Design and Implementation of Lattice and Chaos based Post-Quantum Cryptography Algorithms

A

Thesis submitted

for the award of the degree of

DOCTOR OF PHILOSOPHY

By

BIKRAM PAUL

DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781 039, ASSAM, INDIA

MARCH 2022

Certificate

This is to certify that the thesis entitled "**Design and Implementation of Lattice and Chaos Based Post-Quantum Cryptography Algorithms**", submitted by **BIKRAM PAUL** (146102038), a research scholar in the *Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati*, for the award of the degree of **Doctor of Philosophy**, is a record of an original research work carried out by him under my supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the institute and in my opinion has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Dated: 31st March 2022
Guwahati, Assam, India

Dr. Gaurav Trivedi
Associate Professor
Dept. of Electronics and Electrical Engg.
Indian Institute of Technology Guwahati
Guwahati - 781 039, Assam, India.

ॐ त्र्यम्बकं यजामहे सुगन्धिं पुष्टिवर्धनम् |
उर्वारुकमिव बन्धनान् मृत्योर्मुक्षीय मामृतात् | |

ऋग्वेद ७.५९.१२
Rigved 7.59.12

Dedicated to the **Almighty**,

&

to my **parents** for their blessings, love and support.

"ॐ नमः शिवाय"

Acknowledgements

Meagre expression conceals major part of my cardiac gratitude and indebtedness for my research supervisor, Dr. Gaurav Trivedi, a distinguished conceptualist, prolific trailblazer, not out of sheer imposition for mere duty's sake; but for his heraldic highlight, extolling suggestions, constant guidance, timely advice, scrupulous scrutiny and explicit encouragement.

I would like to extend my profound sense of gratitude to Prof. H. B. Nemade, Prof. S. R. Ahamed and Dr. A. Sahu for thorough inspiration and valuable suggestions. I am also grateful to faculty members and the office staffs of the Department of Electronics and Electrical Engineering, IIT Guwahati, for their help in carrying out this research work.

I also express my sincere and deep sense of gratitude to Dr. S. Krishnaswamy for his scholastic advices, benevolent and many fold helps in connection with this research project.

My sincere thanks go to Dr. Satyabrata Dash for his assistance in completion of the thesis. I am thankful to Dr. Swati Shukla, Mr. Praveen Tiwari, Ms. Meenali Janveja, Dr. Sunil Dutt, Mrs. Sushree Sila P. Goswami, Mr. Abhishek Agrawal, Mr. Souradip Pal, Mr. Tarun Kumar Yadav, Mr. Balbir Singh, Dr. Uddipana Dowerah, Mr. Shubhadip Poria, Ms. Parmita Roy, Mr. Ananda YR, Dr. Partha Das and Mr. Mridul Goswami for their ungrudging help and sympathetic encouragement.

Last but most important are my parents and other family members whose blessings and love made my path of success. I attribute this achievement to my father, mother and sister for their constant support, silent prayers for my success and moreover, making me stand in this position.

BIKRAM PAUL

Abstract

The achievements we have made in the field of varied technology have largely fueled modern human growth. Data and digital information based innovations are playing a major role in the accelerated growth. Personal and other important secret data are considered vital asset since historical time and have become prime focus for the current information and digital generation. Various rudimentary data encryption methodologies, such as, Caesar cipher, Scytale cipher, Vigenère cipher, Steganography, Enigma machine were introduced in the past. With the invention of digital computers, to introduce greater resilience in security newer symmetric (e.g. DES, AES etc.) and asymmetric (e.g. RSA, ECC etc.) cryptography schemes are formulated in late ′90s, which are acting as the primary protector of data against adversary and unwanted intruders.

The rapid advancements of powerful and efficient hardware, data processing and computation are getting stronger, and breaking encryption has become simpler. To deal with new and stronger attacks, larger keys are being deploying to existing algorithms and more advance encryption schemes are getting developed. Moreover, it is theoretically proven that a sufficiently large quantum computer can easily break encryption based on asymmetric key cryptosystems employing Shor's algorithm and is able to exponentially shorten the time to determine key in symmetric cryptosystems using Grover's algorithm. According to Moscas inequality, there is a technical race persists between developing stronger encryption and actual scalable quantum computer. A security system that is completely secure against all types of attacks, is practically nearly impossible to implement. Therefore, in order to maintain a leading position in the race against advanced and efficient sophisticated quantum computing, better and stronger data security methodologies are of prime focus of the researchers.

A fully quantum cryptosystem can resists attacks generated from a quantum computer, which employs quantum channels and qubit as the medium of communication and computational units, respectively. To implement fully quantum security algorithms, massive changes need to be incorporated

within current communication infrastructure, which is not feasible in the recent times. Therefore, we need to adopt an alternative path to design cryptography algorithms called as post quantum cryptography (PQC) algorithms, which can overcome the vulnerability of classical as well as quantum computing based crypto attacks. Although, several PQC algorithms are developed recently, but due to hardware resource constraints, these algorithms were never deployed in any major cryptography applications. Thus, in the proposed work, we aim to resolve these issues.

In order to develop PQC algorithms, building blocks, such as, custom Pseudo-Random Number Generators (PRNGs) and polynomial modulo multiplier of finite field arithmetic are realized on FPGA platform. We propose two PRNGs in this thesis; BluXor for general purpose applications and MPCG for low-power IoT applications, which are inspired from Blum-Blum-Shub (BBS), Xorshift and Permuted Congruential PRNGs. The dynamic power consumption of the proposed PRNGs is $17mW$ at $48.31Mhz$ and $16mW$ at $42.90Mhz$ while generating 4.83×10^7 and 4.29×10^7 random 32-bit numbers per-second, respectively. The proposed PRNGs pass all the benchmark tests of standard NIST-SP $800-22$ and Diehard battery suite benchmark test for the analysis of randomness quality. These proposed PRNGs are further used during implementation cryptosystems.

Thereafter, a tensor based novel modulo multiplication method for multivariate polynomials over $GF(2^m)$ is realized, which is an essential building block while designing lattice LWE cryptosystem. The proposed method utilizes 6% lower resources, 6.5× less power consumption and achieves more than 6× speedup with respect to other contemporary single variable polynomial multiplication implementations. Its total power consumption is estimated to be approximately $0.2W$ and $0.492W$ at 100 MHz and 197 MHz while implement on Zynq-7000-Z020 and Artix-7-200T FPGA board, respectively. For multivariate polynomial of 9 variables having maximum degree of 128 for each variable, chip area, power and delay estimated for the proposed multiplier are $50113.9\mu m^2$, $1.485\ W$ and $8.812\ ns$ during its ASIC realizing. The computational complexity of single variable and multivariate polynomial multiplications are $O(n)$ and $O(np)$, respectively, where n is the maximum degree of a polynomial having p variables.

Finally, employing the above building blocks two PQC schemes, chaotic triple pendulum based cryptosystem and polynomial lattice based cryptosystem, are developed. A novel nonlinear chaos generator scheme is derived from a mechanical model depicting nonlinear dynamics of a triple pendulum physical system and a its effectiveness is validated against various standard tests, such as Lyapunov ex-

ponents test, bifurcation diagrams, sensitivity to parametric and to initial values, ergodicity, collision test, NIST randomness test etc. The proposed chaotic generator is utilized to develop a symmetric key encryption scheme, which is secure against various crypto attacks. The cryptosystem is verified through an FPGA implementation to assess its usage in low power high throughput applications, where its power consumption, resource utilization and throughput are 1.785×, 1.825× and 2.396× better as compared to other known contemporary methods. Further, it is observed that the proposed design can work efficiently with various wide range of applications. The average power and area of its ASIC implementation at 180-nm technology are 61.8836 mW and 0.20374 mm^2 at 250 MHz, respectively while realizing with SCL's 180-nm CMOS technology node. Moreover, the stable power consumption profile is generated by varying the operating frequency ranging from 1 KHz to 500 MHz.

Further, two lattice based fully homomorphic encryption (FHE) schemes are designed and implemented using software hardware co-design (on ARM-SoC and FPGA) approach, which provide strong resistance against quantum as well as classical computer based adversary security attacks. It is to mention that all the proposed algorithms and building blocks are compared with the best known contemporary methodologies. The proposed first scheme takes $\{6.88\mu s, 16.18\mu s, 2.84\mu s\}$ and $\{1.65\mu s, 4.38\mu s, 0.72\mu s\}$ for encryption, decryption and recryption using Zedboard and Virtex-7 FPGA platforms, respectively. Similarly, the second scheme consumes $\{1.44\mu s, 0.96\mu s, 0.22\mu s\}$ and $\{0.35\mu s, 0.225\mu s, 0.055\mu s\}$, respectively for encryption, decryption and recryption on the above mentioned FPGA platforms, respectively. The best implementation of the proposed FHE scheme is found to be 52.71× more resource efficient than another contemporary method based on BGV RLWE. The cumulative throughput of the proposed scheme is 6523.752 Mbps exhibiting 43% and 29% improvement over non-pipelined and pipelined BGV RLWE schemes, respectively, on Virtex-7 FPGA board. Similarly, the cumulative delay of our proposed FHE scheme developed for IoT enabled hardware applications is $0.63\mu s$, which is 23× lower than the BGV RLWE based FHE schemes. Since the proposed methods are having lowest area and delay footprints among the other contemporary schemes, they are the most suitable candidates for providing edge security to enabled IoT devices and other applications.

Contents

List of Figures

List of Tables

List of Symbols

$\mathbb{R}$	Set of real numbers
$\mathbb{Z}$	Set of integers
$\mathbb{Q}$	Set of rational numbers
$\mathbb{N}$	Set of natural numbers
$\{\mathbf{A}, \mathbf{B}, \dots\}$	Input matrices
$\mathbf{I}_n$	Identity matrix of size $n \times n$
$\mathbb{F}_p$	A finite field of order p, a prime number
$\mathbb{Z}_p$	Set of integers in the interval $(-\frac{p}{2}, \frac{p}{2}]$
$\{a, b, \dots\}$;$\{\mathbf{a}, \mathbf{b}, \dots\}$	Scalar; vector elements of a matrix
$\mathcal{A}$	Set of $\{a, b, \dots\}$ and $\{\mathbf{a}, \mathbf{b}, \dots\}$
$\lfloor x \rfloor$, $\lceil x \rceil$, $\lfloor x \rceil$	Rounding of x down, up and to nearest integer
$\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$	Inner product of the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$
$x \xleftarrow{\$} \mathcal{S}$	x is sampled from a set $\mathcal{S}$ of uniform random integers
$\mathbb{F}_p[x_1, \dots, x_l]_r$	Set of polynomials with degree $\leq r$; $r \in \mathbb{N}$
$GF(2^m)$	Binary Galois Field
$\mathcal{I}$	Ideal polynomials ring
$\mathcal{I}_{\leq d}$	Ideal polynomials ring of degree d
$\mathcal{T}_x$	Tensor matrix of polynomial coefficient x
$\wedge_s$	Diagonal coefficient matrix
$\mathcal{B}_{\mathcal{T}}$	Bilinear map of $\mathcal{T}$
$\langle f_i \rangle$	Elements of polynomial ring f_i
$A_A,\ A_X$	Area of 2-inpur AND, XOR gate
$T_A,\ T_X$	Propagation delay of 2-inpur AND, XOR gate
$\phi(m_i, m_j)$	Homomorphic operation on message m_i, m_j

$\mathcal{L}$	Lagrangian operator
$\dot{\theta}_i, \ddot{\theta}_i$	Angular velocity, acceleration
Δt	Time step
$P_{xx}f$	Power Spectral Density of a signal of f
$half(n)$	Half-Precision Floating Point n
$\pi^{-1}(k)$	Permutation map
N_c	Number of characters for encryption
$\mathcal{A}$	PPT adversary model
χ	Probability Distribution
$\mathcal{N}$	Noise distribution ($\mathbf{0}$,χ) of the vectorspace $\mathcal{V}$
$\mathcal{S}_{\mathcal{I}\leq r}$	Secret key in ideal $\mathcal{I} \leq r$

1

Introduction

Contents

1.1 Data Security: Evolution of Cryptography

Cryptography is derived from two Greek words "kryptos", which means "secret", and "graphos", which means "writing"; thus it literally means "secret writing". Cryptography is the science of secret writing, with the goal of securing data, such that, only the intended recipients can decrypt and read it. From the time it was probably first used by the Egyptians of ancient Indian culture until now, cryptography has evolved manyfolds. It is unthinkable nowadays to have private or secret material without encryption. As the current peer-to-peer communication has shifted mostly through Internet, we now have sensitive information or data stored digitally all over the web. Due to their vulnerability to interception, transmitted, received or stored data rely entirely on the encryption to keep their communication private. Therefore, cryptography methods employed have to be strong enough to check various cryptanalytic attacks, so that data is protected against any malicious adversary. Academicians have segmented cryptography history and development into three chronological time periods as given below.

(i) **Ancient** (until 1918): A few example of ancient cryposystems are Caesar cipher [5], Scytale cipher [6], Vigenère cipher [7], Atbash cipher [8], Steganography [9] etc., which are mostly based on substitution, repetitions and shifting operations.

(ii) **Technical** (1919 – 1975): During World War I and World War II, mechanical machinery were utilised to execute encryption, marking the start of cryptography's technical age. Hagelin machine [10], Enigma machine [11], Biuro Szyfrów cipher [12], Lorenz cipher [13] etc. were being used to encrypt and decrypt messages transmitted and received, respectively.

(iii) **Paradoxical** (from 1976−now): After the invention of computers and digital devices, data was started getting stored in it in the digital form. Therefore, various sophisticated algorithm driven cryptosystems were designed, such as, RSA [14], ECC [15], DES [16], DiffieHellman [16], AES [17] etc.

Encryption and decryption are two complementary functions of a cryptography scheme. Encryption transforms plaintext into an incomprehensible form depend on the input key. Decryption, on the other hand, works with ciphertext to retrieve the original message with the help of either the same key (symmetric encryption) or a different key (asymmetric encryption). Cryptanalysis, on the other

hand, is the reversed process of an encryption, which focused only to overcome security without pre-knowledge of the encryption key. Cryptography and cryptanalysis are two separate scientific pursuits that compete with one another, with the first attempting to conceal a secret, while the latter attempts to reveal it. In this regard, Shannon's theory of "Confusion" and "Diffusion", and Kerchoff's principle are very important. In a 1949 paper [18], Claude E. Shannon explained two aspects of encryption that thwart frequency analysis: "confusion" and "diffusion". Confusion is used to obscure the relationship between the plaintext and the ciphertext, and it can be accomplished via a sophisticated substitution algorithm. Diffusion, on the other hand, seeks to distribute the statistics throughout the message to prevent cryptanalysts from exploiting any existing redundancy generated through repeated permutations. Auguste Kerchoff von Nieuwenhof, a Dutch linguist, mentioned in his 1883's book "La Cryptographie Militaire" [19], that the security of a cryptosystem must be entirely based on the secrecy of the key, not the algorithm's secrecy. This is known as the Kerchoff principle, and it is still relevant and useful while designing modern cryptography algorithms.

The gradual evolution of technology was not only limited to cryptography domain, but also various innovations has propelled the computation methodologies better and robust. Recent advancements in quantum computing encourage researchers to develop innovative solutions to accelerate parallel computations and execute powerful quantum algorithms. These rapid developments in the field of quantum computer have pushed the conventional classical cryptosystem on verge of hefty security threats. Most widely used symmetric and asymmetric cryptosystems, such as, RSA [14], ECC [15], DES [16], DiffieHellman [16], AES [17] etc. are vulnerable to quantum Shor's factoring algorithm and Grover's search algorithm operating on quantum computers. According to "Moscas inequality", $D + T \geq Q_c$ [20], there is a technical race persists between stronger encryption scheme and the development of actual multi qubit quantum computer. In the above relation, "D" is the total time to keep the data secured against quantum computer generated threat. "T" is the time required to develop theoretically unbreakable classical algorithm or to port the conventional security system from classical to quantum or post-quantum domain, and "Q_c" is time to develop sufficiently large quantum computer. To address these issues, there are two alternatives; the first one is quantum cryptography, which uses qubits in quantum channels and employ quantum algorithms to process them on quantum computers. This requires massive investment and also have to change entire data communication infrastructure at a large scale and is less feasible in the practical standpoint. The second approach is

to implement an alternative cryptosystem, named as post quantum cryptography (PQC), which can be realized by classical logic circuits and can easily be implemented on the classical systems. The main advantage of PQC algorithms are to provide strong resistance against classical as well as quantum computer generated crypto attacks. PQC algorithms are attack resilient against quantum computers because they are deigned based on specific mathematical problems, which are equally hard to solve on the classical as well as quantum computers.

1.2 Quantum Computing: Impact on Security

With the advancement in the field of engineering, we can harness quantum mechanical properties of photons, ions and other smaller particles. The basic concept of computation using quantum properties is not new, rather it is first introduced by Richard Feynman [21] in early 70's. Quantum Computing opens a completely new dimension of computations, which employs the fascinating laws of quantum mechanics and can solve compute intensive and possibly intractable problems with more ease as compared to existing ones. Conventional or classical computers only use a small subset of these possibilities. Quantum computers follow the quantum physical laws and gain huge boost in the computational power due to its ability to be in the multiple states at same time, which is called superposition of quantum particles. With the help of quantum superposition, quantum computer can perform multiple tasks using all possible permutations simultaneously in the same time a single operational unit takes in a conventional processing unit. The above mentioned property is utilized to analyze various computationally complex problems [22] in mathematics, physics, chemistry, biology and cryptography, which are practically intractable using classical computers.

Unlike deterministic classical computers, where algorithmic logic leads to microarchitecture consist of classical gates and registers, quantum computers (QC) compute information non-deterministically [23, 24]. Deterministic classical machines always produce same set of results for the same inputs using a particular algorithm, whereas, QC generates statistical and probabilistic output based on quantum interactions between qubits. Quantum gates follow quantum arithmetic [25–27], which can be expressed in the form of complex matrix transformations. Qubits can hold "0" or "1" or both due to superposition property and, due to that, 2^n distinct quantum states can be accommodated in n qubits. It is necessary to translate all the steps of quantum algorithm into appropriate physical qubits [28–30], which give massive parallelism in the computations.

Quantum computation challenges many intriguing problems and divides them in two separate categories, such as, tractable and intractable problems. Breaking a cryptosystem is the most significant example of quantum algorithms, which was previously believed that it was practically impossible to be cracked using classical counterparts [31]. Large prime number factoring and discrete logarithm problems [32], which are the cornerstone of many classical encryption algorithms, such as, RSA cryptosystem [33] and Diffie-Hellman key-exchange [34] can be solved by using Shor's quantum algorithm [35]. The main security features of conventional classical cryptography are based on the data encryption, symmetric or asymmetric key distribution, digital signatures, one way or trapdoor functions, pseudo-random number generation, zero-previous knowledge proofs etc. Strength of classical cryptography is measured on the basis of time to find solution of a mathematical problem, which has exponentially time computation complexity on a classical computer. Contrary to that, quantum computations use quantum Fourier transform (QFT) [36], which can perform exponentially faster as compared to classical computers. Table 1.1 exhibits current standing of various cryposystems against QC generated attacks or QC assist cryptanalysis.

Table 1.1: Current Status of Various Cryptosystems

Encryption Schemes	**Vulnerability Against Quantum Computers**
RSA Public key encryption	Cracked
Diffie-Hellman Key-exchange	Cracked
Elliptic curve cryptography	Cracked
Buchmann- Williams key-exchange	Cracked
Algebraically Homomorphic	Cracked
McEliece public key encryption	haven't broken yet
NTRU public key encryption	haven't broken yet
Chaos-based symmetric key encryption	haven't broken yet
Lattice-based public key encryption	Not broken till date

For example, factoring large number is long-studied mathematical problem and many algorithms have proposed to accelerate the process, such as, Lehman's method, Pollard's method, and Shanks's class group method [37]. After the invention of RSA public-key cryptosystem several sub-exponential time algorithms were invented to factor larger numbers. The number field sieve [38, 39] developed in 1989 is the best known classical algorithm for the factorization and has time complexity $exp\left(c\,(\log(n))^{\frac{1}{3}}\,(\log\log(n))^{\frac{2}{3}}\right)$, where c is a constant. It is to mention Shors algorithm on a quantum computer can solve this problem in polynomial time.

1.2.1 HSP Problems: Abelian and Non-Abelian Groups

Peter Shor's factoring and discrete log algorithms can solve efficiently many hidden subgroup problem (HSP) [40] with the help of quantum computer [41]. The HSP problems are defined as follows. Given a group and a function that is constant and distinct on co-sets of some unknown subgroup, find a set of generators for the subgroup. Fourier sampling, which is employed for computing Fourier transform and measuring, is the main algorithm for solving HSP when underlying group in finite and abelian. There exist some non-abelian cases, for example, an extension to Pells equation [42], which can also be solved by reducing them to the abelian case. The table below presents the current status of the abelian HSP.

Table 1.2: Current Status of Abelian HSPs [1]

Abelian Group G	**Related HSP Problem**	**Quantum Algorithm**
$\mathbb{Z}_2^n$	-	Susceptible
The integers $\mathbb{Z}$	Factoring	Susceptible
Finite groups	Discrete Log	Susceptible
The reals $\mathbb{R}$	Pell's equation	Susceptible
The reals $\mathbb{R}^c$, c is constant	Unit group of number field	Susceptible
The reals $\mathbb{R}^n$, n arbitrary	Unit group, general case	Resistant

If Fourier transform over abelian group G is efficient when hidden subgroup is normal and, if the computation of intersection of a set of representations can be done efficiently, then we can conclude that the non-abelian HSP [43] can also be solved. The following table shows the status of selected non-abelian HSPs.

Table 1.3: Current Status of Non-Abelian HSPs [1]

Nonabelian Group G	**Associated Problem**	**Quantum Algorithm**
Heisenberg group	-	Yes
$\mathbb{Z}_p^r \rtimes \mathbb{Z}_p$, r is constant		Yes
$\mathbb{Z}_p^n \rtimes \mathbb{Z}_n$, p is prime number		Yes
Extra special groups		Yes
Dihedral group $D_n = \mathbb{Z}_n \rtimes \mathbb{Z}_2$	Unique shortest lattice vector	sub-exponential-time [44]
Symmetric group S_n	Graph isomorphism	Evidence of hardness

We can compute a set of generators for an HSP in polynomial time, if finite group instances of an associated HSP is given [45]. There exists standard methods based on Shor's algorithm and Simon's algorithm for the above-mentioned problems. Tables 1.2 and 1.3 present the current position of various HSP problems against QC based cryptanalysis. For the nonabelian HSP cases, whether

a standard method can provide a solution or not, is determined by an underlying group. Although we can extract some information about the solution from the subgroups, but it may be difficult to compute the solutions from the samples. As we know that polynomially many co-sets can provide enough information to compute the subgroup or to convert a nonabelian HSP to a simpler problem.

1.2.2 Quantum Search Algorithms

Grovers search algorithm [46] is a quantum algorithm, which demonstrates how the qualities of quantum systems can be used to improve the lower runtime bounds of conventional classical algorithms. If a boolean function f has value s, whose structure or circuit is not known, Grover search algorithm can provide at least one pre-image [47]. This search algorithm can also find whether a particular given value is a pre-image of a given boolean function f or not. This property can be utilized to find collisions test for one or two functions. The quantum Grover search algorithm is presented in algorithm 1 for completeness.

Algorithm 1: Grover's Search Algorithm

1 **Input: Boolean function** $f : \mathbb{F}_2^n \to \mathbb{F}_2$ **given by the associated operator** $U_f : \mathbb{F}_2^n \times \mathbb{F}_2 \to \mathbb{F}_2^n \times \mathbb{F}_2^s : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$, **and** $M =$**card** $f^{-1}(1)$.

2 **Output: Some** $y \in \mathbb{F}_2^n$ **with** $f(y) = 1$.

1: If $M > \frac{3}{4}.2^n$**, then chose** y **randomly and uniformly from** $\mathbb{F}_2^n$ **and return**y**.**

2: Compute θ **satisfying** $\sin^2\theta = \frac{M}{2^n}$**, and set** $r \leftarrow \lfloor \frac{\pi}{4\theta} \rfloor$**.**

3: Transform

$$|0\rangle |1\rangle \xrightarrow{H^{\otimes(n+1)}} \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{F}_2^n} |x\rangle (|0\rangle - |1\rangle) \xrightarrow{G^r} \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{F}_2^n} \alpha_x |x\rangle (|0\rangle - |1\rangle),$$

where $G = U_f.(H^{\otimes(n+1)}(2|0\rangle\langle 0| - 1)H^{\otimes n}) \otimes$ **Id.**

4: Measure and output the first n **bits of the result.**

As we know, the collision test for a function is very much important for the cryptanalysis, i.e. searching for two arguments giving same function value. For this purpose, we assume a subset M as a domain of the boolean function f. Further, M can be stored in the memory and Grover search algorithm can be applied to the function below to detect collisions.

$$g : \mathbb{F}_2^n \to \mathbb{F}_2 : x \mapsto \begin{cases} 1 & \text{if there is a } \quad y \in M \quad \text{with} \quad x \neq y \quad \text{and} \quad f(x) = f(y), \\ 0 & \text{else.} \end{cases}$$

1.2.3 Cryptanalysis of classical algorithms

As we know, quantum computing possesses a serious threat to conventional security systems based on current cryptography algorithms. RSA and other conventional cryptography algorithms are resilient against classical computer based attacks. In the present era, General Number Field Sieve (GNFS) algorithm [48] is the best known algorithm for factoring large integers with computation complexity $O(1.923n^{\frac{1}{3}}ln(n^{\frac{2}{3}}))$. In Tables 1.4 and 1.5, total time in cracking RSA algorithm are showcased using conventional computing machines.

Table 1.4: Factoring times in MIPS-years for various moduli using GNFS [2]

RSA Key Size (bits)	512	1024	2048	4096
Factoring time (MIPS-years)	2×10^4	2×10^{12}	6×10^{22}	3×10^{36}

Table 1.5: Projected future factoring times using GNFS for various moduli using 1000 workstations [2]

RSA Key Size (bits)	1024	2048	4096
Factoring time in 1997	10^7 years	3×10^{17} years	2×10^{31} years
Factoring time in 2006	10^5 years	5×10^{15} years	3×10^{29} years
Factoring time in 2015	2500 years	7×10^{13} years	4×10^{27} years
Factoring time in 2024	38 years	10^{12} years	7×10^{25} years
Factoring time in 2033	7 months	2×10^{10} years	10^{24} years
Factoring time in 2042	3 days	3×10^8 years	2×10^{22} years

With the advancement of large practical quantum computer, Shor's and Grover's algorithms can solve mathematical trapdoor functions and lead classical cryptography in jeopardy. Grovers Algorithm can solve black box search problem in $O(log(n))$ making DES, AES and other symmetric key cryptosystems vulnerable to quantum computers. Similarly, Shors algorithm can find prime factors for large numbers and due to that RSA and other asymmetric key cryptosystems become vulnerable to quantum computers in the near future. The factoring time of prime numbers on the classical computer is $e^{O(n^{\frac{1}{3}}log(n^{\frac{2}{3}}))}$, whereas an equivalent quantum computer takes only $e^{O(log(n))}$ steps. Table 1.6 depicts factorization time using a 100-MHz theoretical Quantum Computer [2].

Table 1.6: Quantum factoring times on a 100-MHz theoretical Quantum Computer

RSA Key Size (bits)	512	1024	2048	4096
Quantum memory (qubits)	2564	5124	10244	20484
Number of quantum gates	3×10^9	3×10^{10}	2×10^{11}	2×10^{12}
Quantum factoring time	33 seconds	4.5 minutes	36 minutes	4.8 hours

Since it is anticipated that the quantum computer having sufficiently qubits can be developed in near future, in order to keep information and data security intact, the need of the hour is to develop cryptography algorithms, which cannot be cracked by quantum computers, i.e. post-quantum-cryptography algorithms. In the next section, a survey of post-quantum cryptography schemes is presented for completeness.

1.3 Survey of Post Quantum Cryptography Algorithms

According to Shannon's information theorem [49, 50], one-time generated key having same size as that of the message makes an absolute secure system. It is observed that the algorithms based on Shannon's information theorem are practically very inefficient. Therefore, various practical and efficient one-time signature schemes are proposed based on hash based methods. In this regard, Lamport-Diffie one time signature scheme (LD-OTS) [51] and Winternitz one-time signature algorithm (W-OTS) [52] are among the notable schemes. LD-OTS employs one-way function f and a cryptographic hash function g, where $f : \{0,1\}^n \rightarrow \{0,1\}^n$ and $g : \{0,1\}^* \rightarrow \{0,1\}^n$. Here, n is a positive integer. Since W-OTS applies one string in one signature key and generates several message digests, therefore, it generates a smaller signature compared to LD-OTS. The limitation of one-time signature scheme is the incapability to reuse same key-pair. This issue is addressed by introducing Merkle signature scheme (MSS) [53], where a binary hash tree is used as a public key for the authentication. With the generated MSS key pair, 2^H number of documents can be signed or verified. Here, H is height of the Merkle tree and $H \in \mathbb{N}, \quad H \geq 2$. Further, the security features of MSS can be enhanced by introducing Channing MSS (CMSS) [54] and distributed signature generation or Generalized Merkle Signature Scheme GMSS [55].

Code-based cryptography facilitates construction of public-key cryptosystems, which are secure against attacks generated from a quantum computer. Public key code (PKC) based encryption scheme

is first proposed by Robert J. McEliece [56], where a private key used is a random binary irreducible Goppa code [57]. Thereafter, a few code based cryptosystems, such as, Niederreiter encryption scheme [58], Courtois, Finiasz and Sendrier (CFS) signature scheme [59] and various other identification schemes [60–62], for example, Stern's scheme [63], Gabidulin codes [64], Bose-Chaudhuri-Hocquenghem (BCH) codes [65], Reed-Muller codes [66], GRS codes [67], algebraic geometric codes [68], Graph based low-density parity-check code (LDPC), Luby transform (LT) or turbo code, Goppa/alternate codes [69] etc. are developed as code based PQCs. Although code based cryptosystems are known for their lower algorithmic complexity in encryption and decryption, but their disadvantage is the longer key length and malleability of the ciphertext.

