_s is the sampling frequency. The computation of $P_{xx}(f)$ can be performed only at a finite number of discrete frequency points given by

$$f_k = \frac{kF_s}{N}, k = 0, 1, ..., N-1 \tag{5.5}$$

Fisher's g-statistic is defined as the ratio of the maximum periodogram value($P_{xx}(f_k)$ or $I(\omega_k)$) to the sum of all periodogram values of the signal.

$$g = \frac{\max_k I(\omega_k)}{\sum_{k=1}^{[N/2]} I(\omega_k)} \tag{5.6}$$

where $I(\omega_k)$ are the periodogram values. Using the value of g, we are able to calculate the dominant periods of the signal as shown in [11].

The key of this symmetric cryptosystem includes the parameters, initial conditions, time duration, time step, minimum value of $\hat{\mathbf{y}}$ and ϵ.
$K = \{\theta_1^{(0)}, \theta_2^{(0)}, \theta_3^{(0)}, \dot{\theta}_1^{(0)}, \dot{\theta}_2^{(0)}, \dot{\theta}_3^{(0)}, m_1, m_2, m_3, l_1, l_2, l_3, I_1, I_2, I_3, k_1, k_2, k_3, g, \hat{y}_{min}, \hat{y}_{min}/\epsilon\}$

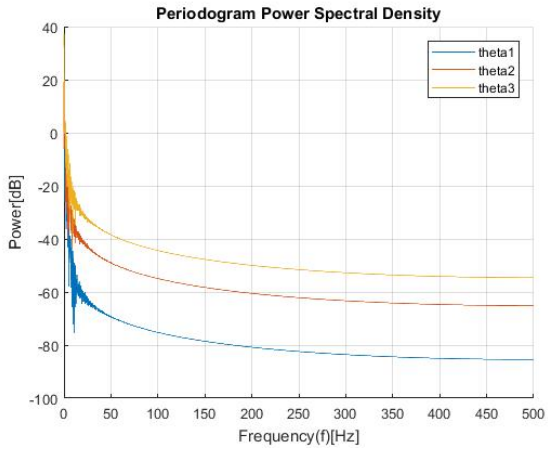
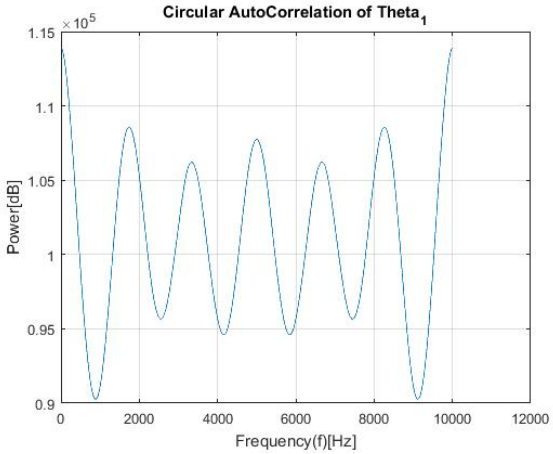
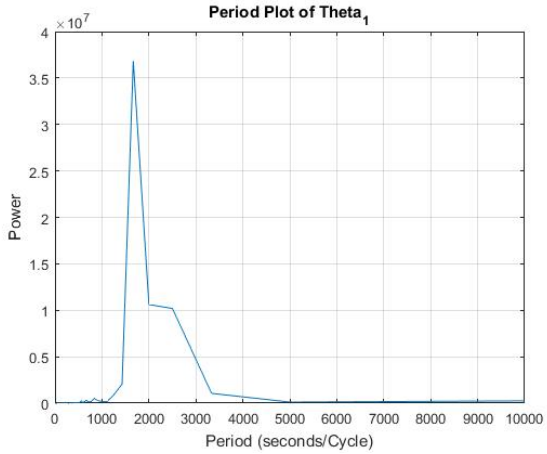


Figure 5.10: Periodogram Plot for θ

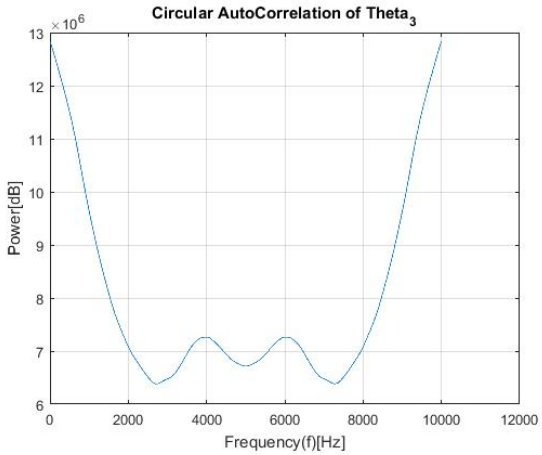


(a) Plot of Circular Auto-Correlation for θ_1

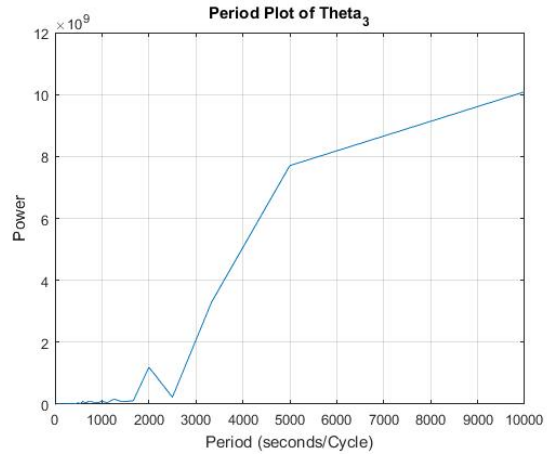


(b) Plot of Periodicity for θ_1

Figure 5.11: Periodic Properties of θ_1



(a) Plot of Circular Auto-Correlation for θ_3



(b) Plot of Periodicity for θ_3

Figure 5.12: Periodic Properties of θ_3

From the figures 5.8 & 5.9 , it is observed that there is peak at 1700 (seconds/Cycle) for θ_1 whereas no such peak occurs for θ_3. Thus θ_1 is periodic in nature. The values obtained from the periodicity test and plots of circular auto-correlation clearly differentiates the parameters and initial values which leads to periodic nature of the motion and those which lead to non-periodic nature of the motion. Thus by iterating through all possibles values of the parameters and checking likewise for non-periodic nature, a set of keys was generated and stored.