Lattice based cryptography is an integrated part of PQC algorithms and many lattice based cryptosystems are presently being used in various security applications. In Ajtai [70], a family of lattice one-way trapdoor function, SWIFFT hash function [71] based on cyclic hash function, is introduced. Security of this scheme depends upon the worst-case hardness of shortest vector problem (SVP) of n^c, where c is a constant parameter of $n \times n$ lattice. There are three basic categories of lattice PQCs, viz., Goldreich, Goldwasser and Halevi/ Hermite normal form (GGH/HNF) public key cryptosystem, NTRU scheme and Learning with errors based (LWE) cryptosystem. GGH/ HNF [72, 73] is derived from the hardness of decoding linear codes over finite fields, whereas NTRU [74, 75] is based on ring-based lattice cryptosystem. The most efficient lattice based cryptosystem is Regev's learning with errors (LWE) scheme [76, 77] and homomorphic property introduced by Gentry [78].

The piece-wise linear chaotic map (PLCM) [79–81], piece-wise nonlinear chaotic maps, logistic maps [82, 83], various polynomial chaotic maps based systems can be implemented, which are considered as PQCs. Now-a-days, physical chaotic dynamical systems have become an active area of research in the chaos based cryptography [84, 85]. For strengthening the security and to increase the speed of chaotic cryptosystems, a combination of multiple algorithms are presented in [86–89]. Symmetric block cipher encryption scheme applying multi-dimensional chaotic maps are elaborated in [90–93]. Moreover, only a few chaotic map-based multimedia encryption schemes have been developed based on hyperchaotic temperature fluctuation model [84], peanut-shaped symmetric equilibrium curve [94] and multistability of two circles of equilibrium [95] etc.

Multivariate public key cryptography (MPKC) schemes, first proposed by Shigeo and Hideki [102], are preferred due to NP-hardness in solving quadratic polynomial over a finite field, which are an

Table 1.7: Comparison between the Post Quantum algorithms

Algorithm	Type	Advantages	Limitations	Implementable Areas
Lamport- Diffie One-time Signature [51] Winternitz One-time Signature [52] Merkle's tree authentication [53] PRNG One-time Signature Distributed signature generation [55]	Hash	Fast, one time signatures are secure, reduced key size in Merkle's tree, and same key can apply in multiple encryption schemes.	Vulnerable against brute force attacks, Grover's algorithm on QC can determine key easily, collision preimage resistant is low.	Can be used as replacement of RSA and ECC based signature algorithms.
McEliece PKC [56] Niederreiter PKC [58] Courtois, Finiasz and Sendrier (CFS) [59] Stern's identification scheme [63]	Code	No known crypto analytical attacks Goppa, GRS, Reed-Mullar, Rank matrix code	Larger key-size and slower computation	Can be utilized in performance driven high security applications
The Ajtai-Dwork lattice [70] SWIFFT hash based lattice [71] The GGH/HNF public key [72, 73] The NTRU public key [74] LWE based cryptosystem [76, 77] GGH with NTRU digital signature [75]	Lattice	NP-hard problems, no known Quantum algorithms to solve in polynomial time. On the other hand, simple to design on hardware	Larger key length, many parameters are needed to generate key	Can be utilized both in asymmetric and symmetric cryptosystem, LWE lattices are most widely employed PQC
Piece-wise linear chaotic map [79–81] Piece-wise nonlinear chaotic maps Chaotic logistic maps [82, 83] Chaotic physical dynamic systems [84, 85, 94, 95]	Chaos	High throughput, parallel architecture, easy to realize on hardware platform	Longer key-size with multiple independent parameters	Can be employed in realtime applications using low power embedded SoC driven systems
Rainbow signature $(2^8, 18, 12, 12)$ [96] PMI+ signature $(136, 6, 18, 8)$ [97] Quartz/HFEv signature $(2, 129, 103, 3, 4)$ [98, 99] Unbalanced oil-vinegar scheme [100] Polly-cracker & various Isogeny [101]	Multivariate	Fully homomorphic lattice, useful in secure cloud computing	Vulnerable against a few algebraic and differential rank attacks	RFID, wireless sensor, PDA etc. low power IoT applications

extension of lattice based PQCs. Trapdoor functions in the multivariate PKC cryptosystems are converted in the form of multivariate quadratic polynomial map over a specific finite field. The Rainbow $(2^8, 18, 12, 12)$ [96], Perturbed Matsumoto-Imai plus PMI+$(136, 6, 18, 8)$ [97], QUARTZ [98] or HFEv-$(2, 129, 103, 3, 4)$ signature scheme [99] are Tame transformation signature (TTS) based cryptosystem. Another lesser known MPKC algorithm is the unbalanced oil and vinegar (UOV) algorithm [100]. Recently, a few public-key cryptosystem based on isogenies [101] are proposed in the literature. Among them, fast modular multiplier based supersingular isogeny [103], Diffie-Hellman (SIDH) [104], supersingular elliptic curve isogenies (SECI) [105] are the most notable ones. In Table 1.7 a comparison of various PQC algorithms is presented briefly. It can be concluded from this table, chaos and lattice based cryptosystems can be implemented efficiently using conventional computing machines as well as in the hardware.

1.4 Motivation and Research Contributions

In the previous section, the issues related with conventional cryposystems are discussed. The PQCs can be employed to mitigate these issue and it is found that lattice and chaos based PQCs are the most suitable candidates among all. The reason behind choosing these two PQCs are presented below. A n-dimensional lattice means a periodic structure comprised of n-linearly independent vectors $a_1, \ldots, a_n \in \mathbb{R}^n$,, which can be represented as equation 1.1, where, the vectors $a_1, \ldots, a_n$ are called basis of that n dimensional space.

$$\mathcal{L}(a_1, \ldots, a_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{a}_i : x_i \in \mathbb{Z} \right\} \tag{1.1}$$

The reason behind selecting lattice based encryption systems to be a suitable candidate among all PQC algorithms is the quantum computer's inability to solve lattice problems. It is to mention that quantum Shor's algorithm achieves a very little success in finding periods in lattice. Till date, there are no polynomial time quantum algorithms that can approximate the lattice problems. Computationally hard lattice problems are depicted below for the reference.

* *Shortest Vector problem* (SVP) [106] [107]: Finding the shortest nonzero vector $\mathcal{L}(A)$ for a given lattice basis vector A

* *Closest Vector Problem* (CVP) [108] [109]: For a fixed target vector t, which is not necessarily incorporated in the given lattice, finding lattice point $v \in \mathcal{L}(A)$ closest to the target vector t,

where A in the basis vector for a given lattice

* *Shortest Independent Vector Problem* (SIVP): Find n linearly independent lattice vectors $\mathbf{S} = [s_1, \ldots, s_n]$ (where $s_i \in \mathcal{L}(A)$ for all i), minimizing the value of $\|\mathbf{S}\| \equiv max_i\|s_i\|$

Finding the exact solutions for SVP or approximate factors in the polynomial time $poly(n)$, and the time complexity of the best-known algorithm is $2^{O(n)}$.

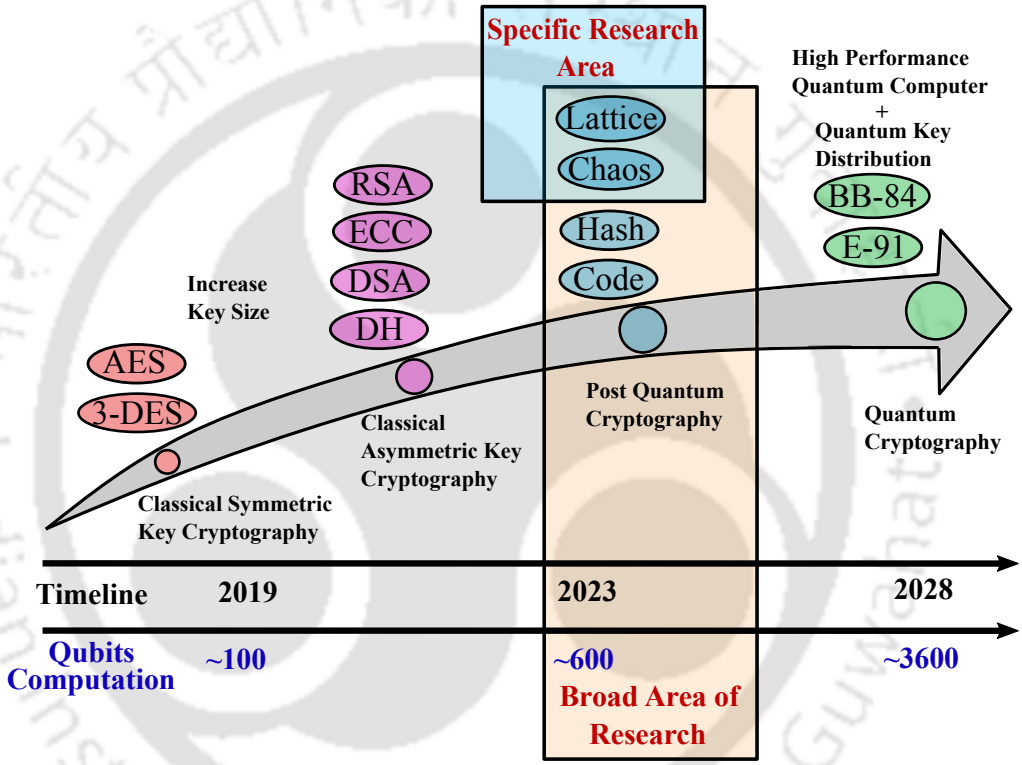


Figure 1.1: Cryptography and Quantum Computing and Research Area

Similarly, nonlinear chaotic physical dynamic systems can be utilized to formulate cryptosystems. Chaotic dynamic system based algorithms can be categorized as PQC algorithms and also be a viable solution to quantum computing generated attacks as there are no known successful attacks against them [110]. These systems are distinguished by their high sensitivity to initial conditions, statistical similarity to random signals and a continuous broadband power spectrum [111–113]. All this has garnered interest from cryptanalyst to propose various chaos based cryptographic systems [114, 115] and can be subdivided into two classes. In the first class, a chaotic system is numerically iterated over a period employing a message as an initial data [116], whereas, in the second class, a message is scrambled with a chaotic dynamic function. The relevance and usefulness of chaos have been

demonstrated through comparative studies employing characteristics of a chaotic system and the requirement of a strong cipher [115, 116].

Design and implementation of above two categories of PQC on classical machine is simple and provide resilience against quantum computer generated attacks. Therefore, in the proposed research work, we focused on the development of two PQC cryptosystem based on lattice and nonlinear chaotic physical dynamic systems, which are also highlighted in Figure 1.1. The main research goal is to develop PQC algorithm and to implement it over conventional hardware available with us for resisting both classical and quantum computer based security attacks. In order to achieve above mentioned goals, it is imperative to design various essential building blocks efficiently, which are elaborated in the next section briefly.

1.4.1 Research Contributions

- **Custom pseudo random number generator (PRNG)**
  - Two PRNGs, BluXor (a combination of BBS and Xorshift) and MPCG (a modification of the conventional PCGs) are designed on the and hardware platforms, ZedBoard Zynq^{TM}−7000 FPGA. It is observed, BluXor can be employed for general purpose cryptographic applications, whereas, MPCG can be used in low-power and area-constrained applications.
  - BluXor and MPCG generate 4.83×10^7 and 4.29×10^7 random numbers each seconds, consuming $17mW$ at $48.31Mhz$ and $16mW$ at $42.90Mhz$ with a maximum throughput of $184.288MBps$ and $163.651MBps$, respectively.
  - The randomness quality of the proposed PRNGs and existing PRNGs (AES, KISS, MT19937, LCG, Modular exponentiation, BBS and Xorshift) are evaluated using standard Diehard battery suite and NIST SP $800-22$ suite.
- **Tensor based polynomial modulo multiplier**
  - A novel design methodology of multivariate polynomial modulo multiplication based on tensors is developed, which outperforms other multipliers in terms of resource utilization and power consumption. The proposed multiplier is embarrassingly parallel and scalable, and performs linearly for single variable polynomial multiplication and quadratically for multivariate polynomial multiplication in the worst case.

- For single variable polynomial multiplication, the proposed multiplier utilizes minimum 6% less resources, consumes 6.5× less power and more than 6× faster as compared to other polynomial modulo multipliers. As per our knowledge, a multivariate polynomial modulo multiplier is realized first time on the hardware platform and its performance is analyzed with the multivariate polynomial of 9 variables, where the maximum degree of each variable is 128.
- Performance of the multivariate multiplier is linear with respect to various parameters and it is successfully validated using a LWE based lattice multivariate encryption scheme to measure its efficacy. Further the usability of the multiplier is an ideal choice for efficient homomorphic encryption algorithms implementation on various low power devices.

- **Chaos based cryposystem development**
  - A new symmetric key cryptography scheme based on a nonlinear triple pendulum chaotic map is discussed with a detailed analysis and standard tests of the proposed method classify it to be considered as a potential PQC algorithm.
  - The generated chaotic map is ratified as a cryptographically secure pseudo random number generator through various standard benchmark tests viz. sensitivity to parametric and to initial values, ergodicity, collision test, NIST randomness test etc. The proposed chaotic generator is utilized to develop a symmetric key encryption scheme and is realized on FPGA platform whose overall power consumption, delay and throughput are 0.186 W, 1.592 ns and 2396.164 MBps, respectively.
  - The highly nonlinear and interdependent unpredictable nature of our proposed scheme makes it difficult to breach the security without complete knowledge of its key. It is also unbreakable practically with the partial knowledge of its key because of high sensitivity of chaotic maps to key values. Encryption algorithm is tested against various attacks to prove it's cryptographic strength.
  - The chaos dynamic system based algorithm derived from nonlinear triple pendulum can encrypt both text and multimedia as input data. The cryptosystem is further implemented using 180nm Bulk CMOS technology occupying 0.20374 mm^2 die area and consuming 61.88 mW power at 250MHz at 180nm and it can operate from low (at least 100KHz) to high

operating frequencies efficiently without showing any logic error or performance deterioration.

- **Lattice based partially homomorphic encryption (PHE) scheme**
  - A software hardware co-design of two variants of learning with error (LWE) based lattice PHE schemes employing an ARM-SoC and an FPGA, which provides necessary acceleration to PHE schemes, are designed. The first PHE variant is developed for generic homomorphic encryption, while second variant is aimed for resource optimal lightweight IoT driven applications. It is to mention that LWE PHE scheme is analyzed for its algorithmic correctness. The first variant takes {6.88μs, 16.18μs, 2.84μs} and {1.65μs, 4.38μs, 0.72μs} for encryption, decryption and recryption using Zedboard and Virtex-7 FPGA platforms, respectively. Similarly, the second variant consumes {1.44μs, 0.96μs, 0.22μs} and {0.35μs, 0.225μs, 0.055μs}, respectively for encryption, decryption and recryption on the above mentioned FPGA platforms, respectively.
  - A robust and reliable low latency data transfer protocol is developed between FPGA based accelerator IP and ARM-SoC host system for their seamless assimilation. Moreover, a Linux based embedded operating system is customized with board support package (BSP), device-tree and first stage boot loader (FSBL) to enable all the essential functionalities required to communicate cross platform devices for performing encryption and decryption.
  - The best implementation of the proposed PHE scheme is found to be 52.71× more resource efficient than another contemporary method based on BGV RLWE. The cumulative delay and throughput of developed scheme is 0.63μs and 6523.752 Mbps exhibiting 43% and 29% speed improvement and 23× lower delay over non-pipelined and pipelined BGV RLWE schemes on Virtex-7 FPGA board.

1.5 Organization of the Thesis

The proposed research work is organized into six chapters in this thesis work. The summary of each chapter is given below.

- **The second chapter** presents PRNGs, which is the most important building block in the security algorithms. Two custom PRNGs, BLUXOR and MPCG, are proposed and their detailed

description, comparison with other well-known standard PRNGs are also elaborated in this chapter.

- **The third chapter** discusses a tensor matrix based multivariate polynomial modulo multiplier. This optimally designed multiplier is first mathematically analyzed and is designed both on software (MATLAB) as well as on hardware (FPGA) platforms. Later, it is compared with contemporary multiplier implementations on various parameters to establish its efficacy. A lattice based PQC is implemented using the proposed multiplier to showcase its effectiveness.

- **The forth chapter** elaborates a novel encryption scheme based on triple pendulum chaotic physical dynamic system in details. The mathematical construction, software hardware (both FPGA and ASIC) implementation, bifurcation proof of chaotic attractors and examples of image and text encryption-decryption are depicted in this chapter. The symmetric encryption algorithm proposed in this chapter presents its security against various cryptanalytic attacks.

- **The fifth chapter** explores the formulation and implementation of another PQC algorithm based on LWE lattice architecture with partially homomorphic property. Further, two variants of this scheme is developed, one for general purpose application and another for low power IoT applications. Finally, the whole algorithm is realized as a hardware accelerator and is integrated on an ARM enabled Zynq SoC with high-speed data transfer protocol and design semi-custom software stack.

- **The sixth chapter** summarizes the work presented in this thesis highlighting the main contributions of the work and the directions for future research.

2

Pseudo Random Number Generators

Contents

Objective

Pseudo-Random Number Generators (*PRNGs*) are an integral part of cryptographic applications, such as key generations, digital signatures, Internet-of-Things (*IoT*) security, etc. These applications require low-power and high-throughput PRNGs along with statistically secure random numbers capability. In this chapter we propose two PRNGs, BluXor and MPCG, inspired from Blum-Blum-Shub (*BBS*), Xorshift and Permuted Congruential PRNGs. BluXor and MPCG are elaborated in this chapter in detail.

2.1 Introduction to PRNGs

Pseudo-Random Number Generators (*PRNGs*) [117,118] are deterministic and reproducible algorithms, which are used to generate sequences of random numbers for various cryptographic applications. Data and network security domains require PRNGs having various statistical properties, such as low autocorrelation values, secure, irreproachable and high linear span with non-repeatability. The key design challenge is to develop secure PRNGs, which can generate random numbers with high throughput as well as can stand against malicious attacks. In other words, an attacker should not differentiate between number generated from PRNGs and truly random number sequences.

In the recent years, a few random number generators, especially hardware True Random Number Generators (*TRNGs*) [119–122] have been proposed for cryptographic applications. As compared to PRNGs, the quality of randomness is better in TRNGs, but due to the lower speed and non-reproducible with unknown period, it is difficult to use them in most of the cryptographic applications, although TRNGs are useful in few low-speed cryptographic applications. On the other hand, PRNGs are the most commonly used in encryption schemes and security-related systems due to higher speed, less complexity and fewer hardware requirements.

At hardware level of abstraction, there are different approaches used to design PRNGs, such as Linear Feedback Shift Registers (*LFSRs*) [123], Linear Congruential Generators (*LCGs*) [124, 125], Multiple Recursive Generators (*MRGs*) [126], modular explanation based generators [127], nonlinear chaos based generators [128] etc. LFSRs are represented by a single variable polynomial equations, which provide a simple means for generating random lists of numbers very fast. Generating the pseudo-random numbers using LFSRs requires right/left-shift and XOR operations only. LCGs are the oldest and widely used PRNGs, which generate a sequence of pseudo-random numbers using several dis-

continuous weighted linear equations that can be implemented easily. Modular exponentiation based PRNGs perform exponentiation over a modulus and it is very useful in public-key cryptographic applications. The operation of modular exponentiation produces larger periods by calculating remainder when an integer base is raised to some power of exponent, which is divided by a positive integer or the modulus. Some of the standard PRNGs are also designed based on the concepts of chaotic dynamic system, whose main benefits are unpredictable, disordered and sensible to its particular seed. Prior to use PRNGs in any application, they should be checked for statistically perfection, which can be determined using Diehard battery tests [129] and NIST tests [130]. These tests are globally considered and accepted as the standard statistical evaluation tests for any random number generators.

In this chapter, two PRNGs derived from BBS, Xorshift and PCG algorithms, which are based on modular exponent, and LFSR and LCGs approaches, respectively. The key feature of the proposed PRNGs is that they are highly secure consuming low-power and delivering high-throughput. In Section 2.2, BBS, Xorshift and PCG algorithms are discussed. Subsequently, PRNGs, BluXor (combining BBS and Xorshift) and Modified PCG (*MPCG*), which is derived from the conventional PCGs, are presented. The hardware implementation on ZedBoard (Zynq^{TM}−7000 FPGA) is elaborated in section 2.3. In section 2.4, the proposed PRNGs are compared with existing well-known PRNGs based on Diehard battery and NIST SP800 − 22 tests to showcase their efficacy.

2.2 Proposed Pseudo Random Number Generators

In this section, proposed pseudo random number generators (PRNGs), BluXor and MPCG, are discussed in detail.

2.2.1 BluXor PRNG

BBS is a non-permutation algorithm proposed by Lenore Blum, Manuel Blum and Michael Shub in 1986 [131]. BBS generates pseudo-random number of size $log_2(log_2 M)$ using quadratic residue congruence, where $M = pq$ is a Blum integer. The special primes p and q are derived from other prime integers which decide period of the sequence. The security of BBS generator depends on Blum integer $M \geq 2^{64}$ which is hard to factorize. The pseudo-random numbers generated using BBS shown in equation 2.1.

$$X_{n+1} = X_n^2 mod M. \tag{2.1}$$

Here, M is the product of two large distinct prime numbers called Blum primes. Blum primes are prime numbers p, such that, $p \equiv 3(mod4)$. BBS algorithm for generating a pseudo random sequence of length 32-bit is given in algorithm 2.

Algorithm 2: Blum Blum Shub Algorithm.

Input: b: 32-bit word;
Output: z: 32-bit word;
Parameters: $M = pq$: product of 2 large primes p and q;
begin
    $b \leftarrow b^2 mod M$
    $z \leftarrow b$

Xorshift is a fast uniform random number generator proposed by George Marsaglia in 2003 [132]. The next state of a Xorshift is obtained by applying a certain number of Xorshift operations to n-bit blocks in the current state. Xorshift operation involves bitwise XOR of n-bit block with a shifted version of itself by a position. Xorshift algorithm for generating a pseudo-random sequence of length 64-bit is presented in algorithm 3.

Algorithm 3: Xorshift Algorithm.

Input: y: 64-bit word;
Output: t: 64-bit word;
Parameters: a, b, c: all are integers;
begin
    $y \leftarrow y \oplus (y \ll a)$
    $y \leftarrow y \oplus (y \gg b)$
    $y \leftarrow y \oplus (y \ll c)$
    $t \leftarrow y$

BBS is based on solving quadratic residue problems, therefore, it has greater strength to resist malicious prediction based attacks. However, the throughput of BBS is very low. Additionally, statistical quality of BBS is lower due to its smaller period. Thus, BBS fails to pass many Diehard battery and NIST tests. On the other hand, Xorshift can generate random numbers at a very high rate, but due to its less complex architecture, it is predictable under various adversary attacks. In order to maintain statistical quality along with high throughput at the same time, a combination of BBS and Xorshift is proposed in this thesis. This PRNG derived from BBS and Xorshift is named as BluXor, which can be applicable for general purpose applications. As shown in figure 2.1, one 32-bit BBS module and three 64-bit Xorshift modules are utilized to design BluXor. Here, output

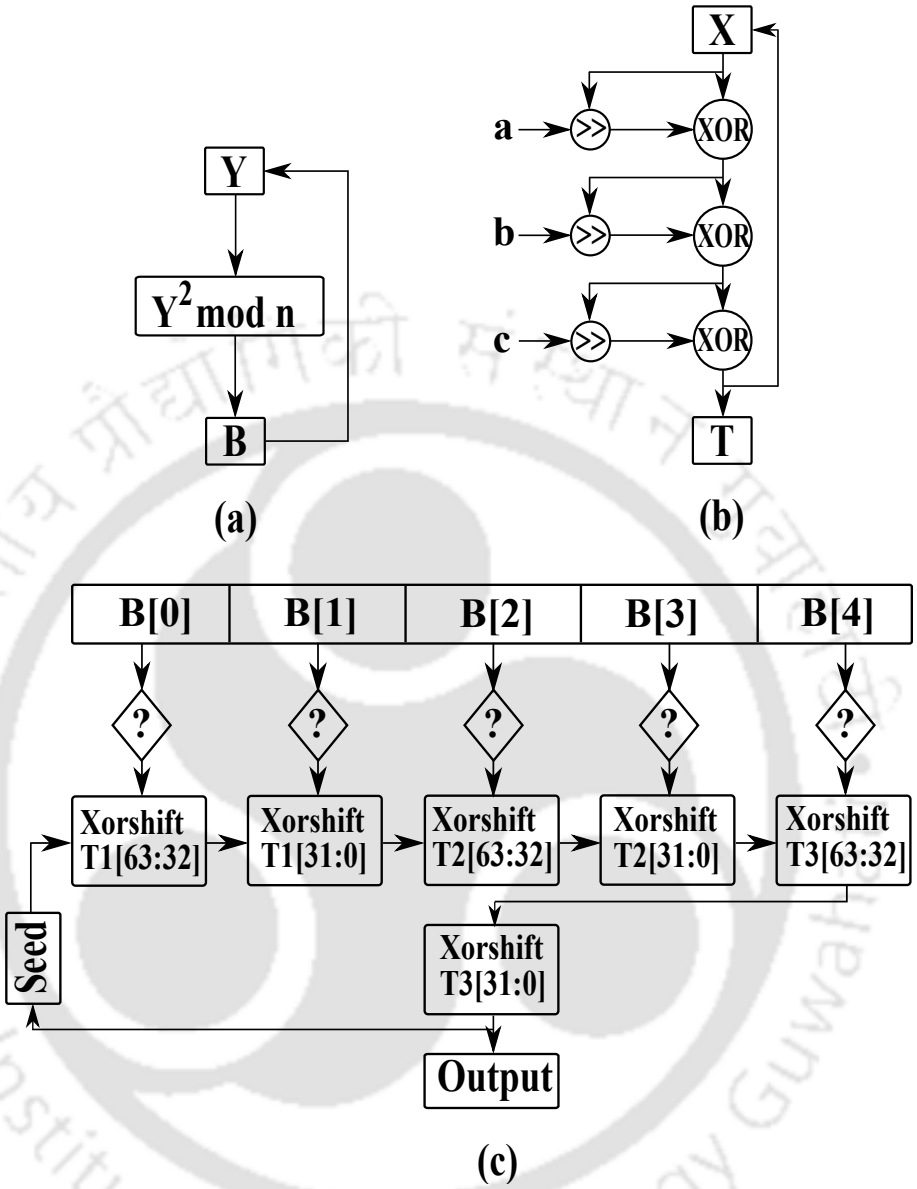


Figure 2.1: Block diagram of: (a) BBS; (b) Xorshift; and (c) BluXor PRNGs.

of the BBS module is applied to generate initial seed values, which are fed to Xorshift modules to produce pseudo random number sequence. Based on the preconditions, Xorshift modules right shift the outputs. It can be seen from figure 2.1 that BluXor takes few clock cycles for the initialization and then at every clock cycle, it produces a 32-bit random number. The sequence predictability problem is resolved by feeding new seeds in Xorshift module from BBS. n-LSBs of BBS act as switches for Xorshift outputs and, at every iteration, n-LSBs of BBS are checked. It is to be noted that n is arbitrarily chosen as 5 in the proposed design. If these n-bits are set, then the higher 32-bit of Xorshift output becomes the new state, otherwise, the lower 32-bit of Xorshift output becomes the new state. In this

way, the limitations of BBS and Xorshift are resolved in BluXor. BluXor algorithm for generating a pseudo-random sequence of length 32-bit is stated in algorithm 4.

Algorithm 4: BluXor Algorithm

Input: x: 32-bit word;
Output: r: 32-bit word;
begin
  $t_1 \leftarrow Xorshift$
  $t_2 \leftarrow Xorshift$
  $t_3 \leftarrow Xorshift$
  $z \leftarrow BBS$
  if $(z\&1! = 0)$ **then**
    $x \leftarrow x \oplus (t_1 \gg 32)$
  if $(z\&2! = 0)$ **then**
    $x \leftarrow x \oplus (t_1\&0 \times ffffffff)$
  if $(z\&4! = 0)$ **then**
    $x \leftarrow x \oplus (t_2 \gg 32)$
  if $(z\&8! = 0)$ **then**
    $x \leftarrow x \oplus (t_2\&0 \times ffffffff)$
  if $(z\&16! = 0)$ **then**
    $x \leftarrow x \oplus (t_1 \gg 32)$
  $x \leftarrow x \oplus (t_3\&0 \times ffffffff)$
  $r \leftarrow x$

2.2.2 Modified PCG PRNG

PCG is a standard PRNG algorithm proposed by Raj S. Katti Rajesh, G. Kavasseri and Vyasa Sai in 2010 [133]. PCG involves transformation of random numbers generated by LCG. The transformations, such as, bitwise rotation, shifting and XOR are applied to the outputs of LCG. The reasons that drive to the selection of PCG are as follows.

(i) PCG is easy to use, flexible and can be designed with less resource as compared to other PRNGs.

(ii) Due to smaller block size, PCG has high throughput.

(iii) The statistical quality of PCG is better as compared to other PRNGs.

There are a number of variants in PCG family and among them, the most significant variants are discussed below.

PCG-XSH-RR (64-bit input, 32-bit output): The strategy is to perform Xorshift to improve randomness of high order bits. The output is then randomly rotated so that all the bits occupy full

period. Thus, it mnemonics as Permuted Congruential Generator Xorshift Right Rotate (*PCG-XSH-RR*). A PCG-XSH-RR function with 64-bit input and 32-bit output can be denoted as mentioned in equation 2.2.

$$out = rot32((in \oplus (in >> 18)) >> 27, in >> 59). \tag{2.2}$$

PCG-XSH-RS (64-bit input, 32-bit output): This variant trade-offs statistical quality for throughput by replacing the rotation operation with shift operation in PCG-XSH-RR. Thus, it is known as Permuted Congruential Generator Xorshift Right Shift (*PCG-XSH-RS*). A PCG-XSH-RS function with 64-bit input and 32-bit output is expressed as equation 2.3 below.

$$out = (in \oplus (in >> 22)) >> (22 + (in >> 61)). \tag{2.3}$$

Here, attackers have access to sufficient number of random sequences to predict the governing permutation of PCGs using supercomputer based brute-force attack. Therefore, there is a scope of improvement in the security of PCGs using intelligent approaches for permutation. It is to mention that Modified PCG (*MPCG*) is proposed with modifications in the conventional PCGs. MPCG generates pseudo-random sequences of 32-bits, which is given in algorithm 5. In MPCG, the permutation technique utilized is based on the tuples that improves the security of PCG. As shown in algorithm 5, MPCG introduced several bitwise XOR and shift transformations and parameters (a, b and c). The algorithm starts with a seed value, k_0 and then the first random number, k_1, is generated employing k_0. Thereafter, the second random number, k_2, is generated using k_1, and this process continues until $i = n - 1$, where $i = n - 1$ and $k_{i+1} = k_n$. In the end of MPCG algorithm, a stream of $(n - 1)$ 32-bit random numbers are generated. It is to be noted that the selected seed value should generate a statistically secure random number. Further, the parameters should be set according to the standard PCG algorithm.

2.3 Hardware Implementation

The proposed PRNGs, BluXor and MPCG, are realized on ZedBoard Zynq$^{TM}-7000$ FPGA platform [134] using Xilinx Vivado EDA tool. For hardware implementation of algorithms 4 and 5, Verilog Hardware Description Language (*HDL*) is employed. The following subsections describe the implementation process step by step.

Algorithm 5: MPCG Algorithm

Input: k_0: 64-bit word;
Output: k_{i+1}: 32-bit stream word;
Parameters: a, b, c: all are integers;
begin
    $k_0 \leftarrow$ 64-bit-seed;
    for *(i = 0 to n)* **do**
        $k_{i+1} = ((a \times k_i$ - $b)$ *mod* $c)$;
        $k_{i+1} = k_{i+1} \oplus (k_{i+1} >> 12)$;
        output $\leftarrow k_{i+1}$;

2.3.1 BluXor Implementation

Figure 2.1 exhibits a generalized block diagram of the proposed BluXor, where combination of BBS and Xorshift to form BluXor is showcased. The three basic modules utilized during hardware implementation are described as follows. First, BBS module is implemented as shown in figure 2.1(a), and uses 32-bit long seed value for the initialization. Here, the values of modulus M and seed are hard-coded. BBS algorithm requires two 32-bit inputs, B_0 and M, which are taken as, 2132 and 3149, respectively. A 64-bit register is used to store the square of the present state. After squaring, modulo operation is performed and finally, the five LSBs from output are returned.

As shown in figure 2.1(b), it can be observed that Xorshift module requires one 64-bit input seed and three parameters. The implementation process is similar as the above mentioned BBS module and the parameters employed in the process are $a = 13$, $b = 17$, $c = 15$. In BluXor as illustrated in figure 2.1(c), three modules of Xorshift, T_1, T_2 and T_3 are employed with different seed values. These seed values are 1145, 3256 and 78945 for T_1, T_2 and T_3, respectively. At each clock cycle, all the three Xorshift provide corresponding outputs.

One BBS and three Xorshift modules are required to implement BluXor, which are depicted in figure 2.1(c). The five LSB bits of BBS act as the switches for Xorshift outputs. The seed value is provided as an input parameter and in every iteration, five LSBs bits of BBS output are checked. Xorshift output is separated into two 32-bit modules. If an odd positioned bit of BBS is set, then higher order 32-bit of Xorshift modules are XORed with the current state. For an even positioned bit, lower order 32-bit of Xorshift modules are XORed with the current state. For each computation of BluXor, one clock cycle is required to process BBS and Xorshift outputs.

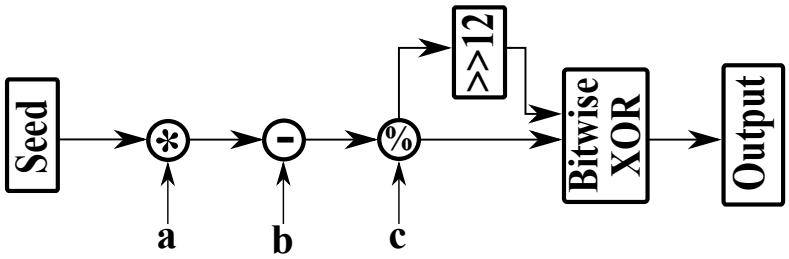


Figure 2.2: Block diagram of the proposed MPCG PRNG.

2.3.2 MPCG Implementation

Figure 2.2 exhibits block diagram of the proposed MPCG. Initially, a seed value is stored in a 64-bit register. The seed value is fed to multiplier block and the output of multiplier is input to the subtracter block. The second operands of these blocks are parameters a and b. The output of subtracter block is $mod(c)$ and the remainder is restored in a 32-bit register. The remainder is further shifted or rotated right by 12 bits. Subsequently, the shifted/rotated value and previous restored value are passed as inputs to a bitwise XOR block. Finally, a 32-bit random number is obtained as an output from the bitwise XOR block.

2.4 Results and Discussion

As mentioned earlier, BluXor and MPCG are realized on ZedBoard Zynq$^{TM}-7000$ FPGA platform [134] and generate a dataset of random numbers. The binary outputs, generated by FPGA, are converted to single precision floating point values according to $IEEE-754$ format. Subsequently, the floating point values are normalized $w.r.t.$ the maximum value and are plotted using MATLAB. For illustration, the first 25000 random numbers are taken for scattered plot, which is shown for BluXor and MPCG in figure 2.3 and 2.4, respectively. It can be seen from figures 2.3 and 2.4 that quality of the randomness and distribution of the number generated using BluXor and MPCG are quantified below showcasing its efficacy.

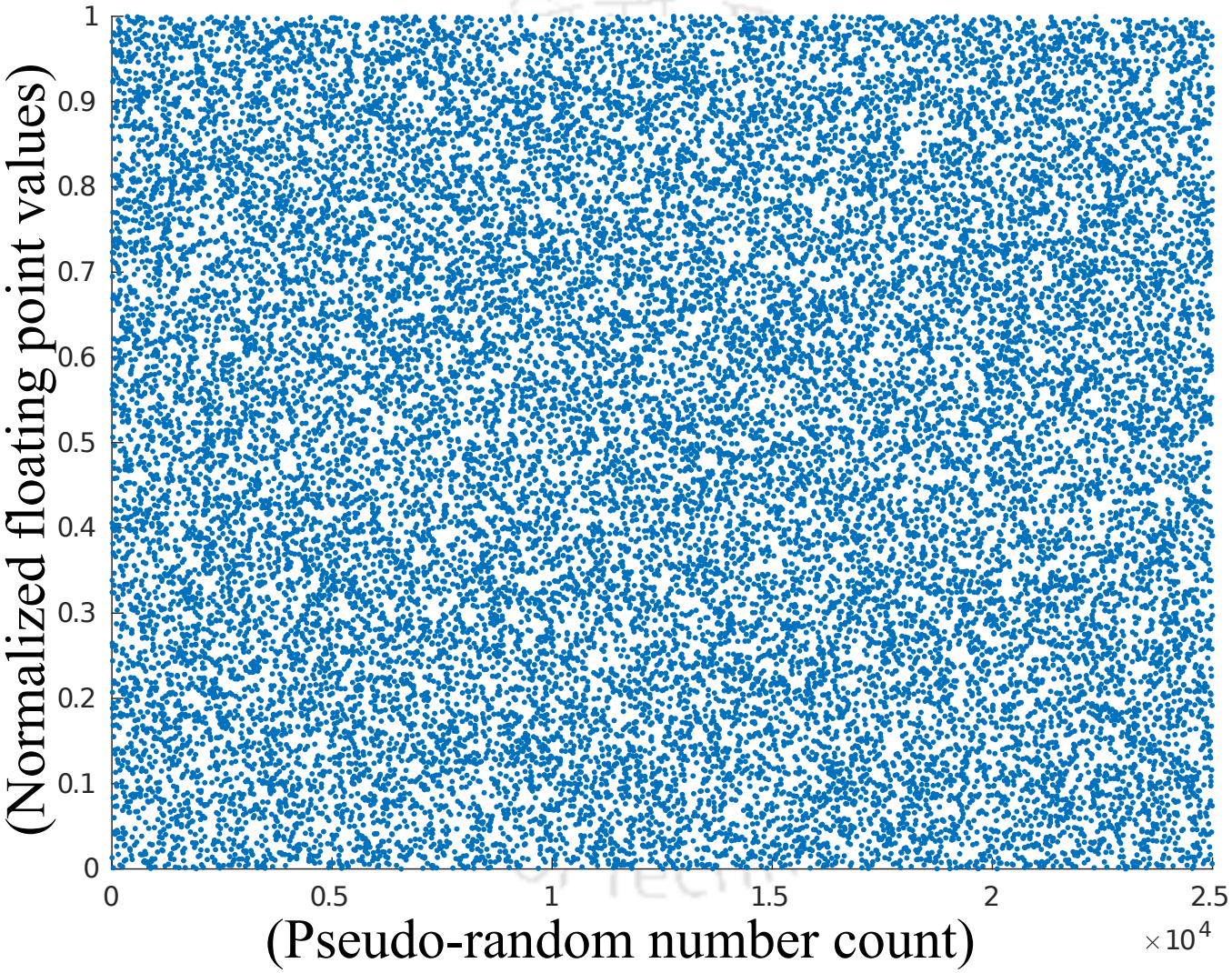


Figure 2.3: Scatter plot of first 25000 normalized random number generated from BluXor

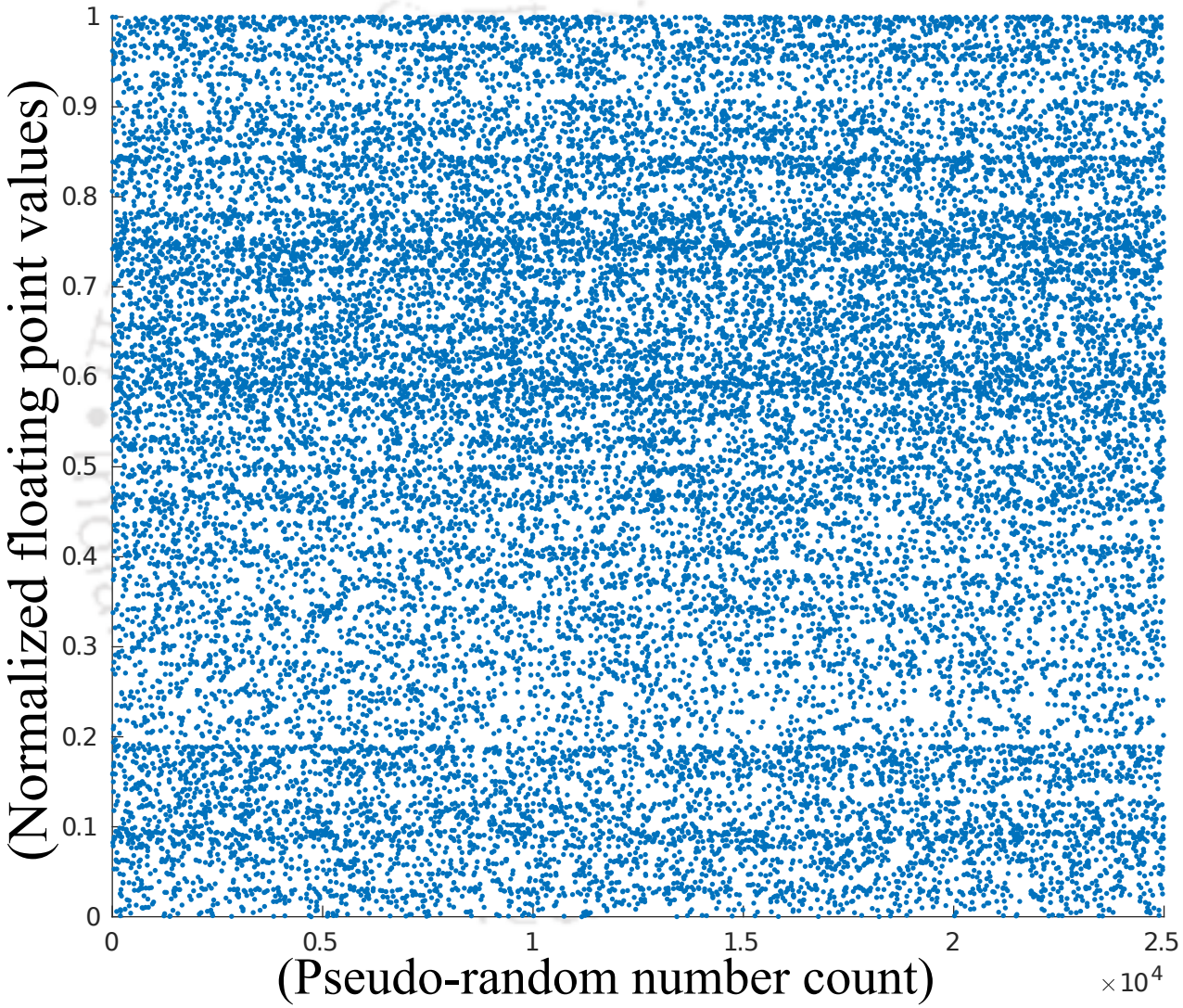


Figure 2.4: Scatter plot of first 25000 normalized random number generated from MPCG

Table 2.1: Hardware Implementation Results For The Proposed PRNGs

Parameters			Proposed PRNGs	
			BluXor	**MPCG**
Resources	**LUTs**		1267 (2.38%)	1371 (2.58%)
	Registers		730 (0.69%)	430 (0.40%)
	DSP Blocks		3 (2.38%)	none
Power	**Dynamic**	**Logic**	6 mW	7 mW
		Other	11 mW	9 mW
	Static		120 mW	120 mW
Delay	**Logic Path**		13.409 nS	12.974 nS
	Network Delay		7.291 nS	10.337 nS
	Critical Path		10.407 nS	11.7125 nS
Speed in Mhz			48.31 MhZ	42.90 MhZ
Throughput	**Generation**		$4.83 \times 10^7/sec$	$4.29 \times 10^7/sec$
	Memory		184.288 MBps	163.651 MBps

Table 2.1 elaborates hardware implementation results for the proposed BluXor and MPCG on ZedBoard Zynq$^{TM}-7000$ FPGA platform [134]. It can be observed from Table 2.1 that the proposed PRNGs consume low power and require less resources. Among both the proposed PRNGs, BluXor is faster, but it dynamic power consumption is more (17mW) and also requires more number of registers (730) and DSP blocks (3) as compared to MPCG. On the other hand, MPCG consumes less dynamic power (16mW) and requires less number of registers (430), which is good for light-weight low power security applications. It is to mention that the throughput of BluXor and MPCG are $184.288 MBps$ at $48.31 Mhz$ and $163.651 MBps$ at $42.9 Mhz$, respectively.

In order to evaluate and compare the randomness quality, the proposed PRNGs are tested against existing PRNGs using the standard Diehard battery suite [129] and NIST SP $800-22$ suite [130]. The test results for Diehard battery suite and NIST SP $800-22$ suite are tabulated in Table 2.2 and Table 4.7, respectively. A particular PRNG is considered to pass a test, if $0.01 \leq p\text{-}value < 1$. For Diehard battery tests, three well-known PRNGs used in the industry are selected, which are Advance Encryption System (AES) [135], Keep it Simple Stupid ($KISS$) [136] and Mersenne Twister (MT) 19937 [137]. It can be seen from Table 2.2 that BluXor passes all the 29 tests and MPCG passes 28 tests. Although MT) 19937 and $KISS$ also clear all the test, but due to algorithmic complexity, they tend to require more resources as compared to BluXor and MPCG. Similarly, NIST SP $800-22$ tests are performed, where LCG [125], Modular exponentiation [127], BBS [138] and Xorshift [138]

Table 2.2: Comparison Of The Proposed PRNGs With Existing PRNGs Based On Diehard Battery Test

Diehard Battery Tests	p-value (acceptable range $0.01 \leq p\text{-}value < 1$)				
	Existing well known PRNGs			Proposed PRNGs	
	AES(OFB) [135]	***KISS [136]***	***MT19937 [137]***	***BluXor***	***MPCG***
Birthdays Test	0.09140706	0.34997935	0.83078550	0.89915249	0.64815728
Operm5 Test	0.31162450	0.42266458	0.64989346	0.37516036	0.83809070
Rank 32×32 Test	0.71907703	0.77470229	0.75293705	0.96345486	0.90655901
Rank 6×8 Test	0.07286808	0.46627877	0.52855744	0.44332652	0.04603721
Bitstream Test	0.14958476	0.87738122	0.61691013	0.98372427	0.64736979
Diehard Opso Test	0.90829551	0.40400810	0.19435334	0.95581938	0.92416133
Diehard Oqso Test	0.70146832	0.49703903	0.67856793	0.26745376	0.31318059
DNA Test	0.79221673	0.23179192	0.58119902	0.38033601	0.16068910
Count 1s-str Test	0.17279175	0.22699974	0.71410437	0.96724261	0.78825851
Count 1s-byt Test	0.77482094	0.08606009	0.77957607	0.22539871	0.70303283
Parking Lot Test	0.24200626	0.79600924	0.96263190	0.69671967	0.06102235
2-d Sphere Test	**0.99532507***	0.58735542	0.31825140	0.21652778	0.36844792
3-d Sphere Test	0.26867516	0.69626383	0.55298681	0.93923026	0.99020443
Squeeze Test	0.79345185	0.57941230	0.87337656	0.12831634	0.82152060
Diehard Sums Test	0.03846382	0.40938086	0.01877345	0.44893896	**0.00339542***
Diehard Runs Test	0.45078296	0.41562095	0.13221154	0.35697680	0.59319437
Diehard Craps Test	0.95367637	0.33323240	0.19381385	0.12462753	0.57860814
Marsaglia Tsang GCD Test	0.61111042	0.19080449	0.04367936	0.89999546	0.36775462
STS Monobit Test	0.08752067	0.94209906	0.10336182	0.12999497	0.85296739
STS Runs Test	0.45966078	0.96986806	0.60370371	0.97131711	0.06398866
RGB Bit-Dist Test	0.99082925	0.57712439	0.71271019	0.43153706	0.33869794
RGB Min. Distance Test	0.92904598	0.76232452	0.90108570	0.13589439	0.24214232
RGB Permutations Test	0.24814401	0.83039362	0.68668492	0.01729007	0.24296868
RGB Lagged Sum Test	0.48695592	0.86641877	0.03396592	0.09780326	0.45691115
RGB Kstest Test	0.13199179	0.20774150	0.94516424	0.53462563	0.68976679
DAB Bytedistrib Test	0.33621324	0.93427572	0.30629994	0.05708488	0.11194593
DAB DCT Test	0.41317487	0.71353450	0.76538553	0.07248676	0.64705683
DAB Filltree Test	0.38917962	0.64056024	0.47541113	0.45674104	0.45756793
DAB Monobit-2 Test	0.37663475	0.27961861	0.71991601	0.24737607	0.74744103

Here, * signifies that a particular PRNG fails to pass a particular test

Table 2.3: Comparison Of The Proposed PRNGs With Existing PRNGs Based On NIST $SP800-22$ Test

NIST $SP800-22$ **Tests**	**p-value (acceptable range** $0.01 \leq p\text{-}value < 1$**)**					
	Existing well known PRNGs				**Proposed PRNGs**	
	LCG [125]	**Modular Exponentiation [127]**	**BBS [138]**	**Xorshift [138]**	**BluXor**	**MPCG**
Frequency Test	0.590351	0.079955	0.324350	0.145326	0.506496	0.106628
Block Frequency Test	0.492735	0.538842	**0.000000***	0.028817	0.484548	0.04156
Cumulative Sums Test	0.525255	0.100490	**0.000000***	0.074404	0.552932	0.067332
Runs Test	0.572252	0.521124	**0.000000***	0.739918	0.489091	0.287543
Longest Run of 1's Test	0.284067	0.517221	**0.000000***	0.554420	0.440516	0.057843
Rank Test	0.539487	0.594646	**0.000000***	0.236810	0.443327	0.15037
Discrete Fourier Transform Test	0.344484	0.541247	**0.000000***	0.514124	0.464124	0.41735
Non-periodic Template Matchings	0.493525	0.488642	**0.000000***	0.512363	0.796262	0.01231
Overlapping Template Matchings	0.494436	0.466825	**0.000000***	0.595549	0.553845	0.31532
Universal Statistical Test	0.532033	0.469121	**0.000000***	0.122325	0.454763	0.24316
Random Excursions Test	0.052382	0.087102	**0.000000***	0.507812	0.281610	0.41754
Random Excursions Variant Test	0.036553	0.126518	**0.000000***	0.289594	0.398311	0.38923
Serial Test	0.489956	0.574242	0.043355	0.495847	0.498307	0.026517
Linear Complexity Test	0.274623	0.483021	**0.000000***	0.249284	0.543714	0.21839

Here, * signifies that a particular PRNG fails to pass a particular test

PRNGs are chosen for the comparison. Table 4.7 exhibits that BluXor and MPCG passes all the 15 NIST tests. The proposed BluXor performs better than BBS and Xorshift compared in the terms of p values. Although MPCG fails marginally in one test of Diehard battery suite only, it consumes low power and requires fewer resources (Table 2.1). Therefore, MPCG is recommended for the light-weighted security applications, in which low power and area are the key requirements. On the other hand, BluXor is recommended for general purpose cryptographic applications, in which security is the foremost requirement.

2.5 Conclusion

In this chapter, design and realization of two PRNGs, BluXor and MPCG, are elaborated. BluXor is a combination of BBS and Xorshift, and MPCG is a modification of the conventional PCGs. BluXor and MPCG generate 4.83×10^7 and 4.29×10^7 random numbers each seconds, respectively. Further, the proposed PRNGs consume $17mW$ at $48.31Mhz$ and $16mW$ at $42.90Mhz$ with a maximum throughput of $184.288MBps$ and $163.651MBps$, respectively. The efficacy of randomness of the proposed PRNGs are ratified using standard Diehard battery suite and NIST SP $800-22$ suite. These PRNGs are utilized in the realization of various PQC algorithms proposed in this thesis.

3

Tensor Based Polynomial Modulo Multiplier

Contents

Objective

Modulo polynomial multiplication is an important mathematical operation in the area of finite field arithmetic. Polynomial functions can be represented as tensors, which can be utilized as basic building blocks for various lattice based post quantum cryptography schemes. A tensor based novel modulo multiplication method for multivariate polynomials over $GF(2^m)$ proposed and is realized on the hardware platform (FPGA). This proposed multiplier is an essential building block while designing further lattice LWE cryptosystem.

3.1 Introduction to Polynomial Modulo Multipliers

With the rapid advancement of computational capabilities, modern cloud computations, network data transactions etc. are facing tremendous security threats against various unwanted adversaries. Post quantum cryptography (PQC) schemes exhibit stronger resistance to classical as well as quantum computing based attacks and also have become of prime interest to cryptograpy community. Among various classes of PQC (hash-based, lattice-based, code-based, supersingular isogeny, chaotic dynamic system based etc.) lattice-based encryption schemes have got the most attention, while designing alternate cryptosystems due to its easy implementation ability on the hardware. Many different security related applications, such as, homomorphic encryption, key generation, encapsulation and hash or signature generation protocols etc., for post quantum cryptosystems can be derived from the polynomial lattice. In general, the lattice-based cryptosystems are implemented using polynomial rings and, execute modulo multiplications and additions of two large-degree polynomials over $GF(2^m)$ (Galois Field) basis, which is a compute intense process.

As we know, many efficient polynomial modulo multipliers are proposed in the literature, which play a critical role in the lattice based encryption schemes. Among them, Bit-parallel canonical methods [139–144], systolic multipliers based on Montgomery techniques [145, 146], digit-serial and systolic Karatsuba methods [147–149], digit-serial dual basis multipliers [150, 151], number theoretic transform (NTT) based multipliers [152, 153], matrix-vector based Hankel and Toeplitz multiples [154–157] etc. are quite well-known. The salient features and limitations of the above mentioned multipliers are listed in Table 3.1 for the ready reference. Majority of these works primarily focus on the implementation of single variable polynomials and, deal with trinomials and pentanomials only, whereas, the method proposed by us is generic in nature and is optimally implemented on the

FPGA. Various FPGA hardware implementations of polynomial modulo multipliers can be found in [147–149, 153, 155–164]. Further, the efficiency of our proposed multiplier is estimated using various parameters, such as computational complexity, resource utilization and scalability with polynomial degree. It is found that the proposed method has less computational complexity as compared to [144, 165] and consumes less hardware resources with respect to [149, 153, 156–158]. It also exhibits higher scalability with polynomial degree than the methods presented in [147, 148, 155, 159].

We developed a generic scalable power efficient multivariate polynomial multiplier, which can provide less delay and optimal area and can be used in various low power time critical applications. Our proposed multiplier works efficiently with Learning-with-Errors (LWE) and Ring-LWE (RWE) based homomorphic encryption schemes. This method is embarrassingly parallel and improves throughput of the proposed method substantially. Along with sequential design we present 4-parallel column multiplication (PCM) and 16-PCM implementation showcased in Table 3.10. Further, the computational complexity of the proposed modulo multiplier is linear for single variable polynomial multiplications. The tensor matrices and input vectors employed in this method are sparse, which optimizes space requirement of the proposed method. Due to its low power consumption, IoT based embedded systems, surveillance drones based smart fencing systems etc. can utilize proposed multiplier for the efficient implementation of lattice based PQC homomorphic encryption algorithms.

3.2 Motivation and Related Works

Almost all the existing implementations are based on a single variable polynomial and are limited to a few number of polynomial coefficients, such as trinomials and pentanomials. In security applications, larger arithmetic fields constructed on multiple variables assure less susceptibility against cryptographic attacks. Gröbner basis is constructed with special algorithmic properties to provide easy solutions for many fundamental problems in a polynomial ring over a field. Lets assume F be a set of polynomials in the ring R, i.e. $f_1(x_1, x_2, \cdots, x_n), f_2(x_1, x_2, \cdots, x_n), \ldots, f_n(x_1, x_2, \cdots, x_n)$. The Gröbner basis G is constructed using above mentioned conditions by ordering [166] monomials in a set of polynomials. The proposed method is based on lexicographic ordering [167–169] of monomials.

Polynomials over the field GF(2^m) have coefficients from domain $\{0, 1\}$ and the highest degree of the polynomial is $(m-1)$. For $m = 16$, a GF (2^{16}) multiplier for a single variable polynomial is presented in [170]. Figure 3.1 exhibits structural organization of a single variable polynomial operation.

Table 3.1: Comparison of Existing Polynomial Modulo Multiplication Methodologies

Existing multiplier Methodologies	**Salient Features**	**Limitations**	**Reference**
Bit-parallel Canonical Multiplier	Multiplexer based approach in the construction of sub-quadratic complex space multipliers	Only irreducible trinomial is employed in this scheme.	[139–143]
Systolic Montgomery Multiplier	Efficient implementation of Montgomery multiplication over GF(2^m) for all-one polynomials	Produces estimated value of quotients, which is useful for RSA and DiffieHellman key exchange, but it is not applicable for lattice based PQC algorithms	[145, 146]
Dual-basis digit serial multiplier	This method is based on irreducible trinomial and a look-ahead technique in the dual basis multiplication, processed by one cell of tree structure in the most significant digit.	Limited to irreducible trinomial only, and cannot be applied for general multivariate polynomial operations	[150, 151]
Hankel matrix-vector Multiplier	This is an iterative Fast Fourier Transform based sparse matrix multiplication method with high degree of accuracy	Slower throughput as compared to most of the methods	[154–156]
Digit-serial Karatsuba Multiplier	This is a fast multiplication algorithm for multi-precision numbers with O($m^{1.58}$) complexity	Less throughput due to longer gate delays	[147, 148]
Chiou's Multiplier	The multiplication block is decomposed in four mutually independent sub-multiplication units, which improves theperformance	Less scalability and more hardware resource intense process	[159]
Toeplitz matrix-vector Multiplier	A single digit is represented by 2-bits and polynomial multiplication in every clock cycle	Higher resource utilization for hardware implementation	[157]
Systolic Karatsuba Multiplier	Efficient implementation of single variable NIST polynomial based multiplier	Less throughput, less scalable and more power consumption	[149]
Number Theoretic Transform Multiplier	Efficient implementation of single variable Kyber scheme polynomial for FFT like architecture	Only works for CRYSTALS-KYBER PQC scheme and limited for $x^n + 1$ polynomials ring	[152, 153]

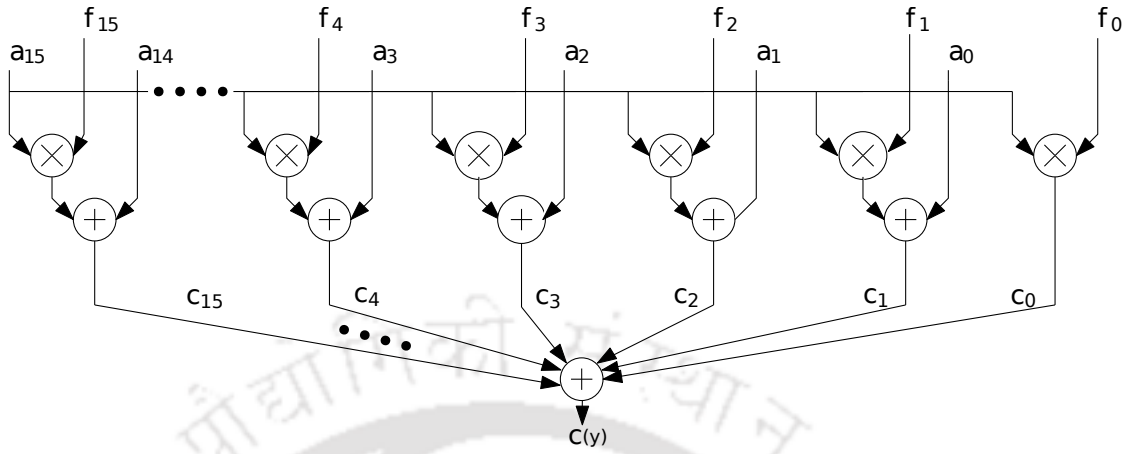


Figure 3.1: The structural construction of GF (2^{16}) multiplier for a single variable polynomial

According to Galois field, multiplication and addition are the main operations in polynomial modular multiplication [170]. Therefore, if $a(y)$ and $b(y)$ be two polynomials in $GF(2^m)$, the product polynomial $c(y)$ can be expressed as equation 3.1.

$$c(y) = a(y).b(y) \mod f(y) \tag{3.1}$$

Initially, $a(y)$ is multiplied with the coefficients of $b(y)$ sequentially. In this procedure, $a(y)$ is multiplied first with y, thereafter $a(y)y$ modulo $f(y)$ is determined. Subsequently, $a(y)$ is multiplied with y^2 and $a(y)y^2$ modulo $f(y)$ is calculated. This process continues until the limit of Galois field dimension is reached. Multiplying $a(y)$ with $b(y)$ and then division by $f(y)$ results into equation 3.2.

$$(a_{15}f_{15} + a_{14})y^{15} + (a_{15}f_{14} + a_{14})y^{14} \ldots\cdots + (a_{15}f_1 + a_0)y + a_{15}f_0 \tag{3.2}$$

An univariate SISO (*serial-in-serial-out*) modulo multiplier for polynomial modulo multiplication is proposed in [171]. The computational complexity of SISO multiplier is $O(m)$. This algorithm is suitable for cryptographic applications due to less area utilization and higher output rate. All the relevant data and parameters related with SISO multiplier over GF (2^m) can be found in [171].