5.3 FPGA Implementation

The complete design has been implemented at Register-Transfer Level (RTL) in **SystemVerilog HDL** and the target device chosen is Digilent Nexys Board with **Xilinx Artix-7 FPGA**. The synthesis tool used is Xilinx Synthesis Tool (XST). The following diagram shows the implementation plan for the design :

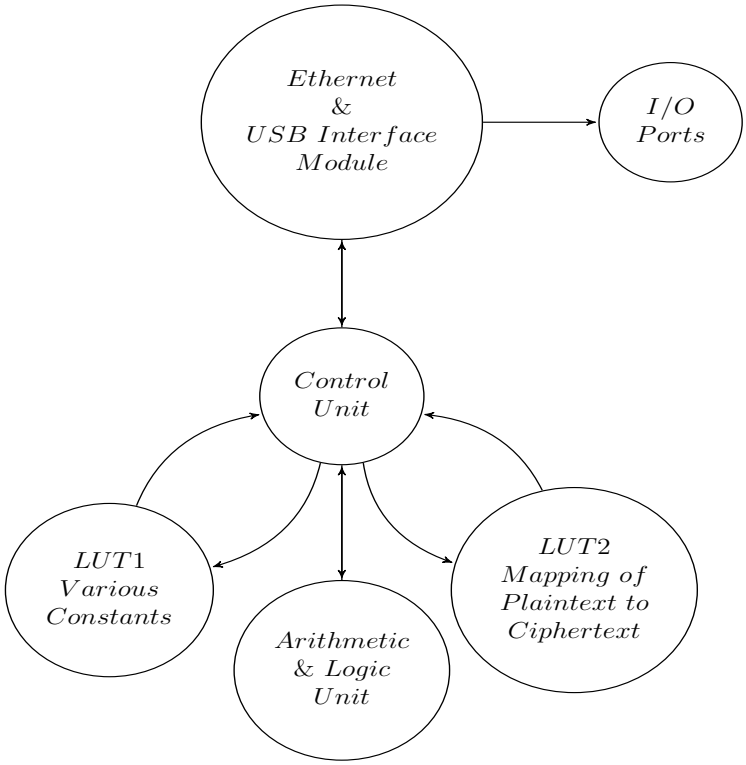


Figure 5.13: Overview of FPGA Implementation

For simplicity in verilog design, we fix the permuation map ($\pi(k)$) as the identity map and also store atmost three state variable values for a particular interval(i.e. for each character) during the encryption phase. As a result, we are require to search over N_c for the interval index during decryption phase. To evaluate Eq. (5.1) we convert the RHS expression into postfix format and store the postfix expression in ROM. Then we

evaluate the expression using the stack based postfix evaluation technique. Also the angle combinations of the sinusoidal terms present in the expressions are cached in LUT and appropriately decoded while evaluating the expression.

5.3.1 Memory & Look Up Table (LUT)

A number of LUTs are used for solving the differential equations by evaluating the postfix expressions in FPGA. These are :-

1. Angle Combinations
2. Parameters
3. State Variables
4. Numeric constants

Table 5.1: LUT Details

(a)

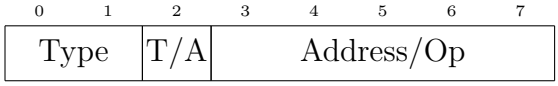
Angle Combinations	**Value at t=0**
$2\theta_1 - 2\theta_3$	2.1124
$2\theta_1 - 2\theta_2$	1.4896
$\theta_1 - \theta_2$	0.7448
$\theta_1 + \theta_2 - 2\theta_3$	1.3676
$\theta_1 - \theta_3$	1.0562
$\theta_1 - 2\theta_2 + \theta_3$	0.4334
θ_1	5.8229
$\theta_1 - 2\theta_3$	−3.7105
$2\theta_2 - 2\theta_3$	0.6228
$\theta_1 - 2\theta_2$	−4.3333
θ_2	5.0781
$\theta_2 - \theta_3$	0.3114
$2\theta_1 - \theta_2 - \theta_3$	1.801
$2\theta_1 - \theta_2$	6.5677
$\theta_2 - 2\theta_3$	−4.4553
$2\theta_1 + \theta_2 - 2\theta_3$	7.1905
θ_3	4.7667
$2\theta_1 - \theta_3$	6.8791
$2\theta_2 - \theta_3$	5.3895
$2\theta_1 - 2\theta_2 + \theta_3$	6.2563

(b)

Constants
0
1
2
3
4
6
8
12
16
24
32
48
64

Postfix Evaluation: The RHS expressions in the Eq.(5.1) are converted into postfix format. In the postfix expression, there are 4 types of terms:(1) Numeric constants ,(2) State Variable/Parameters, (3) Operations & (4) Sinusoidal(sin or cos). Each such term are encoded to an 8-bit code. The entire postfix expression is stored in a LUT in coded format. During postfix evaluation these codes are decoded and the appropriate data value

is fetched from the corresponding addresses. The meaning of the 8-bit encoding is shown below:



Here, the first 2 bits, determines the type of term.