The multivariate polynomial residue number system (MPRNS) based algorithm is presented in [172] to compute equation 3.3 mentioned below, where, $A(x)$, $B(x)$ and $C(x) \in R[x]$.

$$C(x) = A(x).B(x). \mod \prod_{i=1}^{L}(x_i^{N_i} \pm 1) \tag{3.3}$$

The quotient polynomial ring, $R[x] = Z_p/\prod_{i=1}^{L}(x_i^{N_i} \pm 1)$, has coefficients from modular ring Z_p. The existence of an isomorphism between rings $R[x]$ and $Z_p^{N_1,N_2.....N_L}$ forms the basis of MPRNS(L) for L-variate polynomials.

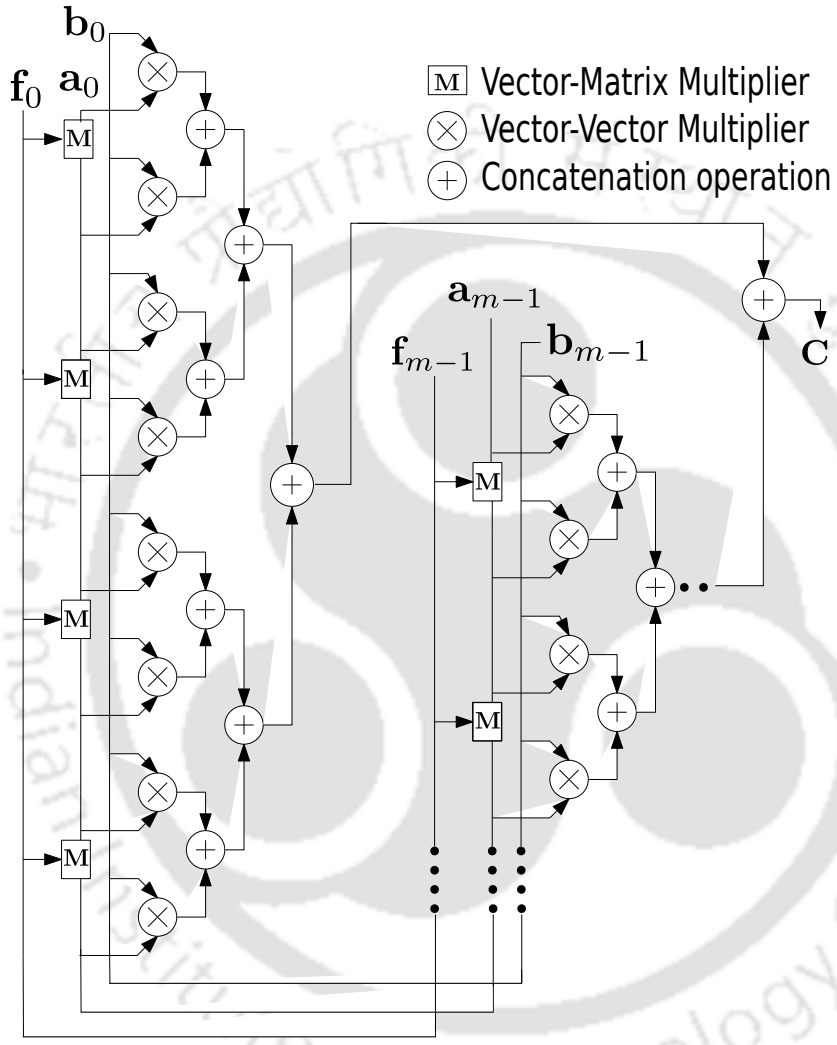


Figure 3.2: The Structural Construction of GF (2^{16}) Multiplier for Multivariate Polynomial

The Figure 3.2 exhibits basic structural organization of the multiplication of two polynomials. This architecture can be scaled up theoretically for multiple polynomials by implementing identical blocks. Because of its sequential architecture, throughput of this method is limited and, area, delay and power penalties are imposed.

The motivation of the proposed work is to design an area, power and delay efficient scalable generic multivariate polynomial multiplier to improvise performance of post quantum lattice cryptography algorithms. The primary advantage of the proposed approach is the efficient management of polynomial

coefficients, which is achieved by employing tensor based methods. In this method, tensor matrix is computed only once and is used repeatedly to perform any modular multiplication. Therefore, hardware complexity of this operation reduces significantly by incorporating tensor matrix method for polynomial modulo operations, which is discussed comprehensively in the next section.

3.3 Tensor Based Polynomial Modulo Multiplication

3.3.1 Tensor Basics

A tensor, T, is a multi-linear map, such that $T : V_1 \times V_2 \times \ldots V_n \rightarrow W$, where $V_1, V_2, \ldots, V_n$ and W are the vector spaces of finite dimensions. If vector spaces have a fixed base, then tensor can be expressed as a multidimensional array. For a bilinear map, one element each from two vector spaces maps into an element in the third vector space. Let V_1 and V_2 be two vector spaces, a bilinear map B, takes one element each from the vector V_1 and V_2 and maps it to a third vector V_3 using the following steps.

(i) $B : V_1 \times V_2 \leftarrow V_3$

(ii) If $v_1 \in V_1$ is fixed, then $v_1 \rightarrow B(v_1, v_2)$ is a linear function from $V_1 \rightarrow V_3$

(iii) If $v_2 \in V_2$ is fixed, then $v_2 \rightarrow B(v_1, v_2)$ is a linear function from $V_2 \rightarrow V_3$

Here, tensor product of the linear maps is described briefly for the completeness. Let $L_1 : V_1 \rightarrow V_2$ and $L_2 : W_1 \rightarrow W_2$ be two linear maps, then the tensor product of two linear maps is a linear map, $L_1 \otimes L_2 : V_1 \otimes W_1 \rightarrow V_2 \otimes W_2$, can be expressed by equation 3.4.

$$(L_1 \otimes L_2)(v_1 \otimes w_1) \rightarrow L_1(v_1) \otimes L_2(w_1) \tag{3.4}$$

A tensor product of two linear maps, A and B, can be represented as shown below.

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{13}B & \cdots & a_{2n}B \\ . & . & . & . \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$$

Therefore, it can be stated that the polynomial multiplication is a tensor map, because polynomials can be expressed as vectors in polynomial space of n variables (i.e $x_1, x_2, \ldots, x_n$ with the maximum

degree of each variable $< d$). This in turn forms a vector space, V, of dimension d^n. Multiplication of polynomials in V *modulo* P can be considered as a function, f, which maps two elements of V into another element of V. Here, $p \in \mathbb{F}_2\ [x_1, x_2, \ldots, x_n]$, such that the degree of $x_1, x_2, \ldots, x_n$ in any monomial is d, i.e. $f : V \times V \rightarrow V$. It is to mention that P is an irreducible polynomial in $\mathbb{F}_2$. This manifests f to a bilinear map enabling it to be employed for the tensor formulation.

The general modulo multiplication method requires multiplication of polynomials, if quotienting polynomial or reducible polynomial is changed. In this section, an alternative tensor matrix based method for modulo multiplication is discussed, which considerably reduces computational effort. As stated above, the main advantage of tensor matrix method is to compute tensor only once for a particular reducible polynomial to perform modulo multiplication. For a given reducible polynomial and the corresponding possible set of quotienting polynomials, a set of tensor coefficients is determined, which is employed to compute all the modulo multiplications until reducible polynomial is changed. In case of lattice polly cracker based methods, for a particular key-pair and the security parameters, the reducible polynomials do not alter much. Therefore, tensor matrix method can be of a great advantage to reduce compute intensive matrix operations. This method is elaborated below with a suitable example for the completeness.

3.3.2 The Tensor Matrix Method

The proposed tensor based multiplication technique is derived from fully homomorphic encryption scheme reported in [173]. The generic description of our proposed method is given below.

Let p_1, p_2 are the polynomials in ideal $\mathcal{I}_{\leq d}$ and $y \in \mathbb{Z}_q^n$, where $\mathbb{Z}$ is an integer and q is a prime number. $\mathbb{Z}_q$ denotes a finite field of cardinality q and $p_1(y).p_2(y) = p_1p_2(y) = p(y)$. Therefore, $p_1.p_2 \in \mathcal{I}_{\leq 2d}$, which is a vector subspace of $\mathbb{Z}_q[x_1, \ldots, x_l]_{\leq 2d}$. Here, dimensions l and d are less than the dimension of subspace n. After the multiplication, let assume that the dimension of this subspace be n', where $n' > n$. We now choose $(n' - n)$ additional points in $\mathcal{I}_{\leq 2d}$, which span the vector space $\mathbb{Z}_q^{n'}$ to evaluate the polynomial and are given below.

$$p(y_{n+1}) = \sum_{i=1}^{n} \beta_i.p(y_i) + \sum_{i=n+2}^{n'+1} \beta_i.p(y_i) \tag{3.5}$$

There exists γ_i^j for $n + 2 \leq i \leq n' + 1$ and $1 \leq j \leq n$, such that for all $p \in \mathcal{I}_{\leq d}$. Here, $\lambda_{s,j}$ are the coefficients of each terms and γ_i^j are the constants. Thus, equation 3.5 can be represented as equation

3.6.

$$p(y_i) = \sum_{j=1}^{n} \gamma_i^j \lambda_{s,j}.p(y_j) \text{ for } n+2 \leq i \leq n'+1 \tag{3.6}$$

From the above mentioned steps, we can observe that a linear transformation from $\mathcal{I}_{\leq 2d}$ to $\mathcal{I}_{\leq d}$ can be obtained, which is depicted in the form of a matrix L in $\mathbb{Z}_q^{(n+1)\times(n'+1)}$ below.

$$L = \begin{bmatrix} L_1^{n\times n} & 0^{n\times 1} & L_2^{1\times(n'-n)} \\ L_3^{1\times n} & 1 & L_4^{1\times(n'-n)} \end{bmatrix} \in \mathbb{Z}_q^{(n+1)\times(n'+1)}$$

To formulate $p(y)$ having dimension n', an integer is added to $(n+1)^{th}$ entry of $p(y)$. This is equivalent to $p_1p_2(y_{n+1})$ mod q and is being obtained using a transformation matrix, B, as shown below.

$$B = \begin{bmatrix} I_n & 0 & 0 \\ \beta_1 \dots \beta_n & 1 & \beta_{n+2} \dots \beta_{n'+1} \\ 0 & 0 & I_{(n'-n)} \end{bmatrix}$$

Now, the vector $p(y)$ is multiplied with L, which generates $p' = \wedge_s LB.p(y)$. Further, modular multiplication of p' is obtained using $p_{mul} = \lfloor p' \rfloor$ mod $q \in \mathbb{Z}_q^{n+1}$. Here, $\wedge_s$ is a diagonal coefficient matrix.

In order to validate correctnes of the above mentioned method, a tensor matrix, $\mathcal{M}$, based scheme is formulated. Lets consider a bilinear map $\mathcal{B}_{\mathcal{M}} : \mathbb{Q}^{(n+1)} \times \mathbb{Q}^{(n+1)} \rightarrow \mathbb{Q}^{(n+1)}$, which can be represented by a tensor $\mathcal{M} \in \mathbb{Q}^{(n+1)(n+1)(n+1)}$. $\mathcal{B}_{\mathcal{M}}(p_1, p_2) = [p_1^T M_1 p_2, \dots\dots, p_1^T M_{n+1} p_2]^T$, where $\mathbb{Q}$ is a rational number. $M_1, \dots\dots, M_{n+1}$ are the slices of tensor $\mathcal{M}$ and is depicted as $\mathcal{M} = \mathcal{N} \times_1 (I)^T \times_2 (I)^T \times_3 \wedge_s LB$. Here, I is an identity matrix and, $\mathcal{N}(i,i,i) = (\lambda_{s,i}^{-1})^2$ $\forall i \neq n+1$, $\mathcal{N}(n+1, n+1, n+1) = \frac{2}{q}$ and $\mathcal{N}(i,j,k) = 0$ everywhere else. Further, $\mathcal{B}_{\mathcal{M}}(p_1p_2) = p_1p_2$ mod q, therefore, $p_{mul} = \lfloor \mathcal{B}_{\mathcal{M}}(p_1.p_2) \rfloor$. The vector $\mathcal{B}_{\mathcal{M}}(p_1p_2)$ is the sum of two vectors v and v', where v is a vector with integer entries, which is equivalent $p(y)$ mod q. $v\prime$ is $(0, 0, \dots, 0, \eta)$, where η can be described as follows.

$$\eta = \frac{2}{q}(p_1 \lfloor \frac{q}{2} \rfloor + e_1 + qJ_1)(p_2 \lfloor \frac{q}{2} \rfloor + e_2 + qJ_2)$$

$$= p_1p_2 \lfloor \frac{q}{2} \rfloor - \frac{p_1p_2}{2q} + \frac{q-1}{q}(p_1e_2 + p_2e_1) + (2e_1 - p_1)J_2 +$$

$$(2e_2 - p_1)J_1 + \frac{2}{q}e_1e_2 + q(p_1J_2 + p_2J_1 + 2J_1J_2)$$

Therefore, $\lfloor \mathcal{B}_{\mathcal{M}}(p_1p_2) \rfloor$ mod $q = \lfloor \eta \rfloor$ mod q. Here, e_1 and e_2 are residue error terms due to performing

operations in $\mathbb{Q}$ space, and the values of $|J_1|$ and $|J_2|$ are less than the norm of an identity matrix.

In our proposed work, the above mentioned modulo multiplication method is implemented over binary field. Therefore, coefficients are represented as either 0 or 1 in the proposed method and modulo multiplication is performed by multiplying input vectors with tensor matrix directly. It is to mention that all the other parameters during multiplication, i.e. e_1, e_2, J_1, J_2, are considered zero due to $GF(2^m)$. An example is given below depicting steps taken by the proposed modulo multiplier.

An Example of the Complete Tensor Operation

Lets consider two multivariate polynomials, $a = (xy + 1)$ and $b = (xy^2 + x)$, and a quotienting polynomial $f = x^2y + xy + 1$. For polynomial ring $\langle f_1 \rangle$, the reducible polynomial is taken as $y^3 + 1$ and $\{1,\ y,\ (1 + y),\ (y^2),\ (1 + y^2),\ (y + y^2),\ (1 + y + y^2)\}$ are the elements of $\langle f_1 \rangle$ (for reference see Table 3.2).

Table 3.2: Table to Find Elements in $\langle f_1 \rangle$

.	1	y	$1+y$	y^2	$1+y^2$	$y+y^2$	$1+y+y^2$
1	1	y	$1+y$	y^2	$1+y^2$	$y+y^2$	$1+y+y^2$
y	y	y^2	$y+y^2$	1	$1+y$	$1+y^2$	$1+y+y^2$
$1+y$	$1+y$	$y+y^2$	$1+y^2$	$1+y^2$	$y+y^2$	$1+y$	0
y^2	y^2	1	$1+y^2$	y	$y+y^2$	$1+y$	$1+y+y^2$
$1+y^2$	$1+y^2$	$1+y$	$y+y^2$	$y+y^2$	$1+y$	$1+y^2$	0
$y+y^2$	$y+y^2$	$1+y^2$	$1+y$	$1+y$	$1+y^2$	$y+y^2$	0
$1+y+y^2$	$1+y+y^2$	$1+y+y^2$	$1+y+y^2$	0	0	0	$1+y+y^2$

Therefore, to perform modular multiplication, the expression, $\frac{(xy+1)*(xy^2+x)}{\langle x^2y+xy+1 \rangle}$ is to be computed, which results $xy^2 + xy + y^2 + 1$ as an output. As we know that the outcome of modular multiplication should always be less than the quotienting polynomial $f = x^2y + xy + 1$, its output comprises of the following coefficients only, i.e. $\{xy^2,\ x^2,\ xy,\ y^2,\ x,\ y,\ 1\}$. These coefficients are the part of either f or $\langle f_1 \rangle$, and the total number of coefficients is equal to the cardinality of $\langle f_1 \rangle$. This leads to the formulation of seven tensor matrices for the above mentioned seven coefficients. It can be seen that the computation of each tensor is performed independently, and each element of a particular tensor is evaluated in parallel as well, making this entire process embarrassingly parallel. The tensor elements are computed by finding remainder, which is obtained by dividing all the monomials of Table 3.3 by $f = x^2y + xy + 1$. It is to mention that the monomials are generated by using above mentioned seven

coefficients.

Table 3.3: Table to Find the Product of Monomials

.	xy^2	x^2	xy	y^2	x	y	1
xy^2	x^2y^4	x^3y^2	x^2y^3	xy^4	x^2y^2	xy^3	xy^2
x^2	x^3y^2	x^4	x^3y	x^2y^2	x^3	x^2y	x^2
xy	x^2y^3	x^3y	x^2y^2	xy^3	x^2y	xy^2	xy
y^2	xy^4	x^2y^2	xy^3	y^4	xy^2	y^3	y^2
x	x^2y^2	x^3	x^2y	xy^2	x^2	xy	x
y	xy^3	x^2y	xy^2	y^3	xy	y^2	y
1	xy^2	x^2	xy	y^2	x	y	1

In the tensor matrix, presence of an element in the remainder is marked by 1 or else as 0. The two main operations of tensor matrix method are described below.

Multiply: Each element of the vector is multiplied with other elements in all possible ways, which is tabulated in Table 3.3.

Quotienting: After the multiplication, products are divided by the quotienting polynomial to get a modular output. This operation is a critical step in the tensor formation followed by multiplication of the next polynomial. All the steps of this process are depicted below using various examples in Figure 3.3. In $Fig: 3(a)$, an example of dividing the product by a quotienting polynomial is presented. The division continues until a leading term of the remainder is no longer divisible by the quotienting polynomial. Thereafter, it is ascertained whether the leading term is further divisible by a reducible polynomial or not. Once the division is over, remainder is recorded. Here, it is analyzed whether xy^2 is present in the remainder or not, and accordingly it is set to 1 or 0 in the tensor matrix. The Table 3.4 exhibits detailed output of $Fig: 3(a)$. In $Fig: 3(b)$, polynomial, x^4, is divided by a reducible polynomial $y^3 + 1 = 0$. In order to perform division, x^4 is multiplied with y^3, so that it becomes divisible by the quotienting polynomial. In $Fig: 3(c)$, it can be observed that xy^4 is not divisible by quotienting polynomial, but is directly divisible by reducible polynomial. Therefore, if the product is neither divisible by quotienting polynomial nor by reducible polynomial, it is considered as remainder, and accordingly tensor matrix elements are formulated.

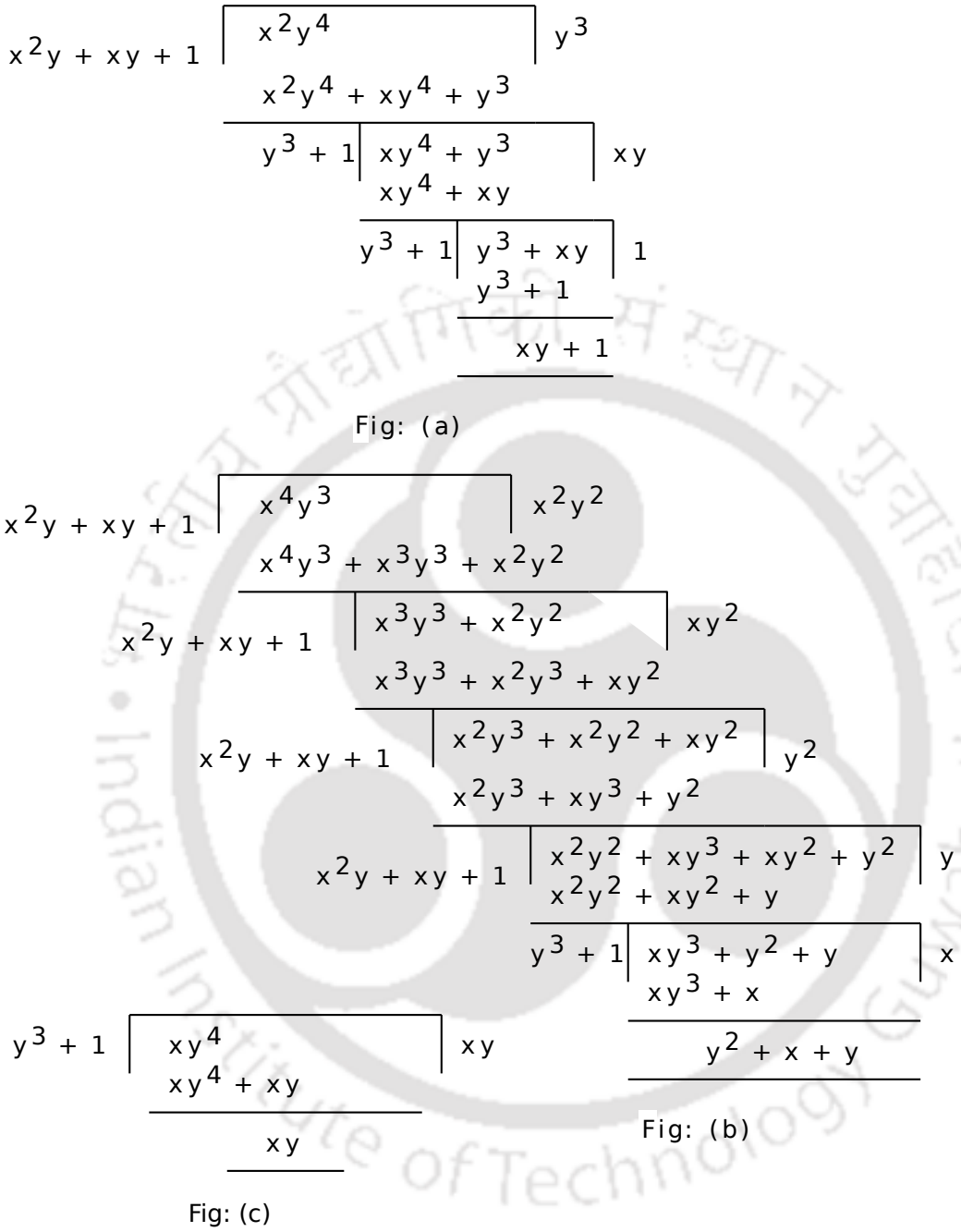


Figure 3.3: Fig: (a) Divide by quotienting and reducible polynomial. Fig: (b)Append the product with y^3 and then proceed. Fig: (c) Divide directly by reducible.

Table 3.4: Remainders for construction of Tensor Matrices

Indices	Check for remainder	Remainder	Indices
a_{00}	$x^2y^4/\langle x^2y+xy+1\rangle$	$xy+1$	0
a_{01}	$x^3y^2/\langle x^2y+xy+1\rangle$	xy^2+xy+y	1

a_{02}	$x^2y^3/\langle x^2y+xy+1\rangle$	y^2+x	0
a_{03}	$xy^4/\langle x^2y+xy+1\rangle$	xy	0
a_{04}	$x^2y^2/\langle x^2y+xy+1\rangle$	xy^2+y	1
a_{05}	$xy^3/\langle x^2y+xy+1\rangle$	x	0
a_{06}	$xy^2/\langle x^2y+xy+1\rangle$	xy^2	1
a_{10}	$x^3y^2/\langle x^2y+xy+1\rangle$	xy^2+xy+y	1
a_{11}	$x^4y^3/\langle x^2y+xy+1\rangle$	y^2+x+y	0
a_{12}	$x^3y/\langle x^2y+xy+1\rangle$	$xy+x+1$	0
a_{13}	$x^2y^2/\langle x^2y+xy+1\rangle$	xy^2+y	1
a_{14}	$x^3y^3/\langle x^2y+xy+1\rangle$	xy^2+y^2+x	1
a_{15}	$x^2y/\langle x^2y+xy+1\rangle$	$xy+1$	0
a_{16}	$x^2y^3/\langle x^2y+xy+1\rangle$	y^2+x	0
a_{20}	$x^2y^3/\langle x^2y+xy+1\rangle$	y^2+x	0
a_{21}	$x^3y/\langle x^2y+xy+1\rangle$	$xy+x+1$	0
a_{22}	$x^2y^2/\langle x^2y+xy+1\rangle$	xy^2+y	1
a_{23}	$xy^3/\langle x^2y+xy+1\rangle$	x	0
a_{24}	$x^2y/\langle x^2y+xy+1\rangle$	$xy+1$	0
a_{25}	$xy^2/\langle x^2y+xy+1\rangle$	xy^2	1
a_{26}	$xy/\langle x^2y+xy+1\rangle$	xy	0
a_{30}	$xy^4/\langle x^2y+xy+1\rangle$	xy	0
a_{31}	$x^2y^2/\langle x^2y+xy+1\rangle$	xy^2+y	1
a_{32}	$xy^3/\langle x^2y+xy+1\rangle$	x	0
a_{33}	$y^4/\langle x^2y+xy+1\rangle$	y	0
a_{34}	$xy^2/\langle x^2y+xy+1\rangle$	xy^2	1
a_{35}	$y^3/\langle x^2y+xy+1\rangle$	1	0
a_{36}	$y^2/\langle x^2y+xy+1\rangle$	y^2	0
a_{40}	$x^2y^2/\langle x^2y+xy+1\rangle$	xy^2+y	1
a_{41}	$x^3y^3/\langle x^2y+xy+1\rangle$	xy^2+y^2+x	1
a_{42}	$x^2y/\langle x^2y+xy+1\rangle$	$xy+1$	0

a_{43}	$xy^2/\langle x^2y+xy+1\rangle$	xy^2	1
a_{44}	$x^2y^3/\langle x^2y+xy+1\rangle$	y^2+x	0
a_{45}	$xy/\langle x^2y+xy+1\rangle$	xy	0
a_{46}	$x/\langle x^2y+xy+1\rangle$	x	0
a_{50}	$xy^3/\langle x^2y+xy+1\rangle$	x	0
a_{51}	$x^2y/\langle x^2y+xy+1\rangle$	$xy+1$	0
a_{52}	$xy^2/\langle x^2y+xy+1\rangle$	xy^2	1
a_{53}	$y^3/\langle x^2y+xy+1\rangle$	1	0
a_{54}	$xy/\langle x^2y+xy+1\rangle$	xy	0
a_{55}	$y^2/\langle x^2y+xy+1\rangle$	y^2	0
a_{56}	$y/\langle x^2y+xy+1\rangle$	y	0
a_{60}	$xy^2/\langle x^2y+xy+1\rangle$	xy^2	1
a_{61}	$x^2y^3/\langle x^2y+xy+1\rangle$	y^2+x	0
a_{62}	$xy/\langle x^2y+xy+1\rangle$	xy	0
a_{63}	$y^2/\langle x^2y+xy+1\rangle$	y^2	0
a_{64}	$x/\langle x^2y+xy+1\rangle$	x	0
a_{65}	$y/\langle x^2y+xy+1\rangle$	y	0
a_{66}	$1/\langle x^2y+xy+1\rangle$	1	0

The remainders can be tabulated as shown in Table 3.4, and as mentioned earlier, each of these step can be calculated in parallel. For generating tensor for a particular coefficient, if it appears in the remainder of a monomial after getting it divided by the quotienting polynomial, that particular monomial index in the tensor is marked as "1", else "0". For example, to generate the tensor of xy^2, elements a_{01}, a_{04}, a_{06}, a_{10}, a_{13}, a_{14}, a_{22}, a_{25}, a_{31}, a_{34}, a_{40}, a_{41}, a_{43}, a_{52} and a_{60} are marked as "1" in the tensor matrix of xy^2, because xy^2 term is present in these tensor elements corresponding to the remainders of monomials in Table 3.4. The tensor matrix for xy^2 is given below as a reference.

$$\mathcal{T}_{xy^2}: \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Similarly for x^2, xy, y^2, x, y and 1, the tensor matrices generated using Table 3.4 are shown below.

$$\mathcal{T}_{x^2}: \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathcal{T}_{xy}: \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathcal{T}_{y^2}: \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathcal{T}_{x}: \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \mathcal{T}_{y}: \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \mathcal{T}_{1}: \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The $\frac{(xy+1)*(xy^2+x)}{\langle x^2y+xy+1\rangle}$ is further analyzed using above mentioned tensors. After lexicographic ordering, $a = (xy+1)$ and $b = xy^2 + x$ can be expressed as,

$$a = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}; \; b = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

As we know that the product of modular multiplication can be represented as the combination of

$\{xy^2, x^2, xy, y^2, x, y, 1\}$. In order to find whether terms are present in the final output or not, a and b are multiplied with each tensor, i.e. $a \times \mathcal{T} \times b^T$. Here, $\mathcal{T}$ is a tensor. For example, to find the presence of xy^2 in the final output, following steps can be taken. If the result is "1", it indicates that the term is present in final output, otherwise not.

$$a \times \mathcal{T}_{xy^2} \times b = 1$$

By applying similar steps, the presence of other coefficients can also be determined.

$$x^2 :\Rightarrow a \times \mathcal{T}_{x^2} \times b = 0; \quad xy :\Rightarrow a \times \mathcal{T}_{xy} \times b = 1;$$

$$y^2 :\Rightarrow a \times \mathcal{T}_{y^2} \times b = 1; \quad x :\Rightarrow a \times \mathcal{T}_{x} \times b = 0;$$

$$y :\Rightarrow a \times \mathcal{T}_{y} \times b = 0; \quad 1 :\Rightarrow a \times \mathcal{T}_{1} \times b = 0.$$

Using the above mentioned steps, the output of modulo multiplication is determined as $xy^2 + xy + y^2 + 1$, which matches with the algebraic output.

3.3.3 Complexity Analysis

The proposed method can be divided into three steps for the complexity analysis. The first step is ordering random multivariate polynomial equations in graded lexicographic order. The second step is to generate a tensor matrix, and the last step is to multiply it with multivariate polynomials to get the modular output. The most recent works in the polynomial modulo multiplication domain only deal with single variable polynomial equations, whereas our proposed scheme can also be applied to multivariate polynomials. The dimension of the tensor matrix is related to the number of variables and the degree of the polynomials.

For lexicographic ordering, the time complexity for a single variable polynomial with n-terms is $(n-1)$ or $O(n)$, whereas the time complexity of the polynomial having n-terms of m-variables is $(nm-1)$ or $O(mn)$. For ordering, monomials are generated using coefficients taken from reducing and quotienting polynomials for generating tensor matrices. Later, every monomial is divided by a quotienting polynomial, and a table is generated for storing the remainders of the individual monomial. Further, a $n \times n$ binary tensor matrix for each coefficient is generated individually by finding its presence in the remainders. Monomials can be generated in parallel using n^2 processing elements in a single step. Similarly, remainders for each monomial are also computed in parallel using n^2 processing

elements. The time complexities of finding monomials and remainders are $O(1)$ and $O(\lceil log_2 n \rceil)$ [174], respectively. It is to mention that for multivariate polynomials, the time complexity to find the remainder is $O(m\lceil log_2 n \rceil)$. Since each tensor can be computed in parallel for all the coefficients, total time complexity for generating all the tensors is $O(\lceil log_2 n \rceil)$ for $m = 1$, i.e. single variable polynomials. As we know, each tensor and input polynomial vectors (a and b) are represented in binary. Therefore, the step $a \times \mathcal{T} \times b^T$ employed for finding the output of the modular multiplier can be performed in $(k+1)n+2$ steps. The multiplication of row vector a with a column of $\mathcal{T}$ takes $O(1)$ using n processing elements. Later, the addition of the binary partial products is also performed in $O(k)$, where k is the depth of LUTs mimicking XOR operations. Further, the resultant row vector after performing $a \times \mathcal{T}$ is multiplied with the column vector b, and it takes $O(1+k)$ to compute the final result of the modular multiplication. Thus, the worst-case time complexity of modular multiplication is $(n-1)+\lceil log_2 n \rceil+(k+1)n+(k+1)$ or $O(n)+\lceil log_2 n \rceil+(k+1)n+(k+1)$ or $O(n)+\lceil log_2 n \rceil+(k+1)(n+1)$. In our case, for a single variable with 256 terms input, where the maximum degree of the variable is 256, modulo multiplication is performed in $O(1)$, i.e., $k = 1$. This is because of performing additions of 256 bits in a single clock cycle. It is to be noted that $k \ll n$ in general, thus, $(k+1)(n+1) \approx (n+1)$. This transforms the overall time complexity of modular multiplication to $O(n) + \lceil log_2 n \rceil + 2n + 2$, which can also be validated using Table 3.9 by analyzing the delay and number of variables.

As we know, a tensor matrix is sparse and stored using sparse matrix storage formats, such as Compressed Row Storage, Harwell-Boeing etc., having space complexity $O(n_{nz})$. Here n_{nz} is the total number of "1" in a given tensor matrix. Thus, space complexity is $O(n_{nz}n)$ for all the tensors. Similarly, for storing monomials, the space complexity is $O(n^2)$ and for storing lexicographically ordered inputs, the space requirement is $O(n)$. Thus, the worst case space complexity of modular multiplication is $O(n^2 + n_{nz}n + n)$, i.e., $O(n^2)$ for a single variable polynomial.

Area complexity can be determined by the total number of gates utilized in designing a particular algorithm. The most resource-intensive part of the proposed multiplier is tensor generation operation, where the remainder is evaluated for every possible monomial by dividing it using an irreducible polynomial, taking $log_2 n$ steps to compute. Every step utilizes n AND and $(n-1)$ XOR operations to produce a remainder in $log_2 n$ steps, and there are n such monomial divisions computed simultaneously. Therefore, the total space required for a tensor computation is $A_A[n^2 log_2 n] + A_X[(n(n-1))log_2 n]$, where A_A and A_X are the area required for single 2-input AND and XOR gates. Subsequently,

Table 3.5: Comparison of Time and Space Complexities, and Latency

Test cases	Type	Proposed	Reyhani-Hasan [144]	Zhang-Parhi [165]	Others	
Trinomial ($P(x) = x^n + x^m + 1$) $n > m > 1$ e is constant)	Time	$(n-1)+\lceil log_2 n \rceil+(k+1)(n+1)$	$n^2 + (2 + \lceil log_2(n-1) \rceil)(n^2-1)$	$n^2 + (2 + \lceil log_2 n \rceil)(n^2-1)$	$(\lceil (n-1)/e \rceil + 3 + \lceil log_2 e \rceil)^{\#}$	[175]
	Space	$A_A[n^2(log_2 n + (r+1))] + A_X[(n^2-n)(log_2 n+(r+1))]$	$A_A n^2 + A_X(n^2-1)$	$A_A n^2 + A_X(n^2-1)$	$A_{NA}n^2+A_{XN}(n^2-1)+A_R(n^2+3n+1)$	
	Latency	$T_A + T_X[(\frac{n}{r} - log_2 n + 1)]$	$T_A + (2 + \lceil log_2(n-1) \rceil)T_X$	$T_A+(1+\lceil log_2 n \rceil)T_X)$	$(T_{NA} + T_{XN})$	
Pentanomial ($P(x) = x^n + x^{k3} + x^{k2} + x^{k1} + 1, 1 < k1 < k2 < k3 \leq \frac{n}{2}$)	Time	$(n-1)+\lceil log_2 n \rceil+(k+1)(n+1)$	$n^2 + (4 + \lceil log_2(n-1) \rceil)(n^2 - 2n - 3)$	$n^2 + (6 + \lceil log_2 n \rceil)(n^2+2n-3)$	-	
	Space	$A_A[n^2(log_2 n + (r+1))] + A_X[(n^2-n)(log_2 n+(r+1))]$	$A_A n^2 + A_X(n^2+2n-3)$	$A_A n^2+A_X(n^2+2n-3)$	-	
	Latency	$T_A + T_X[(\frac{n}{r} - log_2 n + 1)]$	$T_A + (4 + \lceil log_2(n-1) \rceil))T_X$	$T_A+(6+\lceil log_2 n \rceil))T_X$	-	
Single variable s-ESP ($P(x)=x^{ms} + x^{(m-1)s} + \cdots + x^s + 1, n = ms$)	Time	$(n-1)+\lceil log_2 n \rceil+(k+1)(n+1)$	$n^2+(1+\lceil log_2 n \rceil)(n^2-s)$	$n^2 + (1 + \lceil log_2 n \rceil)(n^2-s)$	-	
	Space	$A_A[n^2(log_2 n + (r+1))] + A_X[(n^2-n)(log_2 n+(r+1))]$	$A_A n^2 + A_X(n^2-s)$	$A_A n^2 + A_X(n^2-s)$	-	
	Latency	$T_A + T_X[(\frac{n}{r} - log_2 n + 1)]$	$T_A + (1 + \lceil log_2 n \rceil)T_X$	$T_A+(1+\lceil log_2 n \rceil)T_X$	-	
Generic Polynomial ($P(x) = x^n + x^{kt} + \cdots + x^{k2} + x^{k1} + 1, 1 \leq k1 < k2 < \ldots kt \leq \frac{n}{2}$)	Time	$(n-1)+\lceil log_2 n \rceil+(k+1)(n+1)$	$n^2 + (\lceil log_2(t+1) \rceil + \lceil log_2(\lceil \frac{t}{2} \rceil + 1) \rceil + \lceil log_2(n-1) \rceil)(n+t)(n-1)$	$n^2 + (2t + \lceil log_2 n \rceil)(n+t)(n-1)$	$1.78n^{log_2 3}+(4.91n^{log_2 3}-11n+8.88)(2log_2 n-1)$	[176]
	Space	$A_A[n^2(log_2 n + (r+1))] + A_X[(n^2-n)(log_2 n+(r+1))]$	$A_A n^2+A_X(n+t)(n-1)$	$A_A n^2+A_X(n+t)(n-1)$	$A_A(1.78n^{log_2 3}) + A_X(4.91n^{log_2 3} - 11n + 8.88)$	
	Latency	$T_A + T_X[(\frac{n}{r} - log_2 n + 1)]$	$T_A + (\lceil log_2(t+1) \rceil + \lceil log_2(\lceil \frac{t}{2} \rceil + 1) \rceil + \lceil log_2(n-1) \rceil)T_X$	$T_A+(2t+\lceil log_2 n \rceil)T_X$	$T_A + T_X(2log_2 n - 1)$	
Multivariate polynomial $P(x_1, x_2 \ldots x_m)$	Time	$(mn-1) + m\lceil log_2 n \rceil + (k+1)(n+1)$	-	-	-	
	Space	$A_A[n^2 m(log_2 nm + (r+1))] + A_X[(nm(n-1)(log_2 nm + (r+1))]$	-	-	-	
	Latency	$T_A + T_X[(\frac{nm}{r} - log_2 nm + 1)]$	-	-	-	

from Figure 3.4 it can be observed that the space complexity of the input vector and tensor matrix multiplication is $A_A n(r+1) + A_X(n-1)(r+1)$ for a single multiplication, where r is the number of parallel column multiplications (r-PCM). Similar to tensor generation, this step is also implemented in parallel, and for n such blocks, the total space complexity is $A_A n^2(r+1) + A_X(n^2-n)(r+1)$. Therefore, the overall space complexity of our proposed modulo multiplication method for a single variable polynomial is $A_A[n^2(log_2 n + (r+1))] + A_X[(n^2-n)(log_2 n + (r+1))]$ and for multiple variable polynomials is $A_A[n^2 m(log_2 nm + (r+1))] + A_X[(nm(n-1)(log_2 nm + (r+1))]$, where m is the number of variables. Time delay in terms of T_A (2-input AND gate delay), and T_X (2-input XOR gate delay) can be calculated after aggregating the total time taken by all the AND and XOR gate delays. Here, generation of individual tensor matrices and vector-matrix multiplication of input polynomials with tensors are mutually exclusive processes. The time delay of a single and multiple variable polynomials are $T_A + T_X[(\frac{n}{r} - log_2 n + 1)]$, and $T_A + T_X[(\frac{nm}{r} - log_2 nm + 1)]$. Note that n is the maximum degree of the input polynomial.

The comparison of time and space complexities and latency of the proposed method with [144], [165], [175], and [176] is presented in Table 3.5. A_{NA}, A_{XN}, T_{NA}, and T_{XN} in [175] are area and gate delays of 2-input NAND and XNOR gates. The comparison of the methods reported in [144], [165], [175], and [176] for specific values of n are presented in Table 3.6. It can be seen that for $\{(r,n)\} = \{(16,128);(32,256);(32,512)\}$ latencies are $(T_A + 2T_X)$, $(T_A + T_X)$ and $(T_A + 8T_X)$. These are the optimal latencies of the proposed multiplier for $n = 128$, 256 and 512, and are the least among all the other multipliers. It is to mention that the practical limit of latency cannot be reduced further even if more PCMs are employed. This would only increase dynamic power consumption without further reduction of the latency. Note that our proposed method does not depend on the polynomial expression of a single variable and can be easily extended to multivariate polynomials. It

Table 3.6: Complexity Comparison for Specific Values of n

Trinomial ($x^n + x^m + 1$, where $n > m > 1$); r-PCM is used									
Method	$n = 128$; $r = 16$;			$n = 256$; $r = 32$;			$n = 512$; $r = 32$;		
	#AND	#XOR	Delay	#AND	#XOR	Delay	#AND	#XOR	Delay
[176]	3888	9145	$(T_A + 14T_X)$	11664	29360	$(T_A + 16T_X)$	34992	90124	$(T_A + 18T_X)$
[144]	16384	16383	$(T_A + 9T_X)$	65536	65535	$(T_A + 10T_X)$	262144	262143	$(T_A + 11T_X)$
[165]	16384	16383	$(T_A + 8T_X)$	65536	65535	$(T_A + 9T_X)$	262144	262143	$(T_A + 10T_X)$
Proposed	131072	130048	$T_A + (\frac{128}{r} - 6)T_X = (T_A + 2T_X)$	589824	587520	$T_A + (\frac{256}{r} - 7)T_X = (T_A + T_X)$	2621440	2616320	$T_A + (\frac{512}{r} - 8)T_X = (T_A + 8T_X)$

can be observed from Table 3.5 that the time complexity and latency are linear, whereas the existing methods are quadratic or subquadratic. Further, the space complexity of many existing single variable polynomial multipliers is $O(n^{1+\varepsilon})$ [175] [176], whereas the space complexity of the proposed multiplier is $O(n^2 logn)$. Here, $0.5 < \varepsilon < 1$. This is due to not considering the sparsity of the tensors, which results in the quadratic space complexity.

3.4 Implementation Details

The proposed multiplier is implemented using MATLAB (software) and on FPGA platform (hardware). MATLAB based implementation is described in detail below.

3.4.1 MATLAB Implementation

A multivariate polynomial is represented as,

$$\begin{bmatrix} E_0 & E_1 & E_2 & \dots & E_n \end{bmatrix}$$

where, E_0 represents constant term and, rest of the terms in the polynomial and E_i hold information about the maximum degree of i^{th} variable for $i = 1 \cdots n$. For example discussed in the previous section, E_0, E_1 and E_2 are equal to 1, 2 and 2, respectively. Thus, the program takes an input in the vector form $[E_0, E_1, E_2, \dots E_n]$. It is followed by lexicographic ordering of the polynomials and, subsequently multiplication and quotienting operations are performed.

The following modules are designed to meet the desired goal. Their brief descriptions and algorithmic descriptions are given below.

• **Monomial Ordering Module**: In the proposed work, graded lexicographic ordering method is applied, where degree of a polynomial is given more weightage than the individual order of a variable. The algorithm 6 exhibits steps of the module implementation. We have modified standard lexicographic ordering algorithm [167–169] and have employed it to generate ordered input polynomial vectors for our proposed tensor multiplication. Let assume that the number of polynomial variables is m and the maximum degree of the each variable is n. This results in the total number of possible coefficients $l = (n+1)^m$. $A(X_1, X_2 \dots, X_m)$ and $B(X_1, X_2 \dots, X_m)$ denote unsorted and sorted polynomials of degree n, respectively. The algorithm which performs lexicographic ordering of $A(X_1, X_2 \dots, X_m)$, is described in Algorithm 6. Output of this algorithm is a binary array $S = \{s_1, s_2, \dots, s_l\}$, which is employed in all the subsequent operations in our proposed modulo multiplier. It is to mention that in

our proposed method, lexicographic ordered output exhibits the presence of any particular coefficient in the input polynomial vectors.

Algorithm 6: Graded Lexicographic Ordering

Input: $A(X_1, X_2, \ldots, X_m)$, m, n
Output: S=$\{s_1, s_2, \ldots, s_l\}$, where $l = (n+1)^m$
1 **Initialize:** p=number of coefficients in $A(X_1, X_2, \ldots, X_m)$; S= $\{0, \ldots, 0\}$; $temp = 0$;
2 $A_k = \prod_{j=1}^{m}(X_j^i)$ and $0 \leq i \leq n$, $1 \leq k \leq p$
3 **Initialize** $d[m]$**:** An integer array of size m to hold degree of each variable in A_k;
4 **begin**
5 **for** *($k = 1$ to p)* **do**
6 **for** *($j = 1$ to m)* **do**
7 $d_j =$ *degree of* $X_j \in A_k$;
8 $temp = \sum_{i=1}^{m}\{(n+1)^{m-i}.(n-d_i)\} + 1$;
9 $s_{temp} = 1$;
10 Re-initialize $d[m]$;
11 **return** S

Correctness of lexicographic ordering: Employing our lexicographic ordering algorithm on an unsorted m-variable and n-degree generic polynomial, the final ordered polynomial is $B(X_1, X_2, \ldots, X_m)$ $= \{(X_1^n X_2^n \ldots X_m^n), (X_1^n X_2^n \ldots X_m^{n-1}), \ldots, (X_1^n X_2^n \ldots X_m^0), \ldots, (X_1^0 X_2^0 \ldots X_m^1), (X_1^0 X_2^0 \ldots X_m^0)\}$. In Algorithm 6, it can be observed that the initialization of $d[m]$ can be performed in $O(m)$. Steps mentioned in Lines $8-11$ extract degree of individual variable in A_k and stores it in $d[m]$. This can also be conducted in $O(m)$. Steps stated in Lines $12-13$ determine location of A_k in $B(X_1, X_2, \ldots, X_m)$ in $O(1)$. Also, re-initialization of $d[m]$ is performed in $O(m)$. The whole process, i.e. Lines $6-15$, repeats p times taking $O(pm)$ steps. Thus, the total time complexity of Algorithm 6 is $O(pm)+O(m)$ or $O(pm)$. For single variable polynomials, $m = 1$, which enables Algorithm 6 to perform in $O(p)$ or in linear time. In reality, very less number of coefficients is present in the polynomial, i.e. $p \ll l$, which reduces computation time drastically. It is to be noted that $B(X_1, X_2, \ldots, X_m)$ is an ordered polynomial and a one-to-one map, S, exists between B and S for all of its ordered terms. It can be proved inductively that for p terms, if there is a linear mapping exists between B and S, then for $B_{p+1} \uplus B$, S_{p+1} is uniquely determined assuming that no two terms in A or B are same, and each term $\in B$ has a unique image in S.

For instance, we take two variables (x and y) and the maximum degree of a variable in the polynomial as three, i.e. $m = 2$, $n = 3$ and $l = (n+1)^m = (3+1)^2 = 16$. Here, all the possible

coefficients ordered lexicographically can be expressed as B={x^3y^3, x^3y^2, x^3y, x^3, x^2y^3, x^2y^2, x^2y, x^2, xy^3, xy^2, xy, x, y^3, y^2, y, 1}. These coefficients are uniquely mapped from 1 to 16 in S in an ordered manner. Now consider an input polynomial $(x^2y^3 + y^2 + x^3 + 1)$, where $p = 4$. Thus, in the output vector, there should be four "1s" only. Using Algorithm 6, four array indices can be computed using the expression $temp = ((n+1)^{m-1}.(n - d_1) + (n+1)^{m-2}.(n - d_2) + 1)$, where $\{d_1, d_2\}$ for $p = 1 \rightarrow 4$ are ({2,3}, {0,2},{3,0} and {0,0}), respectively. The array indices, thus calculated, are {4, 5, 14 and 16} for $x^3 + x^2y^3 + y^2 + 1$, which constitute the final binary output in graded lexicographic format as {0001100000000101}.

• **Monomial Multiplication Module**: This module multiplies two monomials to generate their product as shown in Table 3.3. The details of this module has already been explained in section 3.3.

Algorithm 7: Polynomial Quotienting Operation

Input: $D(x)$; $R(x)$; $Divi(x)$;
Output: Remainder $Rem = \frac{Divi(x)}{D(x)} \bmod R(x)$;

Initialize: $A = D(x)$; $B = R(x)$; $C = Divi(x)$; $E = 0$; $Rem = 0$;
begin
  while $(B \neq 0)$ **do**
  while$(a_0 == 0)$ **or** $(b_0 == 0)$ **do**
  if $(a_0 == 0)$ **then**
    $A = \frac{A}{x}$;
    $C = \frac{C + c_0.R(x)}{x}$;
  if $(b_0 == 0)$ **then**
    $B = \frac{B}{x}$;
    $E = \frac{E + e_0.R(x)}{x}$;
  end while
  if $(A > B)$ **then**
    $A = B + A$;
    $C = C + E$;
  else
    $B = B + A$;
    $E = C + E$;
  end while
  $Rem = C - A.E$;
  return Rem

• **Quotienting Module**: We employ a well-known Euclid's polynomial division algorithm to obtain remainders to generate tensor matrices. The algorithm 7 depicts the steps taken to perform quotienting operation. Since Euclid's polynomial division method is widely known, its detailed mathe-

matical description and hardware implementation can be found in [177–179]. In the Algorithm 7, $D(x)$, $R(x)$ and $Divi(x)$ denote quotienting, reducible polynomial and input polynomial to be quotiented, respectively. Here, *Rem* is the final remainder of the quotienting operation.

The ordering of the *input* vectors are determined using the by following steps of Algorithm 6. Thereafter, multiplier module is called recursively followed by quotienting module (Algorithm 7) until all the input polynomials are evaluated and tensor matrices are formulated. It is to mention that tensor matrices shown above are generated using our MATLAB based implementation incorporating all the above mentioned steps.

3.4.2 FPGA Implementation

Modulo multiplication of multivariate polynomials is designed using Verilog HDL. All the modules are synthesized and analyzed using Xiling Vivado platform [180] and, are implemented on Zynq-7000-Z020 and Artix-7-200T Xilinx FPGA boards. Detailed description of these modules are given below.

• **Top module**: This module receives two multivariate polynomials as input. This module is the main control unit of the proposed multiplier. Top module calls following modules in the hierarchy, which are *multipoly*, *divpoly*, *tensorPoly* and *mulVectMat*. These modules multiply monomials, divide the product by quotienting polynomial obtained from the previous module and formulate tensor matrices, which finally get multiplied by the input polynomials. The finite state machine (FSM) of multivariate polynomial modulo multiplication method is presented in Figure 3.5.

• **MultiPoly**: This module takes two monomials simultaneously as inputs and generates their product. Iteratively, it produces product of every combination of the monomials.

• **Divpoly**: This modules receives product obtained from MultiPoly module as an input and divides it first by quotienting polynomial. If the product is not divisible by quotienting polynomial, either reducible or irreducible polynomials are chosen to divide the product. In each iteration, this module generates division of every product obtained from *MultiPoly* module.

• **TensorPoly**: This module receives output of *DivPoly* module and generates tensor matrices.

• **MulVectMat**: Finally, modulo multiplication of polynomials employing tensors is performed by this module.

3.4.3 Vector Tensor Vector Multiplication

The architecture of multiplication unit of input polynomial vector and tensor matrix is presented in Figure 3.4. Initially, it is needed to generate tensor matrices $T = \{T_1, T_2, \ldots, T_x, \ldots, T_n\}$ specific to an irreducible polynomial, which is generated once if this polynomial is unchanged while performing different polynomial multiplications. The first polynomial $A = \{A_1, A_2, \ldots, A_n\}$ is multiplied with a tensor matrix T_x of dimension $(n \times n)$ column-wise and then multiplied with a second polynomial $B^\top = \{B_1, B_2, \ldots, B_n\}^\top$ to determine the presence of corresponding coefficient in the output.

Here, each column of T_x is multiplied with all the coefficient of polynomial A, which takes two clock cycles to compute the result. This implementation is noted as one parallel column multiplications (1-PCM) and can be performed upto r-PCM, where $r \leq n$. As all the coefficient of A, B and a tensor matrix T_x is known to us and stored in BRAM, therefore, all the operations related to PCM can be perform simultaneously. Therefore, as we increase the number of PCM operations we can reduce time requirement drastically with increase in minimal resource utilization and in Figure 3.4, 2-PCM is marked in green.

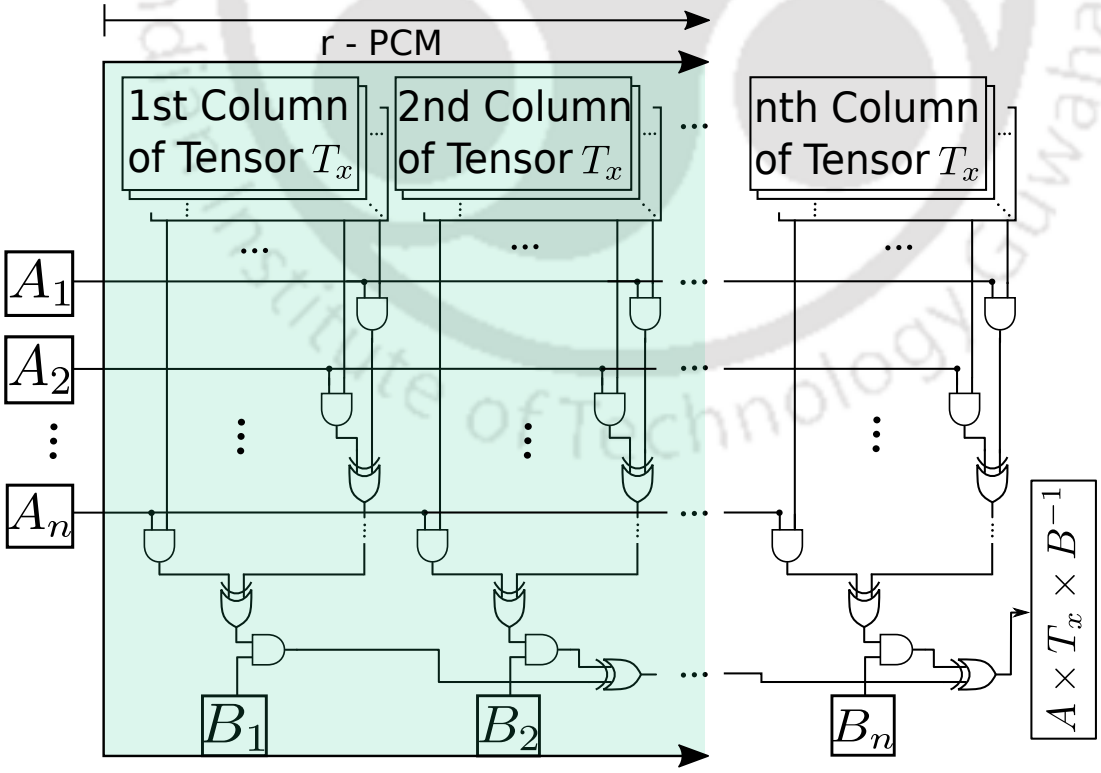


Figure 3.4: r-Parallel Column Multiplication (r-PCM) Architecture

Tables 3.7 and 3.8 present description of all the probable states and transitions with respect to

Table 3.7: State Description of the FSM and Dependency on Flag Registers

State	Specific Description	Flag
X_0	Check for valid data	Val, End
X_1	Compute monomial multiplication	Val, En0
X_2	Quotienting operation to generate remainders	En0, En1, Ovr
X_3	Formulate tensors from X_3	En1, En2
X_4	Vector tensor matrix	En2, En3
X_5	Check for further modulo division	En3, Ovr
X_6	Produce multiplication output	Ovr, End

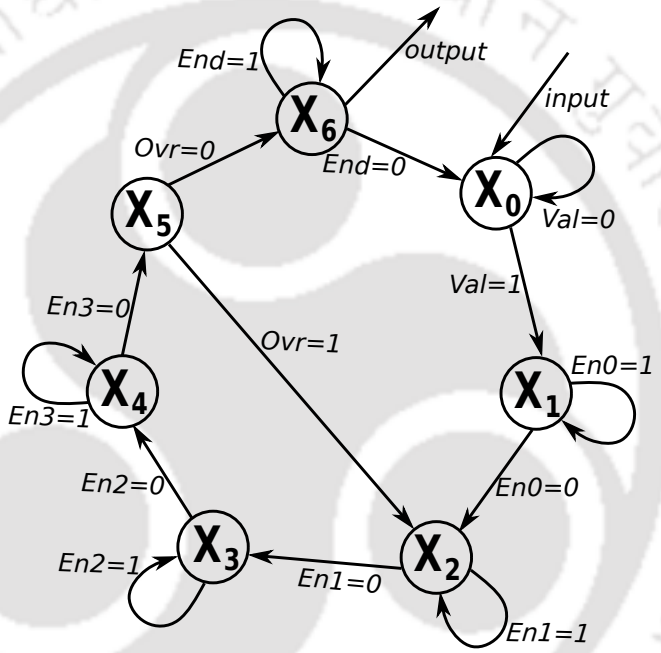


Figure 3.5: Finite State Machine of Main Module

7-bit flag register F={Val, En0, En1, En2, En3, Ovr, End}. It can be observed that there are no ambiguous state transitions, and for all the possible values of flag register, there is a unique transition from the present state to the next state. It is to mention that all inputs to the main multiplication module are assumed to be lexicographically ordered, therefore, ordering is not included in the main module of FSM. The X_0 remains in the same state until it gets lexicographically ordered inputs. As per the state diagram of top module described in Figure 3.5, state X_0 receives input and, checks for valid data and forward it to the next state. If data is not valid, the flag Val is set to "0" and it waits for a valid input until the valid input Val flag is set, and enters into state X_1. State X_1 computes multiplication of elements and stores it in a vector, if $En0$ is set. After completion of the multiplication, $En0$ flag resets to zero and state X_2 is activated. Based on the status of flag $En1$,

Table 3.8: State Transition of the FSM of Proposed Modulo Multiplier

	Flag register		Present State						
			X_0	X_1	X_2	X_3	X_4	X_5	X_6
Next State	**Val**	**0**	X_0	x	x	x	x	x	x
		1	X_1	x	x	x	x	x	x
	En0	**0**	x	X_2	x	x	x	x	x
		1	x	X_1	x	x	x	x	x
	En1	**0**	x	x	X_3	x	x	x	x
		1	x	x	X_2	x	x	x	x
	En2	**0**	x	x	x	X_4	x	x	x
		1	x	x	x	X_3	x	x	x
	En3	**0**	x	x	x	x	X_5	x	x
		1	x	x	x	x	X_4	x	x
	Ovr	**0**	x	x	x	x	x	X_6	x
		1	x	x	x	x	x	X_2	x
	End	**0**	x	x	x	x	x	x	X_0
		1	x	x	x	x	x	x	X_6

"x" is Don't Care condition

the division operation is performed in state X_2. Thereafter, if $En2$ is set, state X_3 formulates tensors and advances FSM into the next state. Subsequently, when $En3$ is set, state X_4 multiplies tensor with input polynomials. State X_5 checks whether modulo division is needed to constrain the output in the desired range or not. If it is affirmative, then it sets flag Ovr to "1" advancing FSM to state X_2, or else move to state X_6. The flag End is set to "1", which implies X_6 is ready to generate and to produce an output. If an external circuit is ready to take the output, then the flag End is reset from "1" to "0" and FSM re-enters into an initial state X_0. Additionally, an overview of the hardware implementation is shown in Figure 3.6.