00 ← Constants, 01 ← Parameters, 10 ← Sinusoidal, 11 ← Operation

If first 2 bits are 01, check the next bit(T/A). If it is 0 then it is a parameter else state variable term. The rest 5 bits are address to the LUT (Fig. 5.2) for the corresponding state variable or parameter value in order.

If first 2 bits are 10, check the next bit(T/A). If it is 0 then it is a sine term else it is a cosine term. The rest 5 bits are address to the LUT (Table 5.1(a)) for the corresponding angle combination in order.

If first 2 bits are 11, check the last 3 bits. They denote the following:

000 ← Exponentiation, 001 ← Multiplication, 010 ← Division,

011 ← Addition & 100 ← Subtraction

Here addition and subtraction are handled similarly, only the sign bit is flipped for subtraction.

If the code is 0xff then it denotes the end of expression. For other cases, the last 6 bits denote address in LUT in order. All these decodings are done using the decoder modules or during term accumulation.

Stack: A stack has been implemented using memory for evaluation of the poxtfix expressions. It consists of 2 immediate registers and stack memory. The immediate registers stores the top 2 values of the stack. Whenever an operation is detected in the expression, the operation is done on the 2 immediate registers and the result is stored in the 1st register. The other register is filled with the value popped from the top of the stack memory. Whenever a value is to be pushed into the stack, the value from the 2nd register is pushed to the memory, the value of 1st register to the 2nd register and the new value to the 1st register. And vice-versa for pop.

5.3.2 Arithmetic and Logic Unit (ALU)

The Arithmetic and Logical Unit (ALU) has been appropriately parametrised to enable it to function at any given level of floating point precision (default format is IEEE-754). It consists of the following datapath units -

1. **Addition Module** - This module performs floating point addition. The complete datapath has been split into 4 stages keeping in mind the target clock frequency 100 MHz. Current design contains 2 units of Addition Module.

2. **Multiplication Module** - This module performs floating point multiplication. The complete multiplication datapath has been split into 5 stages in order to achieve the target clock frequency. Moreover, if required, the module can be configured to function in pipeline mode. Current design contains 1 unit of Multiplication Module.

3. **Exponentiation Module** - This module evaluates the value of an input raised to an integer exponent. It utilises an instance of the multiplication module. Current design contains 1 unit of Exponentiation Module.

4. **Division Module** - This module performs floating point division. The complete datapath has been split into multiple stages keeping in mind the target clock frequency 100 MHz. However, since, the division operation requires significantly more clock cycles compared to other operations, the design has been optimised to reduce the number of division oeration to minimum. Current design contains 1 unit of Division Module.

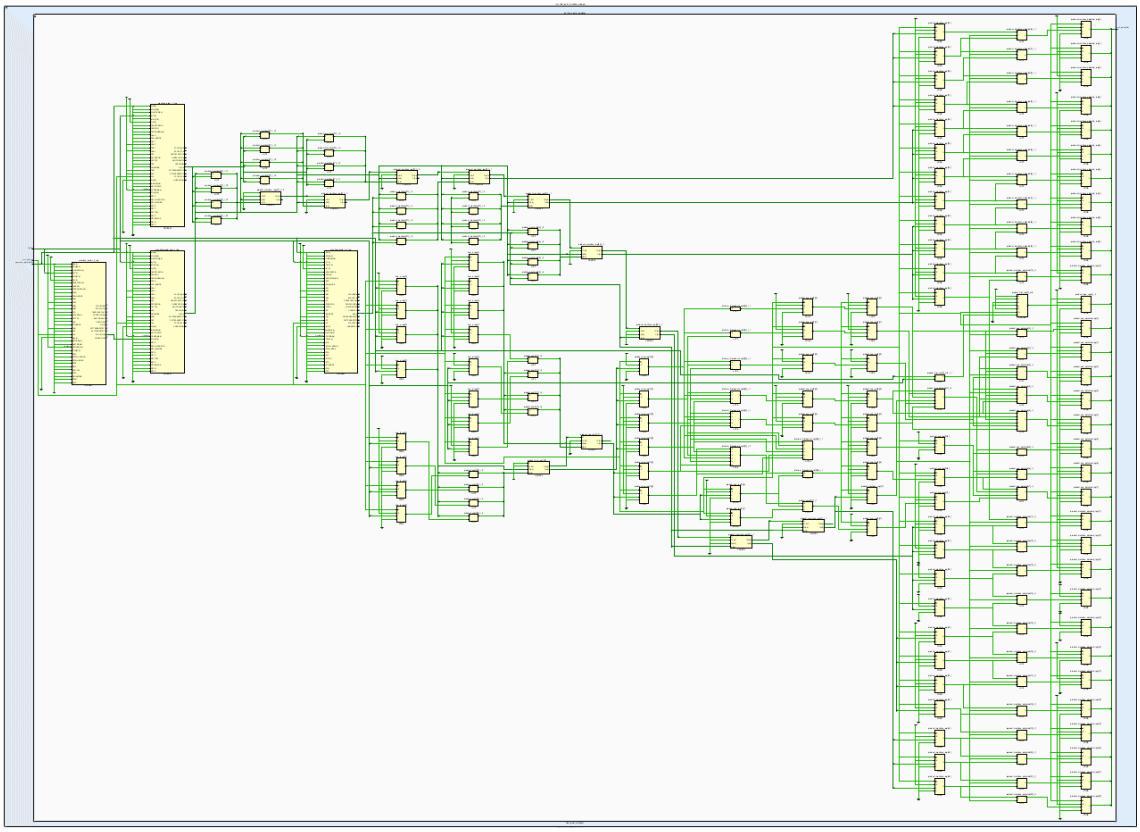


Figure 5.14: Schematic of multiplier module

5.3.3 Control Unit

The control unit of the implementation is essentially a Finite State Machine (FSM) which performs the required tasks in a well-defined sequence. In order to achieve an efficient implementation, the complete FSM has been factored into simpler state machines. A top-level FSM serves as control for these simpler FSMs. Basic description for these state machines is given as follows -

Top-Level State Machine

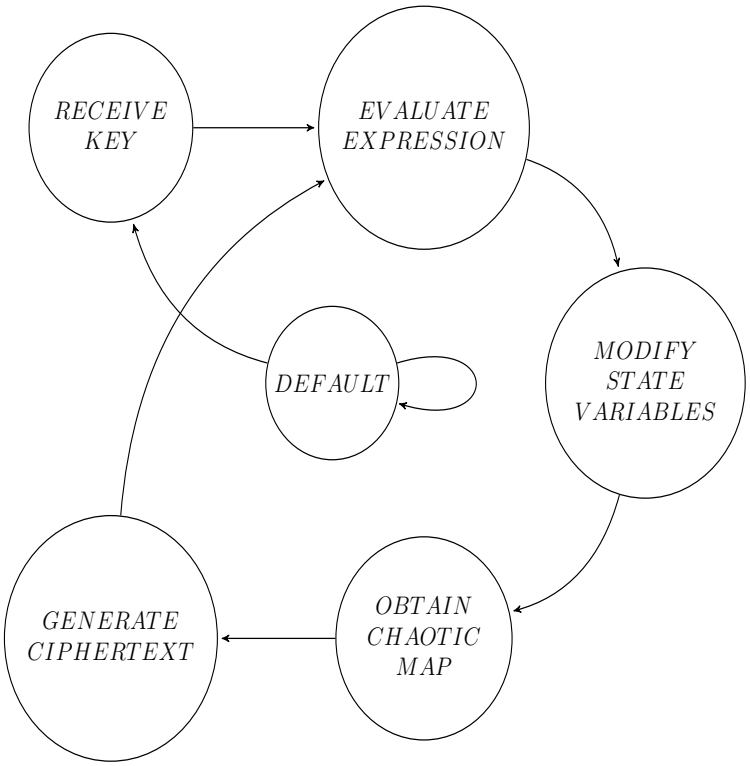


Figure 5.15: Simplified form of Top-Level FSM

- DEFAULT : Initial state of FSM after reset. Involves no processing.
- RECEIVE_KEY : Involves receiving the encryption or decryption key values through USB or Ethernet interface.
- EVALUATE_EXPRESSION : Evaluates the state-space expression using the given initial conditions for the chaotic dynamic system. The processing begins in state STATE_EXP_EVAL_BEGIN and the FSM waits for the complete signal (exp_evaluator_data_ready) in state STATE_EXP_EVAL_WAIT.
- MODIFY_STATE_VARIABLES : Updates the state variables based on the values obtained in previous state. Involves updating the state variables using the values cal-

culated in previous states.(theta_dot(t+1) = theta_dot(t) + theta_dot_dot(t)*deltaT, theta(t+1) = theta(t) + theta_dot(t)*deltaT). Involves using the mult_add module to perform these operations. This operation takes place across states STATE_POST_PROCESS_1, STATE_POST_PROCESS_2, STATE_MULT_ADD_WAIT.

- OBTAIN_CHAOTIC_MAP : Evaluates a chaotic map through a linear combination of values of all state variable. Calculates the linear combination of theta, theta_dot and theta_dot_dot. Currently, the value of theta itself corresponds to the chaotic map. Further, it calculates the intervals in which the value lies and adds it to corresponding addrss in LUT. This is done in state STATE_POST_PROCESS_3, STATE_POST_PROCESS_3_WAIT, STATE_ENCRYPT, STATE_ENCRYPT_STORE.
- GENERATE_CIPHERTEXT : Generates the ciphertext for a particular plaintext string using the generated chaotic map. Uses LUT to convert plaintext to ciphertext.

Evaluate State-Space Expression

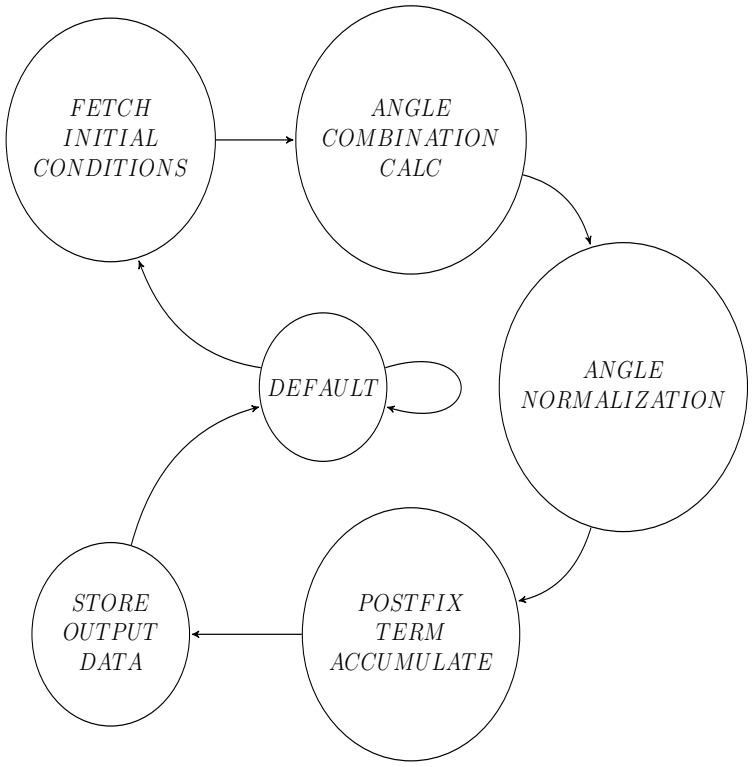


Figure 5.16: Simplified FSM of State-Space solver

- DEFAULT : Initial state of FSM after reset. Involves no processing.
- FETCH_INITIAL_CONDITIONS : Involves fetching the required initial conditions from Block-RAM. This includes STATE_FETCH_INIT_VAL. Reads the theta and

theta_dot values from the BRAM mem_state_var in the top module.