There are two ways to prove correctness of an finite state machine (FSM) as mentioned in [181]. In the first method, an FSM can be split at two abstraction levels, i.e., high and low. The high level abstraction specifies abstract behaviour of an FSM, while the low level illustrates detailed hardware implementation of the FSM. Therefore, the correctness of an FSM can be addressed employing behavioural equivalence between these two processes, i.e. the abstract specifications and its hardware implementation. The second method is based on algebraic computational methodology, in which FSM algebraic techniques based on semantic transitions or syntactic transformations can be analysed for its correctness. The intermediate results obtained through hardware implementation match with Tables

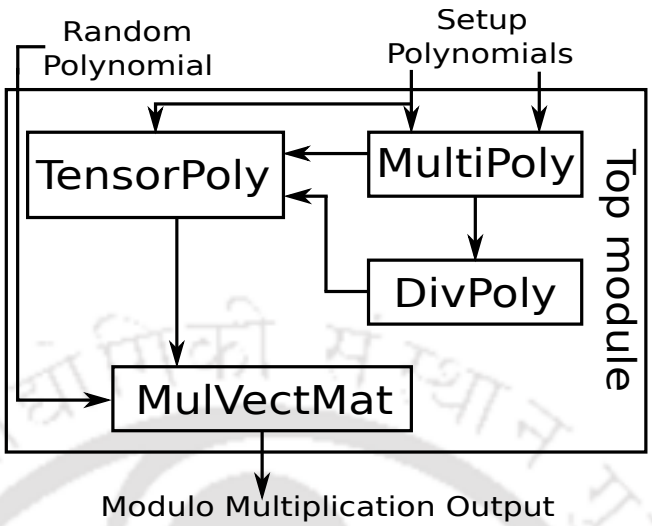


Figure 3.6: Block-level view basic process flow of the FSM

3.3 and 3.4 for the same polynomial inputs. It is to be noted that FPGA implementation outcome is ratified step by step with the theoretical analysis of our proposed modulo multiplier. This fulfills equivalence criterion of the correctness of FSM as stated in [181]. In the next section, experimental details of the proposed multiplier are presented.

3.5 Results and Discussion

In this section, results of FPGA implementation are presented. The proposed modulo multiplier is realized on Zynq-7000-Z020 and Artix-7-200T Xilinx FPGA boards. Each slice of the FPGA contains two types of LUTs; 6-input and 2-output, and 4-input and 2-output LUTs. Every slice contains four such types of LUTs. The Artix and Zynq architecture also has eight flip-flops, one MUX and one look-a-head carry logic cell per slice. Since most of the logic is implemented using pass-transistor logic, total estimated transistor count is around 222 per slice. The proposed multiplier implemented on FPGA is validated with the vast range of inputs. Number of variables in inputs are varied from two to nine, and the maximum degree for each variable ranges from 16 to 128. It is to mention that the experiments are performed over field size ranging from 32 to 1152. The resource, power, area, delay information is presented in Table 3.9 and comparison of the proposed multiplier with the recent multipliers are depicted in Table 3.10 and Table 3.11. From the basic structural diagram for multivariate polynomial multiplication shown in Figure 3.2, the total functional blocks required for the operations can be calculated using equation 3.7. The main polynomial multiplier block, a

vector-matrix multiplier block, and vector addition block are denoted by $Mult_P$, $Mult_V$ and Add_V, respectively. If total number of variables is V and the maximum degree of a variable is D, then the arithmetic functional block utilization can be formulated as equation 3.7.

$$Mult_P \times D + Mult_V \times D \times V + Add_V \times (D-1) \tag{3.7}$$

The Table 3.9 presents detailed physical aspect of the proposed multiplier. In this table, overall transistor count, area, delay, power, *power-delay product* (PDP), *power-delay-area product* (PDAP) and *area-delay product* (ADP) are stated. Transistor count can easily be estimated using total slice count and other resource utilization indicators. For the total area estimation of the proposed multiplier, first, we consider area of a single 22-nm transistor based on Intel Standard library [182] and then multiply it by the total number of transistors. This provides a close estimation of actual ASIC implementation. Power and delay are estimated by employing synthesis reports of the proposed multiplier. It is to mention that delays shown in Table 3.9 are cumulative delay of logic path and net delays only. Due to hardware resource constraints, HDL analyzer may generate different delays for different underlying hardware architectures because of distinctive performance of its synthesis, placement and routing algorithms.

For an FPGA implementation, block level logic path delays can be estimated using $T_{MP} = 0.9ns$, $T_{MS} = 0.1ns$ and $T_{AS} = 0.056ns$, which is obtained using synthesis report of FPGA implementation. Using these delays, generic delays can also be computed. Using Figure 3.2, the total delay can be formulated as mentioned in equation 3.8, where T_{MP}, T_{MS} and T_{AS} are the total delay of individual polynomial multiplier, delay of individual vector-matrix multiplier and delay of vector addition unit, respectively. It can be observed that FPGA based implementation matches with the analytical observations.

$$T_{MP} + (V-1) \times T_{MS} + [log_2^D] \times T_{AS} \tag{3.8}$$

In the Table 3.10, comparison with recent multipliers is presented. It is found that our proposed multiplier is faster as well as power efficient and utilizes lesser resources as compared to other multipliers. The experimental results presented in [149] is for 4-bit power array of two variables. They have implemented single variable NIST polynomials to showcase performance of their proposed multiplier. In [156], the vector-matrix multiplier is implemented specifically for DSP and communication

Table 3.9: Resource Utilization For Various Inputs

Variables	Degree	Slice Count	Transistor Count	Area (μm^2)	Delay (ns)	Power (mW)	PDP ($10^{-12}J$)	PDAP	ADP($\mu m^2.ns$)
2	16	781	173382	3120.9	1.84	205.96	378.97	1182691	5742.42
	32	1565	347430	6253.8	2.736	291.25	796.84	4983197	17110.24
	64	3133	695526	12519.5	4.528	461.81	2091.07	26179027	56688.16
	128	6269	1391718	25051.0	8.112	802.95	6513.47	163168293	203213.10
3	16	793	176046	3168.9	1.94	207.27	402.09	1274152	6922.76
	32	1789	397158	7148.9	2.836	315.61	895.07	6398664	20274.13
	64	3581	794982	14309.7	4.628	510.55	2362.79	33810682	66225.19
	128	7165	1590630	28631.4	8.212	900.41	7394.16	211704601	235120.57
4	16	1005	223110	4016.0	2.04	230.33	469.87	1886952	8192.60
	32	2013	446886	8044.0	2.936	339.98	998.17	8029180	23617.04
	64	4029	894438	16099.9	4.728	559.28	2644.26	42572125	76120.26
	128	8061	1789542	32211.8	8.312	997.88	8294.35	267175315	267744.12
5	16	1117	247974	4463.6	2.14	242.51	518.97	2316420	9551.96
	32	2237	496614	8939.1	3.036	364.35	1106.14	9887833	27138.97
	64	4477	993894	17890.1	4.828	608.01	2935.47	52515702	86373.37
	128	8957	1988454	35792.2	8.412	1095.35	9214.03	329789817	301083.76
6	16	1229	272838	4911.1	2.24	254.70	570.51	2801808	11000.83
	32	2461	546342	9834.2	3.136	388.71	1218.99	11987709	30839.92
	64	4925	1093350	19680.3	4.928	656.75	3236.43	63693759	96984.52
	128	9853	2187366	39372.6	8.512	1192.81	10153.20	399757489	335139.47
7	16	1341	297702	5358.7	2.34	266.88	624.49	3346389	12539.21
	32	2685	596070	10729.3	3.236	413.08	1336.71	14341893	34719.89
	64	5373	1192806	21470.6	5.028	705.48	3547.13	76158641	107953.72
	128	10749	2386278	42953.1	8.612	1290.28	11111.86	477287716	369911.28
8	16	1453	322566	5806.2	2.44	279.06	680.90	3953433	14167.10
	32	2909	645798	11624.4	3.336	437.45	1459.31	16963473	38778.88
	64	5821	1292262	23260.8	5.128	754.21	3867.59	89962694	119280.96
	128	11645	2585190	46533.5	8.712	1387.75	12090.02	562589881	405399.16
9	16	1565	347430	6253.8	2.54	291.25	739.76	4626213	15884.50
	32	3133	695526	12519.5	3.436	461.81	1586.78	19865534	43016.90
	64	6269	1391718	25051.0	5.228	802.95	4197.78	105158264	130966.24
	128	12541	2784102	50113.9	8.812	1485.21	13087.68	655873366	441603.13

2, 3, 4 & 5 Variables Maximum cumulative degree is $(2 \times 128) = 256$, $(3 \times 128) = 384$, $(4 \times 128) = 512$, $(5 \times 128) = 640$, $(6 \times 128) = 768$, $(7 \times 128) = 896$, $(8 \times 128) = 1024$, $(9 \times 128) = 1152$, PDP = power-delay-product, PDAP = power-delay-area-product, ADP = area-delay-product.

Table 3.10: Comparison of Existing Multipliers with the Proposed Multiplier

Work		Platform	Resource LUT/REG/MUX/DSP/BRAM	Comparative Utilization	Clock (MHz)	Power (W)	Delay (nS)	Latency (μs/CC)
Proposed*	1-PCM	Zynq-XC7Z020	8993/6479/6/-/6	**1.345X**	100	0.205	8.734	5.22/522
		Artix-7-200T	8982/2497/7/-/25	**X**	197	0.492	4.52	2.65/522
	4-PCM	Zynq-XC7Z020	9099/6544/6/-/6	**1.360X**	100	0.216	8.734	1.38/138
		Artix-7-200T	9088/2562/7/-/25	**1.015X**	194	0.511	4.61	0.72/138
	16-PCM	Zynq-XC7Z020	9659/6496/6/-/6	**1.405X**	98	0.227	8.82	0.43/42
		Artix-7-200T	9648/2514/7/-/25	**1.059X**	192	0.518	4.65	0.22/42
Toeplitz [157]**		Virtex-5	14701/∼/∼/∼/∼	**1.277X**	301.41	∼	23.065	13.791/∼
Alshawi et al.-I [156]*		Virtex-5	16207/12760/∼/∼/∼	**2.516X**	81.71	∼	12.24	∼/∼
Alshawi et al.-II [156]*		Virtex-5	8594/4700/∼/∼/∼	**1.155X**	65.47	∼	15.27	∼/∼
Karatsuba [149]**		Stratix II	∼/∼/∼/∼/∼	∼	86.80	3.216	41.52	∼/∼
LBPM [158]*		Artix -7	10768/21536/-/-/183	**2.822X**	143	∼	4000	27/∼
Yaman et al. [153]***	16 BTFs	Spartan-6	9898/3688/-/16/70	**1.188X**	115	∼	∼	∼/256
		Artix-7-200T	9508/2684/-/16/35	**1.064X**	172	∼	∼	

Polynomial coefficients in the form of * $x^{256} + \cdots + 1$ and, ** denotes pentanomial and trinomial polynomials $x^{163} + x^7 + x^6 + x^3 + 1$ and $x^{233} + x^{74} + 1$, ***$x^{256} + 1$, respectively. "∼" and "–" denote "Data not specified" and "Resource not utilized", respectively.

applications, whereas, in [157] Toeplitz matrix-vector product-based method is proposed. In [157], a single digit is represented by 2-bits to perform polynomial multiplication in every clock cycle. In [158] a flexible architecture for large binary polynomial multiplier is demonstrated by combining iterative Karatsuba and Comba algorithms. Lastly, in [153] number theoretic transform (NTT) and inverse-NTT (INTT) based multiplication techniques are presented, which exploits the concept of n^{th} root of unity to find multiplication and only optimized for NIST finalist CRYSTALS-KYBER PQC scheme. Considering all these factors, the experimental results of our proposed multiplier are better as compared with the above mentioned methods. It is to mention that multiplier proposed to design state-of-the-art Learning-with-Errors (LWE) and Ring-LWE (RWE) cryptographic homomorphic encryption algorithms efficiently.

The Figure 3.4 is the depiction of vector-matrix operation in the proposed multiplication method. As stated above that the architecture of this vector-matrix-vector operations is realized in the hardware and it is validated with various polynomial inputs. Later, its outcome is compared with other state-of-the-art methods. We realized r-PCM architecture and compare with [153] for their performance analysis. This architecture enables multiplication of r columns of T_x in parallel, where $r \leq n$. Thus, the multiplication can be performed in $((2 \times \frac{n}{r}) + 2)$ steps. Since all the coefficients of A, B and a tensor matrix T_x is known to us and are stored in BRAM, it can be utilized to perform r-PCM as stated above. In Figure 3.4, two PCMs are marked in green for completeness. Therefore, increase in the number of PCM operations reduces computational steps drastically with a minimal increase in the resource utilization. Thus, the computational complexity employing r-PCM is $log_2 n + (2 \times \frac{n}{r}) + 2$. Along with a sequential implementation, two parallel implementations, 4-PCM and 16-PCM, are also realized to compare the performance of the proposed modulo multiplier with [153] and other methods presented in Table 3.10. Using Table 1 in [153], it can be observed that the overall latency of the multiplier proposed in [153] are 3359, 864 and 256 clock cycles using 1-BTFs, 4-BTFs and 16-BTFs, respectively, whereas the proposed multiplier takes 522, 138 and 42 clock cycles only while employing 1-PCM, 4-PCM and 16-PCM, respectively. It can be seen that these implementations are 6.43×, 6.26× and 6.09× faster than the hardware realizations proposed in [153] using 1-BTFs, 4-BTFs and 16-BTFs, respectively.

For an input irreducible polynomial mentioned in Table 3.10, tensor matrix generation consumes average {8649-LUT, 6367-REG, 6-DSP, 6-BRAM} and {8638-LUT, 2412-REG, 7-DSP, 25-BRAM}

resources in Zynq-XC7Z020 and Artix-7-200T FPGAs, respectively, which is approximately 94% of the total resource utilization. A detailed description of vector-matrix-vector multiplication of our proposed multivariate polynomial multiplier is given below for the ready reference of the reviewer. It is to mention that this module consumes very less resources as compared to tensor generation module. Its parallel implementation, i.e. r-PCM blocks, increases overall resource utilization a little, but it reduces computation steps by a factor r. It can be observed that the resource utilization increases by 0.059× for r-PCM in our proposed multiplier, whereas resource utilization escalates by 9.39× by r-BTF in [153], when r changes from 1 to 16. It can be stated that the performance of our proposed multiplier increases with an increase in r at the minimal expense of resources. It is to mention that the work presented in [153] and our proposed work report latencies (computational steps) excluding the steps taken in loading data into BRAM.

Table 3.11: Comparison of Delay and Area and Cycle Requirement of Proposed Multiplier With Existing Multipliers

Multiplier	m	Delay (ns)	Area (μm^2)	Cycle (no.)
Proposed	163	3.046	9029	334
	233	3.266	10998	474
	283	3.426	12430	574
	409	3.816	15921	826
	571	4.326	20486	1150
	1223	6.366	38747	2454
Chen at el. [147]	163	439	13544	4458
	233	4032	19448	6108
	283	4576	19692	7596
	409	7392	20670	12908
	571	21216	127450	22100
	1223	50656	270214	91762
Hua at el. [148, 155]	163	90	4481	1203
	233	161	4715	2337
	283	204	4883	3331
	409	425	5442	6923
	571	816	6186	13163
	1223	3832	9195	60181
Chiou at el. [159]	163	5	61637	32
	233	7	78893	45
	283	7	100112	53
	409	11	133779	77
	571	14	202323	105
	1223	34	262175	227

Polynomial expression applied is $x^m + x^{m-1} \cdots + 1$

As shown in Table 3.10, total LUT logic slices, total flip-flops, MUX and BRAM utilized during implementation of our proposed multiplier on Zynq-7000-Z020 board are 8993, 6479, 6 and 6, respectively, at 100 MHz. Similarly, total LUT logic slices, total flip-flops, MUX and BRAM utilized realizing the same design on Artix-7-200T FPGA board are 8982, 2497, 7 and 25, respectively, at 197 MHz. A comparative resource utilization of all the multipliers is presented in Table 3.10. It can be observed that the resource utilization of our proposed multiplier is optimal and its average critical path is also reduced resulting into higher operating frequency. Its total power consumption is estimated to be approximately $0.2W$ and $0.492W$ at 100 MHz and 197 MHz, respectively, which are minimum $6.5\times$ less as compared to the existing multipliers. For further reduction of power in ASIC, input clock frequency and supply voltage can be reduced.

The static power consumption is higher in our proposed implementation because of storing initial polynomial coefficients and tensors. It is also found that on Artix-7 series FPGA board with XC7A200T, the latencies of our multiplier and the multiplier based on [153] are $2.65\mu s$ and $2.31\mu s$ at the clock frequencies of 197 MHz and 172MHz, respectively, which are quite close. It can be seen in Table 3.10, the multiplier proposed in [153] employs DSP blocks, while our proposed design does not use it. Also, multiplier proposed in [153] utilizes FFT type of operations, which can be performed easily using DSP blocks [183]. This helps in reducing overall latency of the design. Thus, there is a marginal difference in the latencies of both the multipliers. It can also be observed in Table 3.10 that the power consumption of the proposed multiplier is the least among all the other multipliers in the worst case scenario. Thus, our proposed multiplier can be considered area and power optimal with respect to other multipliers, and can be utilized in low power applications including lattice based PQC algorithms, drone based surveillance systems etc.

In Table 3.11, three more well-known multiplication schemes are compared over parameters, such as, area delay and latency in terms of clock cycles with varying polynomial degree. Multiplier based on systolic dual basis is depicted in [147]. In [148] and [155], low complexity systolic multipliers based on Hankel matrix and Karatsuba algorithm are presented, respectively. In [159], multiplication unit is decomposed in four mutually independent sub-multiplication units, which improves the performance as compared to the prior methods. In Table 3.11, a comparison of specific polynomial multipliers having polynomial degrees $m = \{163, 233, 283, 409, 571, 1223\}$, is presented. It is observed that the reduction of area-delay product (ADP) with respect to Chiou's method [159] are 91%, 93.5%, 93.9%,

95.8%, 96.8%, 95.5% for different m, while average reduction in ADP is 94.42%. It can also be seen that the delay improvement for different m are 39.1%, 53.3%, 51.1%, 65.3%, 69.1%, 81.3% and an average delay improvement is 59.87%.

The subsequent section demonstrates an application of our proposed polynomial modulo multiplier. A polynomial based lattice homomorphic encrypting scheme utilizing the proposed tensor method to calculate modulo operations is realized. The detailed formulation, implementation with security parameter analysis is elaborated in the next section.

3.6 Application of Proposed Modulo Multipliers over Lattice Homomorphic Encryption Scheme

Lattice based PQC is one of the alternatives to build a strong resistance against various malicious intrusions. Lattices are discrete subgroups of $\mathbb{R}^m$. A lattice L is a set of linearly independent basis vectors $(b_1, \ldots, b_n)$ in $\mathbb{R}^m$, such that, $L(b_1, \ldots, b_n) = \sum_{i=1}^{n} x_i b_i, x_i \in \mathbb{Z}$, $\forall$ integer linear combinations of b_i. Here, n is the dimension of L. If ϕ represents function to be evaluated over plaintexts m_1 and m_2, then homomorphic encryption scheme computes $\phi(m_1, m_2)$ using encryption $Enc(m_1)$ and $Enc(m_2)$ without the knowledge of m_1 and m_2.

The modulo polynomial operations are employed to formulate lattice homomorphic encryption scheme [184] and are summarized in the following steps.

- First, a message $m \in \mathbb{F}$, is mapped to $p(m)$, i.e, a polynomial in the polynomial ring $\mathbb{F}_q[x_1, ..., x_n]$.
- Second, the coefficient vector of the polynomial obtained in the above step is mapped to a ciphertext matrix C, where $\mathbb{F}$ is a field and $\mathbb{F}_q$ is a field of cardinality q. It is to mention that q is a prime number or integer power.

Considering $\mathbb{F}_q$ as a plaintext space, $n \in \mathbb{N}$ is the number of variables and $c = f(\lambda)$, where λ is the security parameter. $m \in \mathbb{F}_q$ is mapped to $p(m)$ as depicted in equation 3.9.

$$p(m) = m + \sum_{i=1}^{n} r_i g_i \tag{3.9}$$

where, $r_i \in \mathbb{F}_q[x_1, \ldots, x_n]$ and g_i is chosen randomly from $\mathbb{F}_q[x_1, \ldots, x_n]$. The coefficients of $p(m)$ are mapped to matrix C as described below.

- A random matrix $\tilde{C}$ of dimension $N \times N$ is chosen, where $N = (d+1)^n$ and d is the size of vector

space. One column is chosen randomly at a time, using a column shift operator δ. Thereafter, δ^{th} column of the matrix $\tilde{C}$ is replaced with a coefficient vector.

- Later, two random invertible matrices, M and R, are chosen to construct the final ciphertext $C, C = M.\tilde{C}.R$

The homomorphic scheme employing same value of δ is used to evaluate ciphertexts. While encrypting zeros with this δ, the resulting ciphertext can span an entire subspace of $\mathbb{F}_q^{N\times N}$. The upper limit of ciphertexts during homomorphic encryption is equal to the dimension of this subspace. For homomorphic multiplication, the product of ciphertexts is computed using a bilinear mapping. Public evaluation key comprises of this map along with the linear maps M and R. The product of two ciphertexts, C_1 and C_2, is determined by using a bilinear map B and is represented by equation 3.10.

$$B[p(m_1), p(m_2)] = p(m_1).p(m_2) \quad mod \quad f_{pk} \tag{3.10}$$

To calculate f_{pk}, a chain of rings is constructed. $R_0 \subseteq R_1 \subseteq ....R_{n-1}$ and the i^{th} term can be expressed as equation 3.11.

$$R_i = R_{i-1}[x_i]/\langle f_i \rangle, \quad 1 < i < n-1 \tag{3.11}$$

The polynomials, $p_i(m)$, in all the stages are the members of a ring. Considering all symbols with their usual meaning, the product of two ciphertexts, C_1 and C_2, based on the proposed tensor matrix method can be expressed as equations 3.12 and 3.13, where c_i^1 and c_i^1 are the columns vectors of C_1 and C_2, respectively.

$$T(C_1, C_2) = [T_1(C_1, C_2), T_2(C_1, C_2) \ldots T_N(C_1, C_2)] \tag{3.12}$$

$$T_i(C_1, C_2) = [{c_1^1}^T \ldots {c_N^1}^T] T_i \begin{bmatrix} c_1^2 \\ . \\ . \\ . \\ c_N^2 \end{bmatrix} \tag{3.13}$$

The security of such encryption schemes depends on an arithmetic field, in which ciphertexts are mapped and perform operations. If the number of variables increases along with their degree of polynomials, i.e. field, it enhances the security. The relationship between field and security is illustrated in Table 3.12. The relationship of security parameters is expressed as equation 3.14 [3].

Table 3.12: Security Parameters Comparison [3] for Different Field Sizes (for $b = 2$; $k = \sqrt{2log(100)}$)

λ	μ	n	N	α	q	$\alpha q > 2\sqrt{N}$
32	1	16	140	0.0041666666667	3277	False
32	2	22	256	0.0001302083333	104858	False
32	3	28	344	4.069010416667E-06	3355444	False
64	1	18	160	0.0017361111111	26215	True
64	2	22	519	2.712673611111E-05	1677722	False
64	3	26	575	4.238552517361E-07	107374183	False
128	1	24	310	0.0007440476191	209716	True
128	2	26	350	5.81287202381E-06	26843546	True
128	3	28	378	4.541306268602E-08	3435973837	True
256	1	39	790	0.0003255208333	1677722	True
256	2	42	915	1.271565755208E-06	429496730	True
256	3	45	1003	4.967053731283E-09	109951162778	True
512	1	66	2225	0.000144675926	13421773	True
512	2	68	2460	2.825701678241E-07	6871947674	True
512	3	70	2598	5.518948590314E-10	3518437208884	True

$$q \approx \lambda^{b+\mu}$$
$$\alpha = 1/(\lambda^{\mu} log_b^{\lambda\sqrt{\lambda}}) \tag{3.14}$$

Description of various parameters in Table 3.12 is as follows. λ is the parameter to generate polynomials over a finite field of prime size $q(\lambda)$, where q is a prime number. n is the number of variables and α is the parameter to generate discrete Gaussian distribution ($\alpha > 0$). μ states levels of multiplication, while N is the number of elements in the polynomial ring. *False* in the last column states that the scheme based on the parameters in a particular row is not strong enough against quantum computer based attacks, whereas, *True* shows a good measure of resistance against such attacks.

All the polynomial multiplications mentioned above are modulo operations and integral part of lattice cryptosystems. Therefore, polynomial modulo multipliers proposed in this chapter can be utilized to implement a secure and efficient lattice homomorphic encryption scheme.

3.7 Conclusion

In this chapter, a novel design methodology of multivariate polynomial modulo multiplication based on tensors is presented. The proposed multiplier outperforms other multipliers in terms of resource utilization and power consumption. An embarrassingly parallel and scalable architecture of the propsoed multiplier is described, which performs linearly for single variable polynomial multiplication and quadratically for multivariate polynomial multiplication in the worst case. However, its average and best case computational complexity is linear for both the single variable as well as multivariate polynomials. For single variable polynomial multiplication, the proposed multiplier utilizes minimum 6% less resources, consumes 6.5× less power and more than 6× faster as compared to other polynomial modulo multipliers. As per our knowledge, a multivariate polynomial modulo multiplier is realized first time on the hardware platform and its performance is analyzed with the multivariate polynomial of 9 variables, where the maximum degree of each variable is 128. For this polynomial, chip area, power and delay estimated by our proposed multiplier are $50113.9\mu m^2$, 1.485 W and 8.812 ns, respectively. It is found that the performance of our multivariate multiplier is linear with respect to above mentioned parameters. This multiplier is successfully validated using a LWE based lattice multivariate encryption scheme to measure its efficacy. This indicates that our proposed multiplier may be an ideal choice for an efficient implementation of homomorphic encryption algorithms on various low power devices.

4

Chaos Triple Pendulum Based Cryptosystem

Contents

Objective

Recent advancements in the domain of quantum computing are posing a security threat to the classical cryptography algorithms. Popular symmetric and asymmetric cryptosystems including Rivest-Shamir-Adleman *RSA*, Elliptic Curve Cryptography *ECC*, Data Encryption Standard *DES*, Diffie-Hellman etc. can be broken by a quantum computer executing Shor's and Grover's algorithms. This motivated scientific community to design newer encryption schemes to address security vulnerabilities. Hash, Code, Lattice, Multivariate Polynomial based cryptography algorithms, known as post-quantum cryptography algorithms (PQC), exhibit resistance against classical as well as quantum crypto-attacks. Apart from these PQC algorithms, a relatively new method of constructing cryptosystems utilizing the unpredictability property of discrete chaotic dynamic systems has become noteworthy from the practical perspective. A encryption scheme based on the chaotic dynamic physical system derived from a mechanical model depicting nonlinear dynamics is proposed, which exhibits resistance against various attacks. The effectiveness of the proposed chaotic map is validated against various standard tests, such as Lyapunov exponents test, bifurcation diagrams, sensitivity to parametric and to initial values, ergodicity, collision test, NIST randomness test etc. The triple pendulum based chaotic generator is utilized to develop a symmetric key encryption scheme and it is realized on FPGA platform. The proposed cryptosystem is verified through an FPGA implementation to assess its usage in low power high throughput applications.

4.1 Introduction to Chaos Based Cryptography

The enhancement of computing power and the rise of quantum computing have made many widely adopted conventional encryption schemes vulnerable. Due to this scientific community is compel to find a suitable solution to address this issue. Chaotic dynamic system based algorithms, can be categorized as post-quantum cryptography algorithms (PQC) algorithms and also be a viable solution to quantum computing generated attacks as there are no known successful attacks against them [110]. The main advantages of deterministic chaotic systems are higher degree of unpredictability, ergodicity, high sensitivity to initial conditions and control parameters, close resemblance with true randomness and unstable long aperiodic orbits. Chaotic properties can be employed in synchronization and control, communication, cryptography, neural network, etc. [111–113]. These systems are distinguished by their high sensitivity to initial conditions, statistical similarity to random signals and a continuous

broadband power spectrum. All this has garnered interest from cryptanalysts to propose various chaos based cryptographic systems [114, 115] and can be subdivided into two classes. In the first class, a chaotic system is numerically iterated over a period employing a message as an initial data [116], whereas, in the second class, a message is scrambled with a chaotic dynamic function. The relevance and usefulness of chaos have been demonstrated through comparative studies employing characteristics of a chaotic system and the requirement of a strong cipher [115, 116].

Designing cryptographic schemes based on chaotic dynamics can be divided into two steps. In the first step, an encryption algorithm is modeled based on analog chaotic dynamic systems. Thereafter, discretization with finite-precision is introduced during hardware implementation. It is to mention that discretization is the main trade-off between security and resource utilization. As we know, the method of synchronization of multiple chaotic systems is applied in some of the earliest cryptography applications. The piece-wise linear chaotic map ($PLCM$) [79–81] based chaotic maps are simple and one-dimensional system and can be implemented easily, but they are prone to security vulnerabilities due to their smaller keyspace. Various modifications and variants of $1-$D PLCM, such as piece-wise nonlinear chaotic maps, algorithms based on logistic maps [82, 83], polynomial chaotic maps are reported in literature to improve their limitations. Now-a-days, many real world chaotic phenomenon are being modeled to generate ideal chaotic characteristics; therefore, physical chaotic dynamical systems have become an imperative area of research in chaos based cryptography [84] [85].

For strengthening the security and to increase the speed of chaotic cryptosystems, a combination of multiple one dimensional algorithms are used. For example, in [86], a combination of polynomial chaotic map and the piecewise nonlinear chaotic map is employed to enhance the speed and security of a cryptosystem. It is to mention that various chaotic block cipher encryption schemes are also developed. A chaotic tent map based single iterative encryption scheme is proposed in [87], which is further generalized in [88], and multiple iterative based dynamical chaotic system models are presented in [89]. A novel symmetric block cipher algorithm based on invertible two dimensional chaotic maps on a torus is presented in [90], whereas, schemes formulated by employing iterative logistic equation and synchronized chaotic systems are discussed in [91] and [92], respectively. In another approach, different chaotic attractors for a random chunk of binary plain-text data is presented [93] in order to implement chaotic based encryption scheme.