- ANGLE_COMBINATION_CALC : Create a cache containing frequently used linear combinations of state variables. Spans across STATE_ANGLE_COMB_START (trigger the module) and STATE_ANGLE_COMB_WAIT (wait for module to complete execution).
- ANGLE_NORMALIZATION : Normalize the trigonometric inputs to the range $[-\pi, \pi]$. Spans across STATE_NORM_ANGLE_START (trigger the module) and STATE_NORM_ANGLE_WAIT (wait for module to complete execution).
- POSTFIX_TERM_ACCUMULATE : Evaluates the state-space expression stored in postfix form. Spans across STATE_TERM_ACC_START (trigger the module) and STATE_TERM_ACC_WAIT (wait for module to complete execution).
- STORE_OUTPUT_DATA : Store the values of state-space expression obtained in previous state to a Block-RAM .

Postfix Expression Evaluation

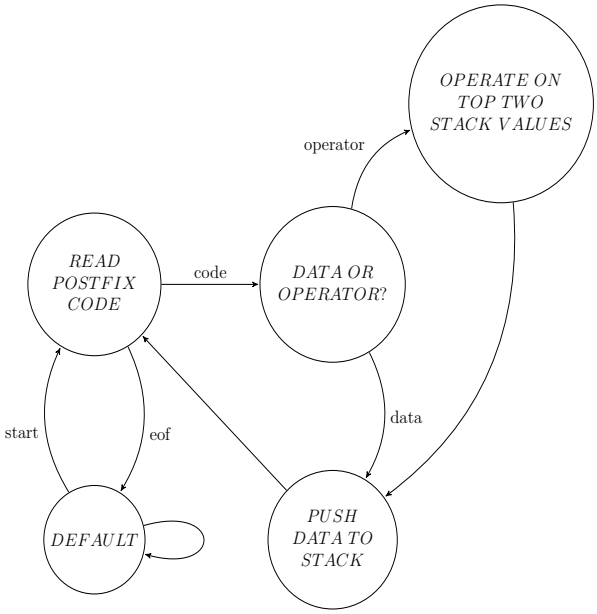


Figure 5.17: Simplified FSM for Postfix Evaluator

- DEFAULT : Initial state of FSM after reset. Involves no processing.
- READ_POSTFIX_CODE : Fetch the code for the next term of postfix expression from rams_sp_rom_pfX(where X is 1/2/3).

- DATA OR OPERATOR : Check if the code fetched corresponds to a data value or an operator. Takes place in STATE_POSTFIX_TERM_READ.
- OPERATE_ON_TOP_TWO_STACK_VALUES : Perform the corresponding operation of the two data values at stored at top of the stack. Spans across STATE_OPN_START (trigger the module) and STATE_OPN_WAIT (wait for module to complete execution).
- PUSH_DATA_TO_STACK : Push the data value in postfix expression or the result of ALU operation onto the stack. STATE_PUSH_DATA_DECODED pushes a data value from memory to stack. STATE_PUSH_DATA_ALU pushes the result of operation to stack.

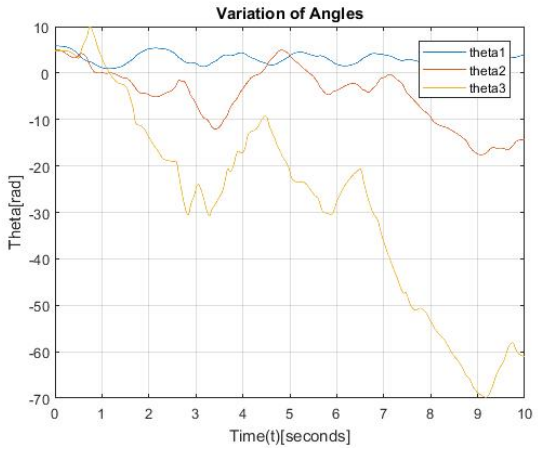
5.4 Analysis

5.4.1 General Properties of the Cryptosystem

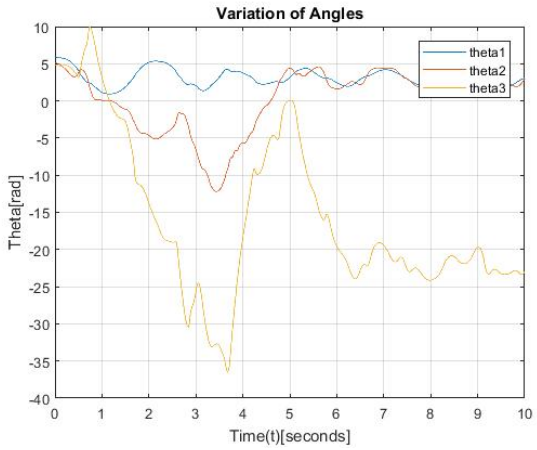
Test for Chaos

There are some essential requirements that need to be obeyed by any chaos based cryptosystem. These requirements include:

1. Sensitivity to Parametric values: It refers to the fact that a small perturbation in one of the system parameters is enough to make two trajectories, starting at the same initial point, diverge at exponential rate.
2. Sensitivity to Initial Condition: Two trajectories starting at two different but arbitrarily close initial points diverge from each other at an exponential rate.
3. Ergodicity: Almost every trajectory tends toward an invariant distribution which is independent of the initial conditions, and almost every trajectory will eventually visit any arbitrary interval of arbitrary size.

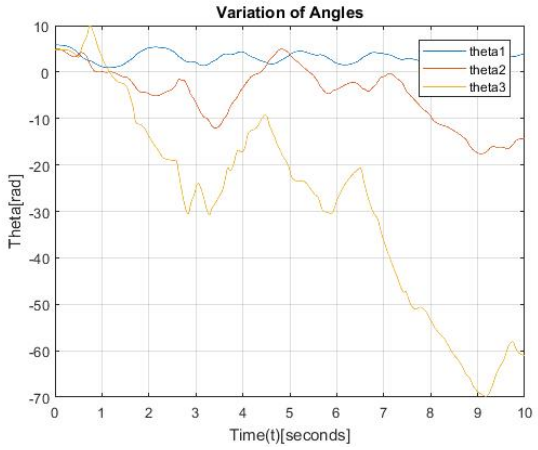


(a) Variation of θ for $m_1 = 0.2944$

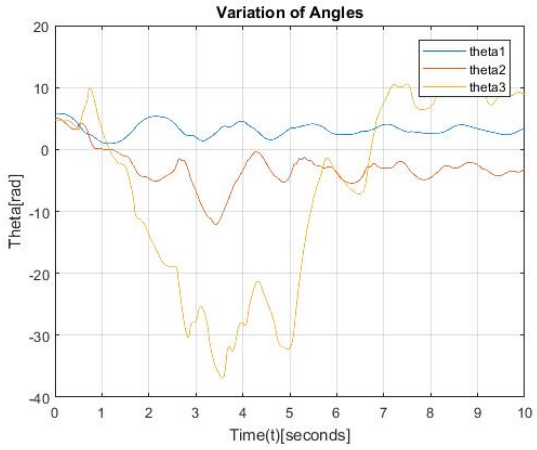


(b) Variation of θ for $m_1 = 0.294401$

Figure 5.18: Plot showing Sensitivity to Parameter value m with $\Delta m = 10^{-6}$



(a) Variation of θ for $\theta_1(0) = -0.4603$



(b) Variation of θ for $\theta_1(0) = -0.460301$

Figure 5.19: Plot showing Sensitivity to Intital condition of $\theta_1$with $\Delta\theta_1 = 10^{-6}$

The proposed cryptosystem also holds such properties. The three qualities mentioned earlier are clearly evident from the plots and graphs shown Fig. 5.15 and Fig. 5.16. It is observed that keeping the parameters in one case and initial conditions in another case, constant leads to two completely different trajectories. Thus the map generated from the compound triple pendulum model possesses the essential chaotic properties to be employed in a general cryptosystem.

Collision Test

Collision resistance is a property of cryptographic algorithms which makes it difficult to find two inputs which are encrypted to the same output. It must be ensured that finding collisions must be kept as hard as some of the hard mathematical problems like integer

factorization or discrete logarithm. Collision resistance does not mean that no collisions exist, it simply means that they are very hard to find. Here, we show that the proposed algorithm is fully collision resistant i.e. no collisions exist.

Since the entire range of $\hat{\mathbf{y}}$ are partitioned according to the number of characters(say 256 i.e. $[\hat{y}_{min} + k\epsilon, \hat{y}_{min} + (k+1)\epsilon)$ for $k = 0, 1, ..., 255$), different characters are mappped to points in different intervals. Hence two different inputs having different characters are completely mapped to different intervals. So the encryption scheme is completely free of collisions.

Test for Randomness

For testing randomness of the chaotic map, necessary statistical tests were applied. These statistical tests are generally employed for testing randomness in pseudo random number generators. Here, Diehard random number generator test suite has been used to do such tests. Developed by George Marsaglia, Diehard is a battery of tests which are used to determine the quality of PRNGs. The list of tests performed on the generated chaotic map data and the corresponding results obtained are shown in Table 5.2.

It is observed that in most of the tests, the chaotic map generator performs well. This can be attributed largely due to the non-linear dynamics of the map.

5.4.2 Complexity Analysis

In this report, we give an overview of the computational complexity of the algorithm both on software as well as hardware level.

Setup Complexity

Firstly, the computation of the state variable (θ) values for a time duration t and time step ΔT requires N iterations where $N = \frac{T}{\Delta t}$. In each iteration only 6 equations are solved for the 6 state variables $(\theta_1, \theta_2, \theta_3, \dot{\theta_1}, \dot{\theta_2}, \dot{\theta_3})$ each in $O(1)$ time. Thus the time required for generating the map is asymptotically $O(N)$. Here space complexity is also $O(N)$ for storing the state variable values. Secondly, Baptista type patitioning takes constant time if the range of the variables are already computed in the previous step. For assigning the variable values to the corresponding intervals, $O(N)$ time is required for each value. Thirdly, for encryption of a plaintext having M characters, there would be M iterations and in each iteration the encrypted value is randomly selected from the computed interval which takes constant time. So the time complexity for setup of the encryption module is $O(N)$ and $O(M)$ is the time complexity of processing each plaintext. Space complexity