As per our knowledge, only a few chaotic map-based image and video encryption schemes have been

developed in the past. Some of these methods are mentioned as follows. In [84], a new hyperchaotic temperature fluctuation model based on the circuit analysis is proposed for an image encryption. Simple first order differential equations ($\dot{x} = z$; $\dot{y} = -z(ay + by^2 + xz)$; $\dot{z} = x^2 + y^2 - |x| - 1$) based a new 3D chaotic system employing a peanut-shaped symmetric equilibrium curve is reported in [94]. A nonlinear continuous-time dynamical chaotic system based on two circles of equilibrium points and its multistability is discussed in [95]. As mentioned above, all these chaotic systems are utilized for image and video encryption only and there is no general-purpose chaotic dynamic system based cryptosystem exist. With the advent of smart systems incorporating Internet-of-Things (IoT) devices, data (or information) security has become of utmost importance. Internet of Everything (IoE) is the future and it demands a system to have very high security, energy efficiency and reliability. Therefore, it has become imperative to design a general purpose low power cryptosystem, which can be used in various applications ranging from IoT based smart systems to high throughput low latency systems.

In this chapter, a novel and generic chaos-based low power cryptosystem is proposed to address above mentioned issues, which is based on a chaotic triple pendulum system. This novel algorithm is also implemented on FPGA as well as on ASIC and is compared with other state of the art methods reported in the literature. Details of our proposed algorithm is presented in the subsequent sections. The rest of the chapter is organized as follows. In the next section illustrates a step-wise approach and methodology to design a triple pendulum based cryptography algorithm. Then the formulations of chaotic double and triple pendulum dynamic systems are presented with case studies. Thereafter, Lyapunov exponents test and bifurcation diagrams to test chaotic behavior of the proposed triple pendulum based chaotic dynamic system is displayed. Later, encryption and decryption algorithms are also depicted in the same section. The details of hardware implementation and their the results obtained on FPGA and ASIC is also discussed to validate effectiveness of the proposed method. Then various tests, such as sensitivity to parametric and to initial values, ergodicity, collision test, NIST randomness test conducted to assure chaotic nature of the proposed method.

4.2 Step-wise Design Methodology

The proposed research in the current chapter focuses on implementing a new chaos-based encryption-decryption technique on Field Programmable Gate Array (FPGA) and as an Application Specific Integrated Circuit (ASIC). We aim to design a system in such a way that it should be less resource-

intense, computationally efficient and with a strong security. For achieving this goal, necessary steps to be performed along with design and implementation are given below.

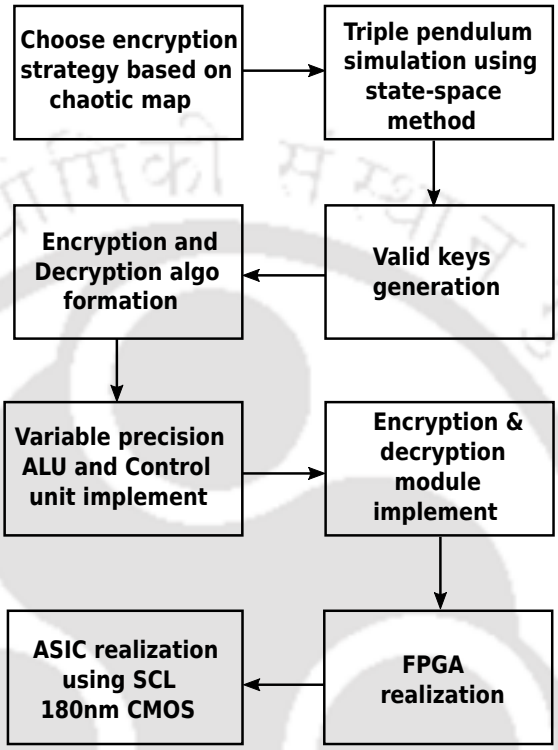


Figure 4.1: Design steps of the proposed method

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

Figure 4.1 depicts design and implementation steps of the proposed cryptosystem. The motivation behind employing the proposed algorithms for encryption and decryption is based on Baptista-type

encryption-decryption scheme. It is described using an interval-partitioning of chaotic orbits of a $1D$ logistic map, which enables construction of very fast and secure encryption-decryption schemes, because of its less complex structure. The main objective of our scheme is to first map text characters to numerical values and, then algorithms are applied to encrypt the message. Decryption is performed by iterating a chaotic map and, then corresponding symbols for numerical values are obtained by inverting the process. After referring literature, we have decided to employ a compound triple-pendulum model for creating desired chaotic map in our cryptosystem. Since, input to the system is a plain message, the system parameter(s) and additionally the initial conditions of a dynamical system are assumed to be the part of the secret key. After analyzing a triple-pendulum system, a valid set of keys for encryption and decryption is generated. The entire mechanical model is implemented on an FPGA along with a custom ALU and a control unit adhering to the specifications of our proposed system. After successful validation of the design on an FPGA, it is realized using Semi Conductor Laboratory (SCL) 180nm Bulk CMOS technology.

4.3 Mathematical Modeling of Multi Pendulum System

4.3.1 Formulation of Double Pendulum System

The formulation of a nonlinear triple pendulum chaotic system is similar to a double-pendulum system. For simplicity and to limit usage of the variables in mathematical formulation, the construction of differential equations of a double pendulum system is demonstrated below [185]. Figure 4.2 exhibits a basic double pendulum pictorially.

The basic position relations from the above figure are denoted as the equation 4.1 and 4.2.

$$x_1 = L_1 sin\theta_1;\ x_2 = L_1 sin\theta_1 + L_2 sin\theta_2; \tag{4.1}$$

$$y_1 = -L_1 cos\theta_1;\ y_2 = -L_1 cos\theta_1 - L_2 cos\theta_2; \tag{4.2}$$

The expressions of potential (P) and kinetic (K) energies, as mentioned in equation 4.3 and 4.4, are obtained by analyzing position relations of equations 4.1 and 4.2.

$$P = -m_1 g L_1 cos\theta_1 - m_2 g(L_1 cos\theta_1 + L_2 cos\theta_2) \tag{4.3}$$

$$K = \frac{1}{2} m_1 \dot{\theta_1}^2 L_1^2 + \frac{1}{2} m_2 (\dot{\theta_1}^2 L_1^2 + \dot{\theta_2}^2 L_2^2 + 2\dot{\theta_1} L_1 \dot{\theta_2} L_2 cos(\theta_1 - \theta_2)) \tag{4.4}$$

Later, Euler-Lagrange condition $\frac{d}{dt}(\frac{\partial L}{\partial \dot{\theta}}) - \frac{\partial L}{\partial \theta} = 0$ is applied to relation $L = K - P$ and the obtained

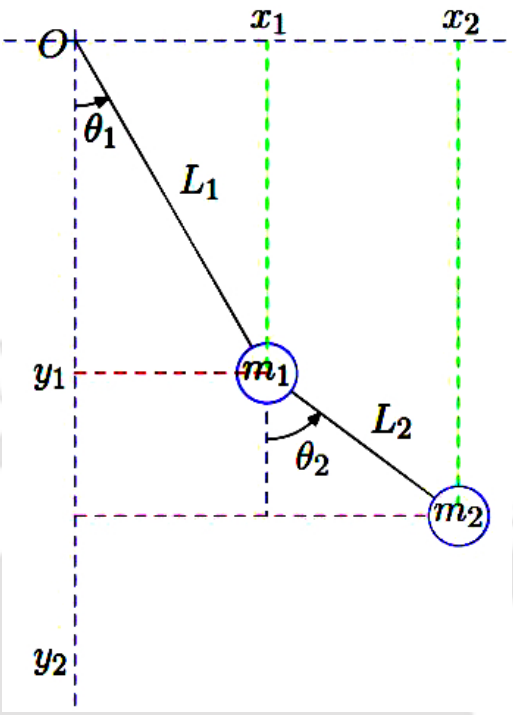


Figure 4.2: Basic double pendulum setup

expression is denoted by equation 4.5.

$$L = \frac{1}{2}(m_1+m_2)L_1^2\dot{\theta_1}^2+\frac{1}{2}m_2L_2^2\dot{\theta_2}^2+m_2L_1L_2\dot{\theta_1}\dot{\theta_2}cos(\theta_1+\theta_2)+(m_1+m_2)gL_1cos\theta_1+m_2L_2gcos\theta_2 \quad (4.5)$$

Thereafter, substituting L into above mentioned Euler-Lagrange equation and solving the resultant expression for θ_1 and θ_2 as second order differential equations are obtained which are given in equation 4.6 and 4.7.

$$\ddot{\theta_1} = \frac{\begin{array}{c}-m_2L_1\dot{\theta_1}^2sin(\theta_1-\theta_2)cos(\theta_1-\theta_2)+gm_2sin\theta_2cos(\theta_1-\theta_2)- \\ m_2L_2\dot{\theta_2}^2sin(\theta_1-\theta_2)-(m_1+m_2)gsin\theta_1\end{array}}{[L_1(m_1+m_2)-m_2L_1cos^2(\theta_1-\theta_2)} \quad (4.6)$$

$$\ddot{\theta_2} = \frac{\begin{array}{c}m_2L_2\dot{\theta_2}^2sin(\theta_1-\theta_2)cos(\theta_1-\theta_2)+gsin\theta_1cos(\theta_1-\theta_2)(m_1+m_2)+ \\ L_1\dot{\theta_1}^2sin(\theta_1-\theta_2)(m_1+m_2)-gsin\theta_2(m_1+m_2)\end{array}}{L_2(m_1+m_2)-m_2L_2cos^2(\theta_1-\theta_2)} \quad (4.7)$$

4.3.2 Formulation of Compound Triple Pendulum System

As stated above, the steps of mathematical formulation of a triple pendulum system is similar to the double pendulum system. In this section, an abstract representation of mathematical formulation

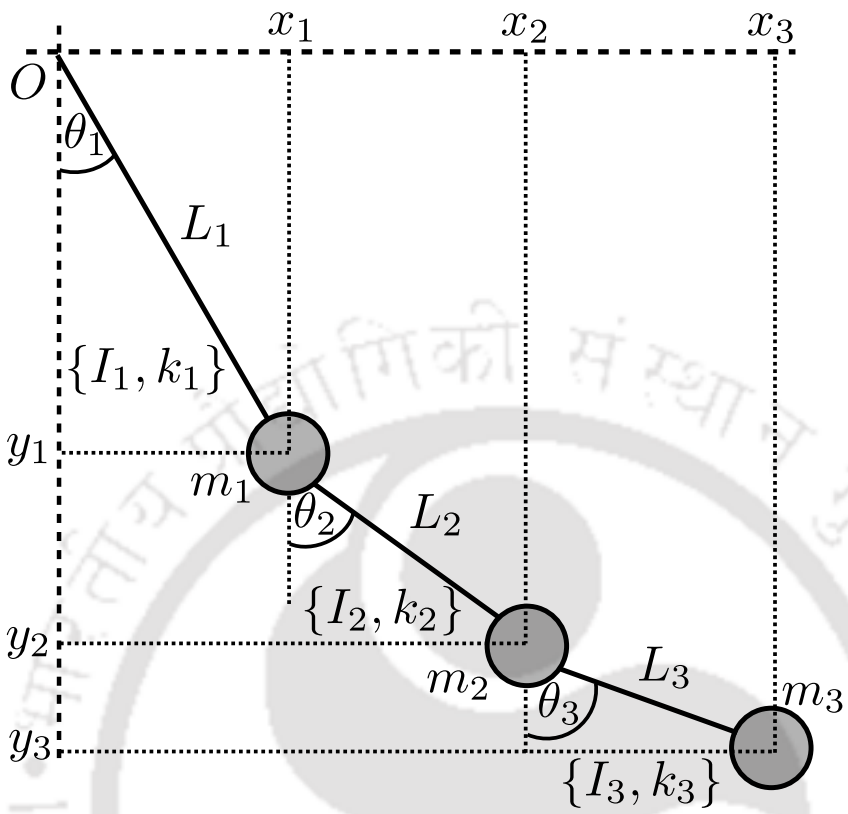


Figure 4.3: Basic triple pendulum setup

of triple pendulum system is presented [185]. The bars of a triple-pendulum have significant mass, so that it can be modeled as a compound pendulum. Damping factors are also included in the model to introduce higher degree of chaos and nonlinearity in the system. Each bar i is defined by a set of four parameters mentioned below and the figure 4.3 is the stick diagram representation of that where m_i is the mass of the bar, l_i is the length of the bar, I_i is the moment of inertia of the bar and k_i is the damping coefficient of the bar rotating about its upper joint.

The position and velocity of the bars are defined by six system state variables, θ_1, θ_2, θ_3, $\dot{\theta}_1$, $\dot{\theta}_2$, $\dot{\theta}_3$. The relevant governing equations to deduce these variable are given below.

$$y_1 = \frac{l_1}{2} cos\theta_1 \tag{4.8}$$

$$y_2 = l_1 cos\theta_1 + \frac{l_2}{2} cos\theta_2 \tag{4.9}$$

$$y_3 = l_1 cos\theta_1 + l_2 cos\theta_2 + \frac{l_3}{2} cos\theta_3 \tag{4.10}$$

$$x_1 = \frac{l_1}{2} sin\theta_1 \tag{4.11}$$

$$x_2 = l_1 sin\theta_1 + \frac{l_2}{2} sin\theta_2 \tag{4.12}$$

$$x_3 = l_1 sin\theta_1 + l_2 sin\theta_2 + \frac{l_3}{2} sin\theta_3 \tag{4.13}$$

The magnitude of the velocity of each bar can be expressed as

$$v_i = \sqrt{\dot{x_i}^2 + \dot{y_i}^2} \tag{4.14}$$

The translational and rotational kinetic energies, TKE_i and RKE_i, respectively, are depicted by

$$TKE_i = \frac{1}{2} m_i v_i^2 \tag{4.15}$$

$$RKE_i = \frac{1}{2} I_i \dot{\theta}_i \tag{4.16}$$

The gravitational potential energy, GPE_i, can be presented as

$$GPE_i = m_i g y_i \tag{4.17}$$

Lagrangian of the system is expressed below.

$$\mathcal{L} = T - V = \sum_{i=1}^{3} TKE_i + RKE_i - GPE_i \tag{4.18}$$

Lagrange's equation can be defined as

$$\frac{d}{dt}(\frac{d\mathcal{L}}{\dot{q}_i}) = \frac{d\mathcal{L}}{dq_i} - \frac{dD}{\dot{q}_i} \tag{4.19}$$

Rayleigh Dissipation Function, D, is described below.

$$D = \frac{1}{2}(\sum_{i=1}^{3} k_i \dot{\theta_i}^2) \tag{4.20}$$

$$\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)) \ldots \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_2^3 + 4I_3 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))\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 + 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)) \tag{4.21}$$

$$\ddot{\theta_2} = (2((l_1^2 sin(2\theta_1 - 2\theta_2)(m_2 + 2m_3)(m_2 m_3 l_3^2 + 4I_3(m_3 + 2m_3)) - l_3^2 m_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_3^2 m_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_1 l_1(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))l_2 \dot{\theta_1} + (-l_3 m_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_3 l_3^2 + 4I_3)) - l_1^2 sin(2\theta_1 - \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_2 - \theta_3)((m_1 + 3m_2 + 2m_3)l_1^2 + 4I_1) - l_1^2 cos(2\theta_1 - \theta_2 - \theta_3)(m_2 + 2m_3))\dot{\theta_3} + g(sin(\theta_2)((m_2 m_3(2m_2 + 3m_3)l_3^2 + 8I_3(m_2^2 + 3m_2 m_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_3^2 m_3^2(sin(\theta_2 - 2\theta_3)(m_2 l_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_3 l_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_1^2 l_3^2 m_3^2 cos(2\theta_1 - 2\theta_3))\dot{\theta_2}))/(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 + 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_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) - 3l_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 - 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)) \tag{4.22}$$

$$\ddot{\theta_3} = -(2(32I_1I_2k_3\dot{\theta_3} - l_2l_3m_3\dot{\theta_2}^2(sin)\theta_1 - \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_1^2sin(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_2^2sin(\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} + 16l_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_1k_3m_3\dot{\theta_1}(cos(\theta_1 - \theta_3)(2m_3l_2^2 + 4I_2) - l_2^2cos(\theta_2 - 2\theta_2 + \theta_3)(m_2 + 2m_3)) - 16I_1I_2gl_3m_3sin(\theta_3) - 4I_2l_1^2l_3^2m_3^2\dot{\theta_3}^2sin(2\theta_1 - 2\theta_3) - 4I_1l_2^2l_3^2m_3^2\dot{\theta_3}sin(2\theta_2 - 2\theta_3) + 8I_2gl_1^2l_3m_3^2sin(2\theta_1 - \theta_3) + 8I_1gl_2^2l_3m_3^2sin(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_3^2sin(\theta_3) - 8I_2gl_1^2l_3m_3^2sin(\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_3sin(\theta_3) - 16k_3l_1^2l_2^2m_2m_3\dot{\theta_3}cos(2\theta - 2\theta_2) - l_1^2l_2^2l_3^2m_1m_3^2\dot{\theta_3}^2sin(2\theta_2 - 2\theta_3) + l_1^2l_2^2l_3^2m_2m_3^2\dot{\theta_3}^2sin(2\theta_1 - 2\theta_3) - 2l_1^2l_2^2l_3^2m_2m_3^2\dot{\theta_3}^2sin(2\theta_2 2\theta_3) + 2gl_1^2l_2^2l_3m_1m_3^2sin(2\theta_2 - \theta_3) - gl_1^2l_2^2l_3m_2^2m_3sin(2\theta_1 - \theta_3) + 2gl_1^2l_2^2l_3m_2m_3^2sin(2\theta_2 - \theta_3) + gl_1^2l_2^2l_3m_2m_3sin(2\theta_2 - \theta_3) + 4I_2gl_1^2l_3m_1m_3sin(2\theta_1 - \theta_3) + 8I_2gl_1^2l_3m_2m_3sin(2\theta_1 - \theta_3) + 4I_1gl_2^2l_3m_2m_3sin(2\theta_2 - \theta_3) - 2gl_1^2l_2^2l_3m_1m_3^2sin(2\theta_1 - 2\theta_2 + \theta_3) - 2gl_1^2l_2^2l_3m_2m_3^2sin(2\theta_1 - 2\theta_2 + \theta_3) - gl_1^2l_2^2l_3m_2^2m_3sin(2\theta_1 - 2\theta_2 + \theta_3) + gl_1^2l_2^2l_3m_2^2m_3sin(\theta_3) - gl_1^2l_2^2l_3m_1m_2m_3sin(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)) \tag{4.23}$$

$$
\begin{aligned}
\ddot{\theta_1}^{(t+1)} &= f_1(\theta_1^{(t)}, \theta_2^{(t)}, \theta_3^{(t)}, \dot{\theta_1}^{(t)}, \dot{\theta_2}^{(t)}, \dot{\theta_3}^{(t)}) \\
\ddot{\theta_2}^{(t+1)} &= f_2(\theta_1^{(t)}, \theta_2^{(t)}, \theta_3^{(t)}, \dot{\theta_1}^{(t)}, \dot{\theta_2}^{(t)}, \dot{\theta_3}^{(t)}) \\
\ddot{\theta_3}^{(t+1)} &= f_3(\theta_1^{(t)}, \theta_2^{(t)}, \theta_3^{(t)}, \dot{\theta_1}^{(t)}, \dot{\theta_2}^{(t)}, \dot{\theta_3}^{(t)}) \\
\dot{\theta_1}^{(t+1)} &\leftarrow \dot{\theta_1}^{(t)} + \ddot{\theta_1}^{(t+1)} \Delta t \\
\dot{\theta_2}^{(t+1)} &\leftarrow \dot{\theta_2}^{(t)} + \ddot{\theta_2}^{(t+1)} \Delta t \\
\dot{\theta_3}^{(t+1)} &\leftarrow \dot{\theta_3}^{(t)} + \ddot{\theta_3}^{(t+1)} \Delta t \\
\theta_1^{(t+1)} &\leftarrow \theta_1^{(t)} + \dot{\theta_1}^{(t+1)} \Delta t \\
\theta_1^{(t+1)} &\leftarrow \theta_1^{(t)} + \dot{\theta_1}^{(t+1)} \Delta t \\
\theta_1^{(t+1)} &\leftarrow \theta_1^{(t)} + \dot{\theta_1}^{(t+1)} \Delta t
\end{aligned}
\tag{4.24}
$$

Table 4.1: Key parameters value and initial condition

Parameter/ Condition	Value	Unit
m_1, m_2, m_2	$0.2944, 0.1756, 0.0947$	kg
l_1, l_2, l_3	$0.508, 0.254, 0.127$	m
I_1, I_2, I_3	$(9.526, 1.625, 1.848) \times 10^{-3}$,	kg.m^2
k_1, k_2, k_3	$(5, 0, 0.8) \times 10^{-3}$	N.m.s/rad
$\theta_1, \theta_2, \theta_3$	$-0.4603, -1.2051, -1.5165$	rad
$\dot{\theta_n}$	0	rad/s

Based on the above mentioned equations, a compound triple-pendulum model is numerically analyzed using MATLAB [186] employing ordinary differential equations (ODEs) used to describe random motions. The parameters and initial conditions of these ODEs are given in Table 4.1. For the analysis, numerical methods are utilized to solve a state-space model of differential equations, which are given in equation 4.21, 4.22 and 4.23. The numerical values corresponding to angular positions of the bars are obtained within a certain duration as per the predefined precision. This generates a chaotic map for an encryption module. The equations of $\ddot{\theta_1}$, $\ddot{\theta_2}$ and $\ddot{\theta_3}$ stated in equation 4.21, 4.22 and 4.23 can be simplified as follows. $\ddot{\theta_1}, \ddot{\theta_2}$ and $\ddot{\theta_3}$ are evaluated using differential equations by iterating from $t = 0$ to $t = N$, where $N = \frac{T}{\Delta t}$.

Later, θ are stored in a vector y denoted as equation 4.25.

$$\mathbf{y}^i = \{\theta_i^{(0)}, \theta_i^{(1)}, \theta_i^{(2)}, \ldots, \theta_i^{(N-1)}\}, \text{ for } i = 1, 2, 3 \text{ and } N = \frac{T}{\Delta t} \tag{4.25}$$

Subsequently, y is phase wrapped in the range $[0, 2\pi]$ (or $[-\pi, \pi]$) to achieve following optimized expression equation 4.26.

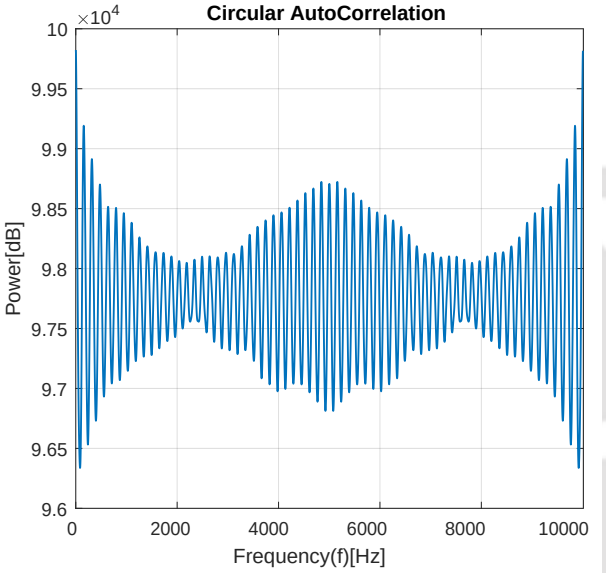
$$\hat{\mathbf{y}}^i = \{\hat{\theta}_i^{(0)}, \hat{\theta}_i^{(1)}, \hat{\theta}_i^{(2)}, \ldots, \hat{\theta}_i^{(N-1)}\} \text{ for } i = 1, 2, 3 \text{ where, } \hat{\theta}_i^{(t)} = \theta_i^{(t)} - \lfloor \frac{\theta_i^{(t)}}{2\pi} \rfloor \times 2\pi \tag{4.26}$$

It is to mention that state variables $\ddot{\theta_1}, \ddot{\theta_2}$ and $\ddot{\theta_3}$ are used to generate a one-to-one function F denoted by equation 4.27.

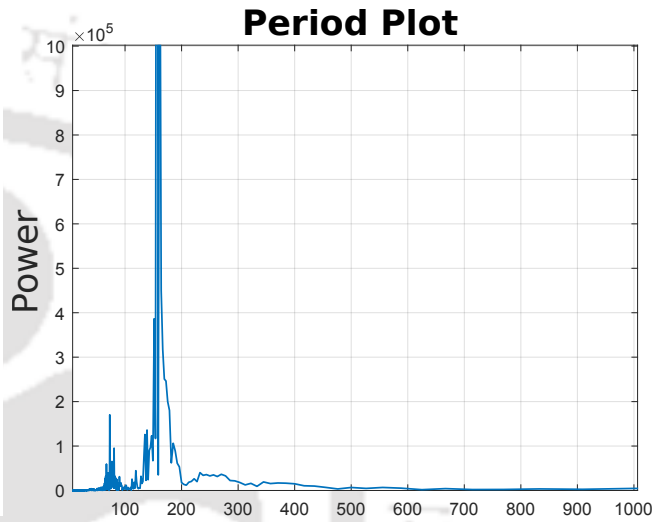
$$\hat{\mathbf{y}} = F(\hat{\mathbf{y}}^1, \hat{\mathbf{y}}^2, \hat{\mathbf{y}}^3) \tag{4.27}$$

The Figure 4.4 presents a circular auto-correlation, estimated power of θ_3 with respect to its periodicity for periodic and nonperiodic trajectories. Periodic nature of circular auto-correlation signifies overall periodic nature of the trajectories and vice-versa. It can be observed from Figure 4.4(a) and Figure 4.4(b) that the power is concentrated around a few frequencies for periodic signals, whereas, power is distributed in the whole domain for aperiodic signals as shown in Figure 4.4(c) and Figure 4.4(d), respectively.

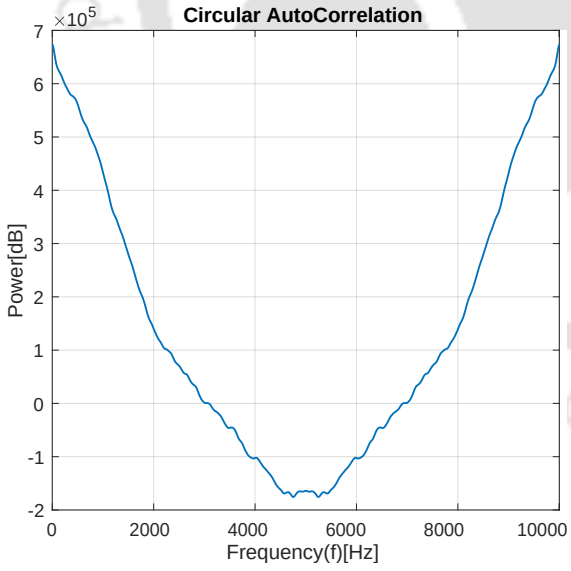
Fisher's G-statistic tests are also performed to further visualize similar properties of periodic and non-periodic trajectories, which are depicted in Figure 4.5. These tests, which are illustrated in Figure 4.5, are periodogram, bob's path traced and the time variation of angles. Periodogram is the power spectral density at each frequency of a bob. Bob's path traced during chaotic oscillations presents an actual physical path traced in 2D plane. The time variation of angles exhibits variations in (θ_1, θ_2, θ_3) with respect to time. The numerical values obtained from periodicity test, circular auto-correlation and other analyses clearly differentiates parameters and initial values, which leads to periodic and aperiodic nature of the motion. Thus, by iterating through all possibles parameters emphasizing nonperiodicity in the selection, a set of keys are generated. In the following section, an implementation of our proposed triple pendulum system is presented to validate its correctness along with a study on its chaotic behavior.