Table 5.2: Dieharder Test

test_name	ntup	tsamples	psamples	p-value	Assessment
diehard_birthdays	0	100	100	0.98409924	PASSED
diehard_operm5	0	1000000	100	0.02570841	PASSED
diehard_rank_32x32	0	40000	100	0.30314396	PASSED
diehard_rank_6x8	0	100000	100	0.07586247	PASSED
diehard_bitstream	0	2097152	100	0.83264505	PASSED
diehard_opso	0	2097152	100	0.93701062	PASSED
diehard_oqso	0	2097152	100	0.63759752	PASSED
diehard_dna	0	2097152	100	0.73795350	PASSED
diehard_count_1s_str	0	256000	100	0.77268562	PASSED
diehard_count_1s_byt	0	256000	100	0.55542008	PASSED
diehard_parking_lot	0	12000	100	0.70977074	PASSED
diehard_2dsphere	2	8000	100	0.58428028	PASSED
diehard_3dsphere	3	4000	100	0.86446205	PASSED
diehard_squeeze	0	100000	100	0.19422339	PASSED
diehard_sums	0	100	100	0.33263813	PASSED
diehard_runs	0	100000	100	0.50609448	PASSED
diehard_craps	0	200000	100	0.36377661	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.66121444	PASSED
sts_monobit	1	100000	100	0.31074688	PASSED
sts_runs	2	100000	100	0.98606705	PASSED
sts_serial	1	100000	100	0.23629469	PASSED
sts_serial	2	100000	100	0.85410639	PASSED
sts_serial	8	100000	100	0.11131795	PASSED
sts_serial	16	100000	100	0.83380616	PASSED
rgb_bitdist	1	100000	100	0.45061356	PASSED
rgb_bitdist	6	100000	100	0.53111009	PASSED
rgb_bitdist	12	100000	100	0.64865649	PASSED
rgb_minimum_distance	2	10000	1000	0.41755433	PASSED
rgb_minimum_distance	4	10000	1000	0.11298477	PASSED
rgb_permutations	2	100000	100	0.79654377	PASSED
rgb_permutations	4	100000	100	0.85371565	PASSED
rgb_lagged_sum	0	1000000	100	0.92039997	PASSED
rgb_lagged_sum	16	1000000	100	0.78219023	PASSED
rgb_lagged_sum	32	1000000	100	0.64955712	PASSED
rgb_kstest_test	0	10000	1000	0.58005862	PASSED
dab_bytedistrib	0	51200000	1	0.01089458	PASSED
dab_dct	256	50000	1	0.06122408	PASSED
dab_filltree	32	15000000	1	0.12524761	PASSED
dab_filltree2	0	5000000	1	0.50605989	PASSED
dab_monobit2	12	65000000	1	0.27940682	PASSED

is still $O(N)$. Similarly, for decryption time complexity is also $O(N)$ and $O(M)$ for setup and processing respectively.

Information Rate: The information rate of any cryptosysytem is defined as the ratio of the size of plaintext to that of the cipher text. In our cryptosystem, we have

$$R = \frac{plaintext\ size}{ciphertext\ size} = \frac{8 * M}{\lceil \log_2 N \rceil * M} = \frac{8}{\lceil \log_2 N \rceil} \tag{5.7}$$

Hardware Resource Utilization

The following (Fig. 5.17) shows the resource utilization of the complete RTL implementation after synthesis using XST tool. The maximum resources used is in the storage of state variable values in the LUT. It is only about 1.3% of the total fgpa memory and hence the memory usage is quite low. It can be seen that 1726 LUTs and 1883 Flip-Flops are required. Also, due to target device being FPGA, instances of DSP48 have been used.

Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	1726	134600	1.28
LUTRAM	49	46200	0.11
FF	1883	269200	0.70
BRAM	2.50	365	0.68
DSP	4	740	0.54
IO	2	285	0.70
BUFG	3	32	9.38

Figure 5.20: Table showing Resource Utilization

5.4.3 Reason for being Quantum-Safe

The argument in regards to the quantum-safe property of this algorithm is due to its innate chaotic nature. Traditional Shor's and Glover's algorithms cannot break this system as they deal with a different paradigm of hard problems like integer factorization and discrete logarithm. Also the highly non-linear and interdependent nature of the dynamical system makes it difficult for even the quantum algorithms to do brute force attacks without the total or partial knowledge of key. Even with partial knowledge of key values, it is useless to do brute force attacks as the chaotic maps are highly sensitive to key values. The key size is about $21 \times (32bits) = 672bits$ for single precision arithmetic which is quite large enough. All these reasons direct us to be confidently stand by this algorithms and call it quantum computing immune.

Chapter 6

Results & Conclusion

The entire top level module was simulated in Xilinx Vivado and the synthesis and working of each of the modules were verified. The functions of encryption & decryption modules were also analysed. The results obtained in the hardware simulation matches with the state variable values simulated in MATLAB within some reasonable accuracy.

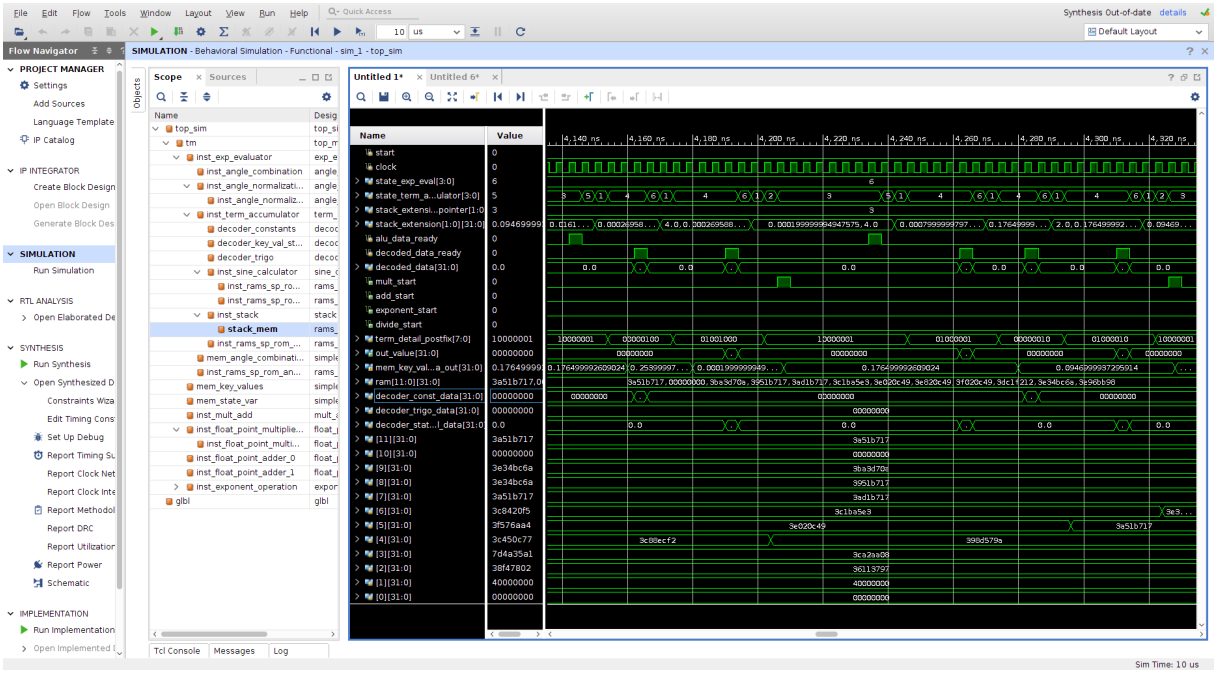


Figure 6.1: Simulation of hardware implementation in Xilinx Vivado

From the analysis presented in section 5.5 and the results obtained from the hardware simulation of the FPGA implementation, we can conclude that this design can serve as a good prototype of a chaos based cryptographic hardware which is immune against brute force quantum computing attacks.

Chapter 7

Future Work

This report so far discussed the methods for encryption-decryption based on non-linear chaotic map and its utility in post-quantum cryptography. Through a number of simulations, we can conclude that this cryptosystem can provide the desired level of chaos and security. Moreover, this cryptosystem also fulfills the basic requirements of a cryptosystem defined by Shannon including diffusion and confusion.

Algorithmic Aspect

The main improvement aspect of this work lies in the extension of this algorithm for asymmetric key cryptography. This can be easily done by introducing safe key manipulation techniques which would enable both parties to retrieve the key values. Once the key is obtained, one can proceed according to steps followed for the symmetric algorithm. Another improvement is introducing permutation map between characters and Baptista type partitioned intervals. Also instead of taking a single state variable value for chaotic map, all three state variable values can be combined through a one-one function to give a better level of encryption. For better validation of the cryptosystem in regards to the quantum-computing aspect, the cryptosystem must be tested against practical quantum computing algorithms for breaking cryptosystems which is beyond our current scope. However, this cryptosystem might serve well practically in terms of both speed and cost.

Hardware Implementation Aspect

Although a complete implementation of the cryptographic algorithm has been presented, there remains a huge scope for improvement. This involves efficient caching of results of operations on constants. Implementation of such a cache can result in significant reduction in the number of clock cycles required. Furthermore, few modifications might be made to enable design scale easily according to the amount of resources at disposal.

Bibliography

[1] Toshiki Habutsu, Yoshifumi Nishio, Iwao Sasase, Shinsaku Mori, "A Secret Key Cryptosystem by Iterating a Chaotic Map", *LNCS, vol 547*, EUROCRYPT 1991.

[2] T. Yang, "A survey of chaotic secure communication systems", *Int. J. Comput. Cogn. 2004.*

[3] M. Hasler, "Synchronization of chaotic systems and transmission of information", *Int. J. Bifursc. Chaos, vol. 8, no. 4, Apr. 1998.*

[4] Ljupco Kocarev, "Chaos-Based Cryptography: A Brief Overview", *IEEE Circuits and Systems Magazine 1(3):6 - 21 , Sept 2002.*

[5] R. Schmitz, "Symmetric ciphers based on two-dimensional chaotic maps", *J. Franklin Inst., vol. 338, pp. 429-441, 2001.*

[6] D. E. Knuth, "The Art of Computer Programming.", *MA: Addison-Wesley, 1998, vol. 2.*

[7] L. Kocarev, J. Szczepanski, J. M. Amigo, and I. Tomosvski, "Chaos-based Cryptography: an overview", *International Symposium on Nonlinear Theory and its Applications (NOLTA2005), Bruges, Belgium, October 18-21, 2005.*

[8] R. Matthews, D. Wheeler, "Supercomputer investigations of a chaotic encryption algorithm", *Cryptologia XV, (1991) 140-152.*

[9] Zbigniew Kotulski, Janusz Szczepanski, "Discrete chaotic cryptography", *Ann. Physik 6 (1997) 381-394*

[10] Sofia Wichert, Konstantinos Fokianos and Korbinian Strimmer, "Identifying periodically expressed transcripts in microarray time series data", *Bioinformatics. 20 1, (2004), 5-20.*

[11] Daniel J. Bernstein, Johannes Buchmann, Erik Dahmen, "Post-Quantum Cryptography"*Springer, Berlin, 2009.*

[12] Peter W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", *SIAM J.Sci.Statist.Comput. 26 (1997) 1484*

[13] Andre E. Botha,Guoyuan Qi, "Analysis of the Triple Pendulum as a Hyperchaotic System"*Chaotic Modeling and Simulation (CMSIM) 2: 297304, 2013.*

[14] Shujun Li, Guanrong Chen, Kwok-Wo Wong, Xuanqin Mou, Yuanlong Cai, "Baptista-type chaotic cryptosystems: Problems and countermeasures", *Physics Letters A, 332(5-6):368-375, 2004*

[15] Auqib Hamid Lone, Prof. Moin Uddin, "Common Attacks on RSA and its Variants with Possible Countermeasures", *International Journal of Emerging Research in Management & Technology ISSN: 2278-9359 (Volume-5, Issue-5), May 2016*