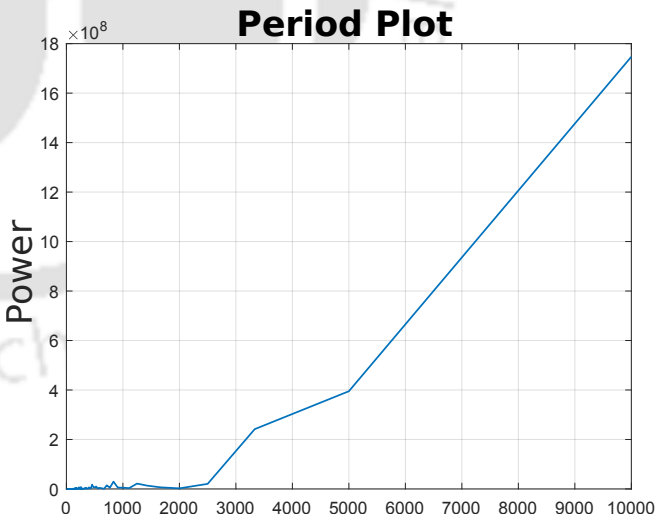
(a) Circular Auto-Correlation for a periodic θ_3



(b) Periodicity for a periodic θ_3

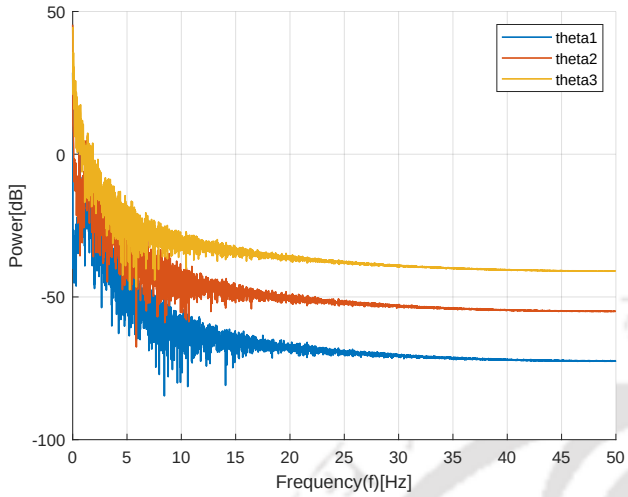


(c) Circular Auto-Correlation for a nonperiodic θ_3

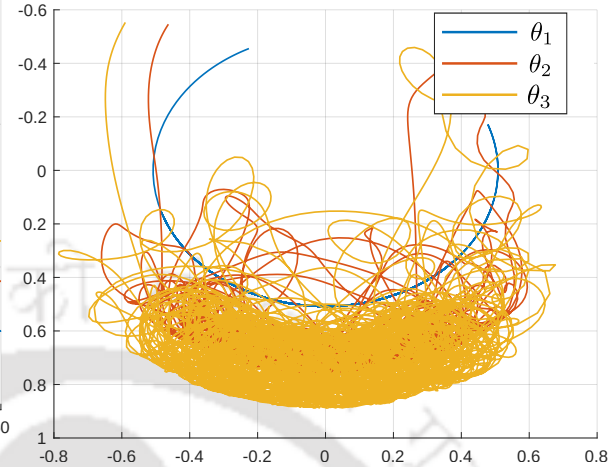


(d) Periodicity for a nonperiodic θ_3

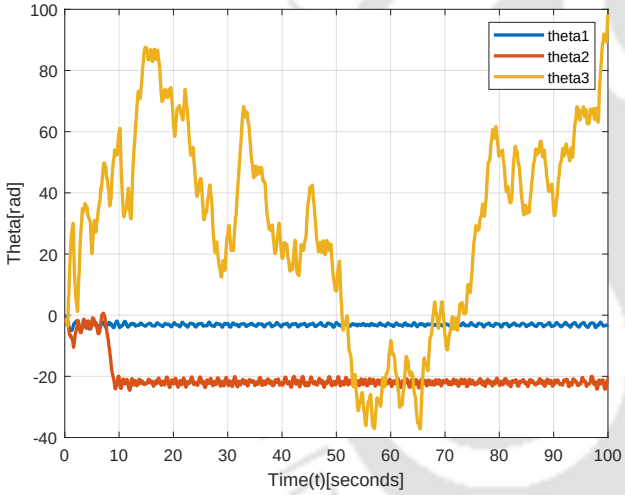
Figure 4.4: Circular auto-correlation and periodicity test for two trajectories of θ_3



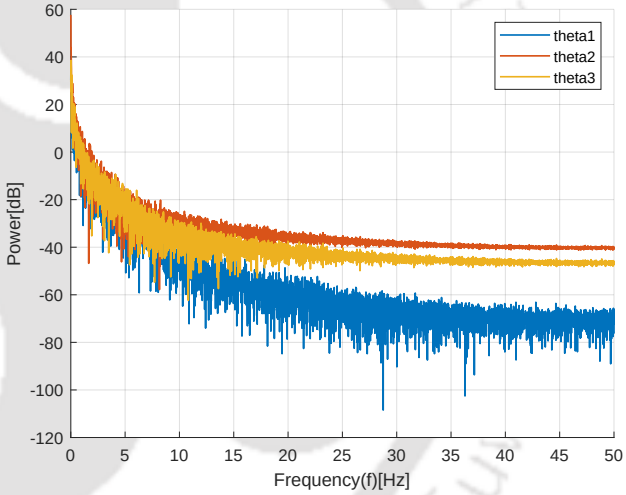
(a) Periodogram power spectral density



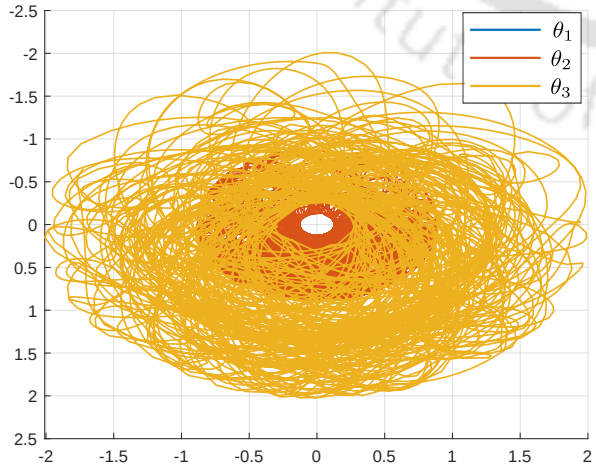
(b) Bob's Path Traced



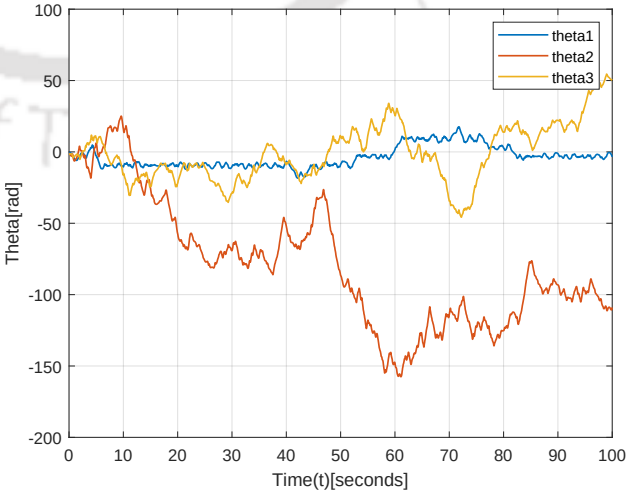
(c) θ vs time (Variation of angles)



(d) Periodogram power spectral density



(e) Bob's Path Traced



(f) θ vs time (Variation of angles)

Figure 4.5: Fisher's G-statistic test with a periodic trajectory (a,b,c) and with a nonperiodic trajectory (d,e,f)

4.4 Key generation and encryption-decryption methodology

In this section, main constituents of our proposed cryptosystem, i.e. key generation and encryption-decryption methodology, are described in detail. The proposed triple pendulum system is implemented using MATLAB in order to validate correctness of the mathematical model presented in section 4.2. Employing initial conditions given in the Table 4.1, the phase wrapped angular displacement profiles of $\ddot{\theta_1}, \ddot{\theta_2}$ and $\ddot{\theta_3}$ are generated. It can be observed that the proposed system is chaotic in nature because of aperiodicity of angular displacements.

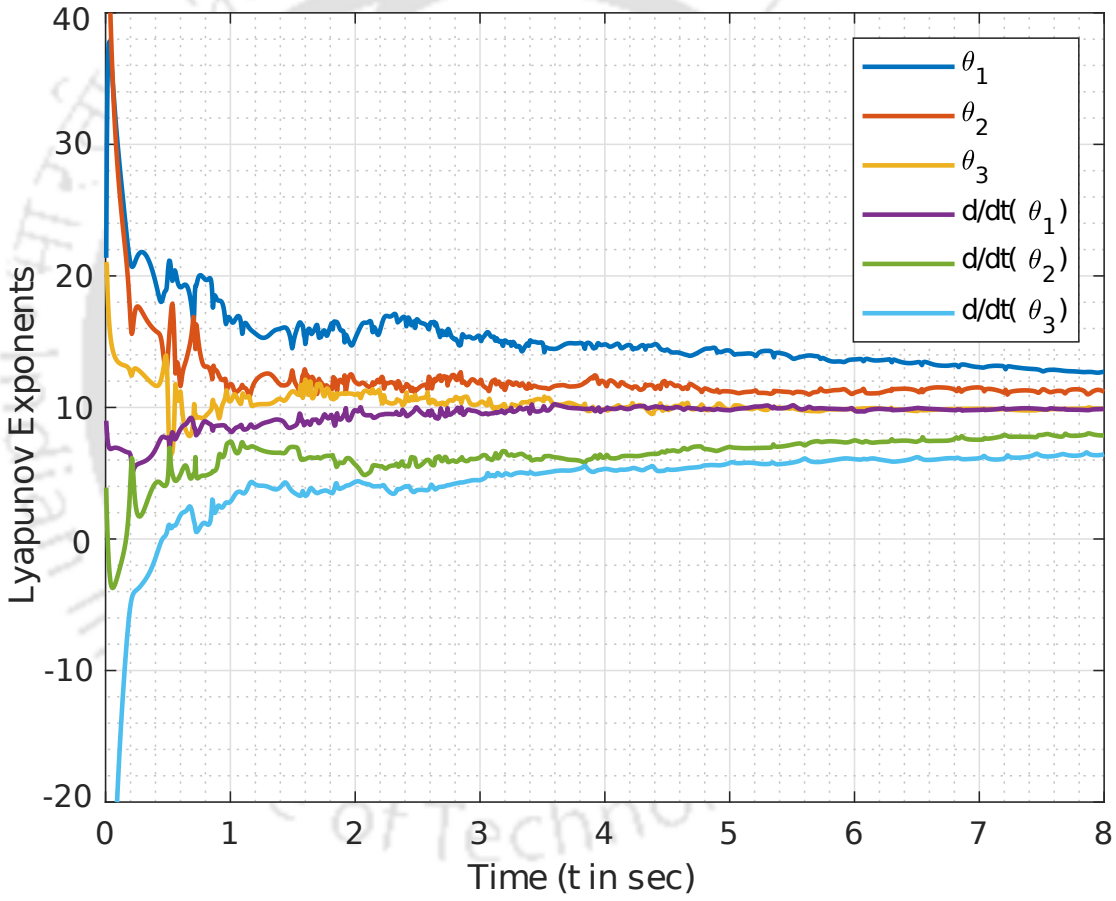


Figure 4.6: Dynamics of Lyapunov exponents time $t = 0$ to 8 sec.

Lyapunov exponents [187] and bifurcation tests [188] for all the key parameters are presented in Figures 4.6 4.7, 4.8 and 4.9 to verify chaotic nature of our proposed system. The encryption scheme based on chaotic dynamic system presented in section 4.2 is validated by attractor profiling obtained using Lyapunov exponents and bifurcation diagrams. The chaotic nature of a dynamic system is estimated by the presence of positive exponents in the Lyapunov test, which implies exponential divergence of the system. It is to mention that chaotic systems do not synchronize with any other

physical system in reality, although their synchronization conditions can be formulated mathematically iff all the initial conditions and parameters resemble exactly with the actual physical systems, which is only possible in an ideal scenario. Thus, two similar chaotic systems produce significantly different outputs, when initial conditions and parameters are marginally different. For estimating Lyapunov exponents [189] for an oscillating chaotic system, a 3D Lorenz equation 4.28 is formulated and is compared with different parameters of our proposed dynamic system. It can be observed from Figure 4.6 that our proposed triple pendulum based dynamic system fulfills Lyapunov criteria and can be considered as a chaotic system.

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{4.28}$$

Thereafter, a bifurcation diagrams are generated for all the independent parameters m_1, m_2, m_3, l_1, l_2, l_3, k_2 and k_3 for θ_1, $\dot{\theta_1}$, θ_2, $\dot{\theta_2}$, θ_3 and $\dot{\theta_3}$ each. It is to mention that generally chaotic nature of systems depends on a single parameter and its bifurcation diagram is produced by changing the value of that particular parameter. Our proposed chaotic dynamic system has total 12 parameters constituting an entire encryption space, and eight parameters among them exhibit highly chaotic behavior. All the bifurcation diagrams illustrated in Figure 4.7, 4.8 and 4.9 are generated by solving $\ddot{\theta}$ using initial conditions of Table 4.1. Using Figure 4.7, 4.8 and 4.9, range of each key values employed for encryption and decryption can be determined. It is to mention that in Figure 4.7, 4.8 and 4.9, θ_1 and $\dot{\theta_1}$, θ_2 and $\dot{\theta_2}$, and θ_3 and $\dot{\theta_3}$ are indicated with green, red and blue colors, respectively.

4.4.1 Key Generation

The key, K, of proposed symmetric cryptosystem includes parameters, such as initial conditions, time duration, time step, the minimum value of $\hat{\mathbf{y}}$, and ϵ. The ϵ is a mapping interval and can be expressed as $\epsilon = \frac{(\hat{y}_{max} - \hat{y}_{min})}{N_c}$. There are total 21 variables constituting basic key structure shown in equation 4.29.

$$K = \{\theta_1^{(0)}, \theta_2^{(0)}, \theta_3^{(0)}, \dot{\theta}_1^{(0)}, \dot{\theta}_2^{(0)}, \dot{\theta}_3^{(0)}, m_1, m_2, m_3, l_1, l_2, l_3, I_1, I_2, I_3, k_1, k_2, k_3, g, \hat{y}_{min}, \hat{y}_{min}/\epsilon\} \tag{4.29}$$

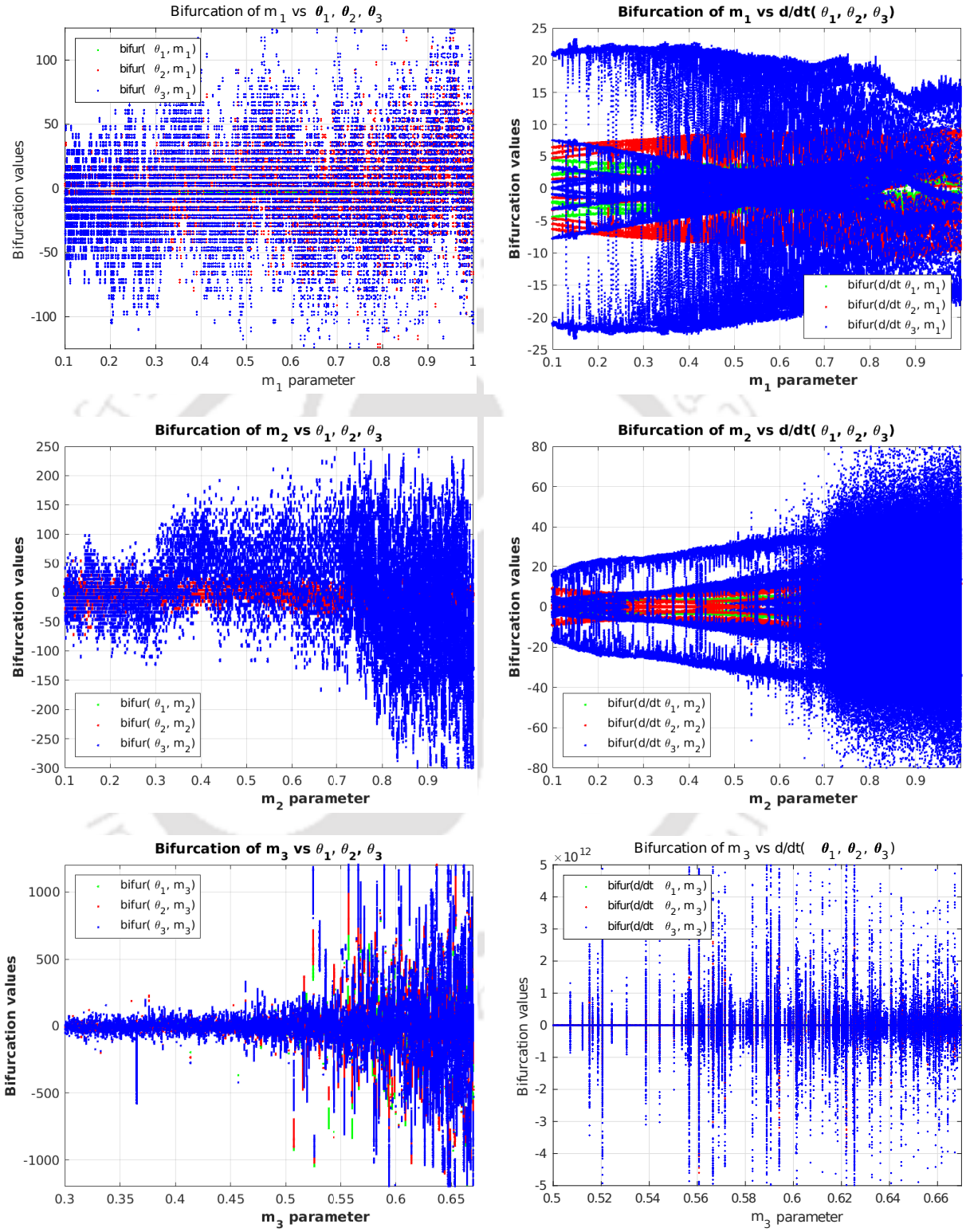


Figure 4.7: Bifurcation diagrams of important independent key parameters $\{m_1, m_2, m_3\}$ for $\{\theta_1, \dot{\theta_1}$ in green$\}$, $\{\theta_2, \dot{\theta_2}$ in red$\}$ and $\{\theta_3, \dot{\theta_3}$ in blue$\}$ [row-wise left to right sequence]

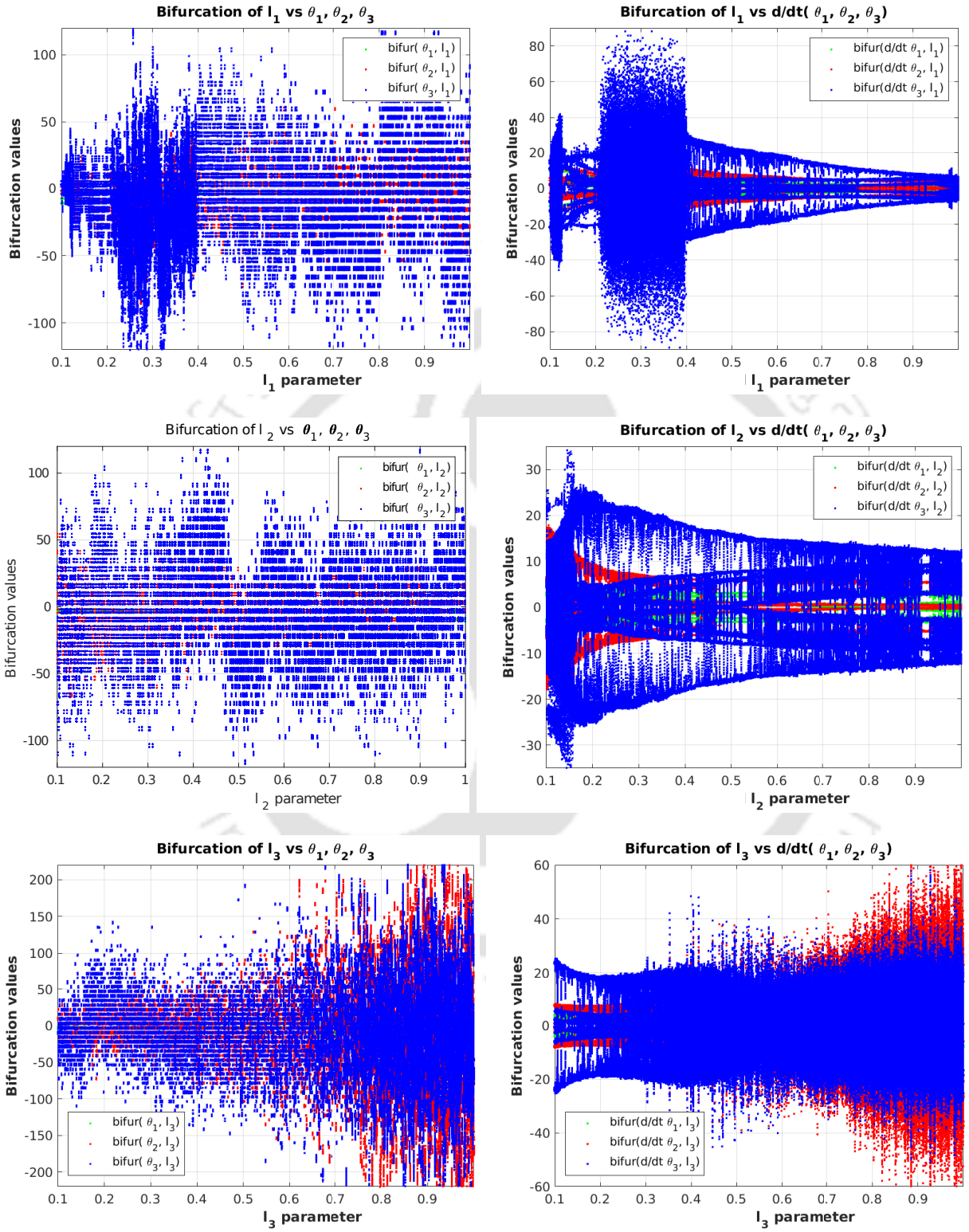


Figure 4.8: Bifurcation diagrams of important independent key parameters $\{l_1, l_2, l_3\}$ for $\{\theta_1, \dot{\theta_1}$ in green$\}$, $\{\theta_2, \dot{\theta_2}$ in red$\}$ and $\{\theta_3, \dot{\theta_3}$ in blue$\}$ [row-wise left to right sequence]

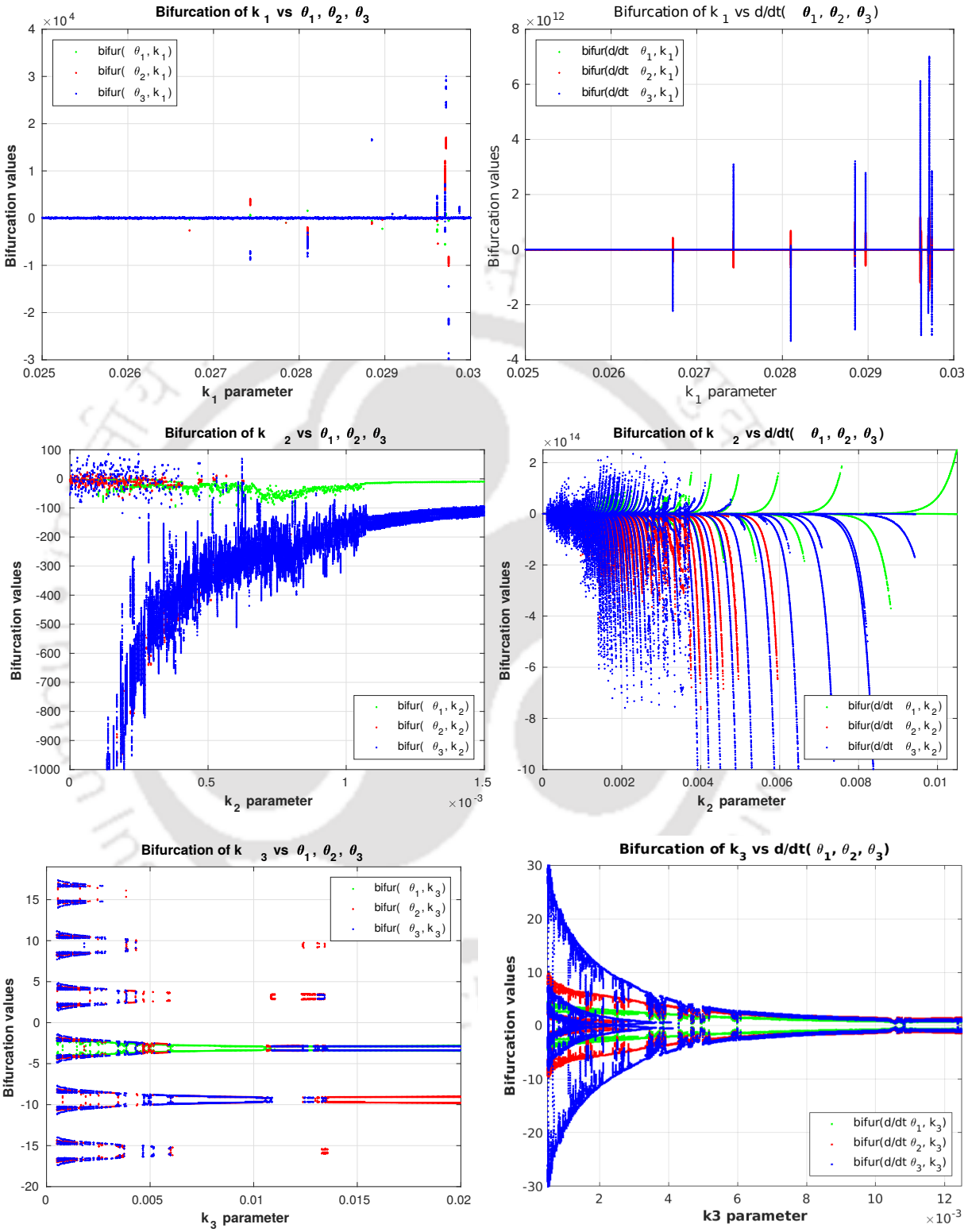


Figure 4.9: Bifurcation diagrams of important independent key parameters $\{k_1, k_2, k_3\}$ for $\{\theta_1, \dot{\theta_1}$ in green$\}$, $\{\theta_2, \dot{\theta_2}$ in red$\}$ and $\{\theta_3, \dot{\theta_3}$ in blue$\}$ [row-wise left to right sequence]

Table 4.2: Size of the key for various PQC algorithms [4]

Algorithm	Symmetric Key size (bits)	Asymmetric Key size (bits)	Key Memory
Lattice Multivariate	14000	6595	<1KB
NTRU	6743	6130	<1KB
Rainbow	740000	991000	125KB
Hash Signature	36000	36000	5KB
Goppa McEliece	92027	8373911	1MB
Quasi-cyclic MDPC McEliece	4384	65542	8KB
SIDH	6144	6144	<1KB
Proposed (single precision)*	672	–	<1KB
Proposed (double precision)*	1344	–	<1KB

$*$ implement for symmetric key

Table 4.2 presents a comparison of key size and key memory for various PQC algorithms. It can be observed that the key size generated by our proposed method is the least amongst all other PQCs. It can be seen that for certain specific parameters or initial conditions, motion of the bobs of a triple-pendulum exhibits periodic nature after a certain span of time. Hence, it becomes imperative to eliminate those parameters or initial conditions for which motion is periodic to preserve chaotic behavior of the system. Therefore, ***Fisher's g-statistic test*** [190], is employed to extract prominent period of the signal using statistical analysis on the spectrum of a signal. This test exploits periodic components of the signal derived from its periodogram, which is used to identify hidden periods (or frequencies) of a time series. This is helpful in identifying dominant periodic behavior in a series. The periodogram indicates relative occurrence of possible frequencies oscillating in between a given interval repeatedly. It estimates *Power Spectral Density* of a signal $x_L(n)$ of length L, which is defined below. Here, F_s is the sampling frequency.

$$P_{xx}(f) = \frac{1}{LF_s} \mid \sum_{n=0}^{L-1} x_L(n) e^{-j2\pi fn/F_s} \mid^2 \tag{4.30}$$

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

$$f_k = \frac{kF_s}{N}, k = 0, 1, ..., N-1 \tag{4.31}$$

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 a signal.

$$g = \frac{\max_k I(\omega_k)}{\sum_{k=1}^{[N/2]} I(\omega_k)} \tag{4.32}$$

where, $I(\omega_k)$ are periodogram values. Using the values of g, dominant periods of a signal are calculated. Thus, after removing the range of the parameters, in which a signal exhibits possible periodic behavior, remaining values of the parameters can be utilized to span key space. Further, based on the bifurcation diagrams, the strongest key may be chosen for encryption and decryption steps. Since bifurcation is a resource intensive process, there is a tradeoff between power and choosing a stronger key for an application. In the following subsection encryption and decryption methods are explained in detail. Key generation of the proposed cryptosystem is presented in the Algorithm 8. Here, "get_tp_trajectory" function generates the trajectory of triple pendulum based on the input arguments, "is_periodic" performs the Fisher's statistical tests, "get_lyapunov_exp" obtains lyapunov exponents and stores in $\lambda(t)$, S is the overall set of keys, and s denotes individual key.

Algorithm 8: Key Generation for the Proposed Cryptosystem

Input: M_{min}, M_{max}, L_{min}, L_{max}, K_{min}, K_{max}, I_{min}, I_{max}, Δ, t_{start}, t_{end}, t_Δ
Output: **K**

Initialize $M = \{M_{min} : \Delta : M_{max}\}$, $L = \{L_{min} : \Delta : L_{max}\}$, $I = \{I_{min} : \Delta : I_{max}\}$, $K = \{K_{min} : \Delta : K_{max}\}$, $S = \{\}$
For $i = 1$ to M
  For $j = 1$ to L
    For $k = 1$ to I
      For $l = 1$ to K
      key=$m1_i, m2_i, m3_i, L1_j, L2_j, L3_j, I1_k, I2_k, I3_k, K1_l, K2_l, K3_l$
       (y, t)=get_tp_trajectory(key, t_{start}, t_{end}, t_Δ)
       IF is_periodic(y, t)==True
          continues the loop
       Calculate $\lambda(t)$=get_lyapunov_exp(y, t)
        IF $\lambda(t) > 0 \; \forall \; t$:
           $S = S \bigcup key$;
           continues the loop
K = choose random $s \in S$;
Return **K**

4.4.2 Encryption - Decryption

The approach discussed here is to convert plain text of M characters into an ASCII format and map it to the intervals formulated using state variables of the proposed triple-pendulum system. Figure 4.10 exhibits flow of this approach. The initial conditions and parameters of different equations constitute

a part of the private key. The differential equations are numerically integrated to get state variable $\theta_1, \theta_2, \theta_3$ followed by phase wrapping in the range of $[0, 2\pi]$ or $[-\pi, \pi]$. Later, employing Baptista-type method, an entire range of chaotic function $[\hat{y}_{min}, \hat{y}_{max}]$ is partitioned into a number of intervals equal to the number of characters (N_c). Each character in the plain-text $\mathbf{P}$ is first mapped to a specific interval (k) and is stamped with a time point (i) randomly selected from this interval. A permutation map $\pi(k)$ is employed to determine interval identifiers for each character's ASCII value. Proposed encryption scheme is depicted in Algorithm 9. Here, η is a constant and $\mathbf{C}$ is cipher text.

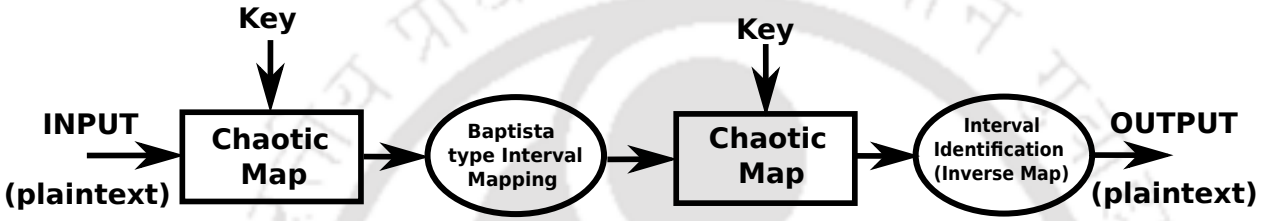


Figure 4.10: Encryption-Decryption Strategy

Algorithm 9: Proposed Encryption Algorithm

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

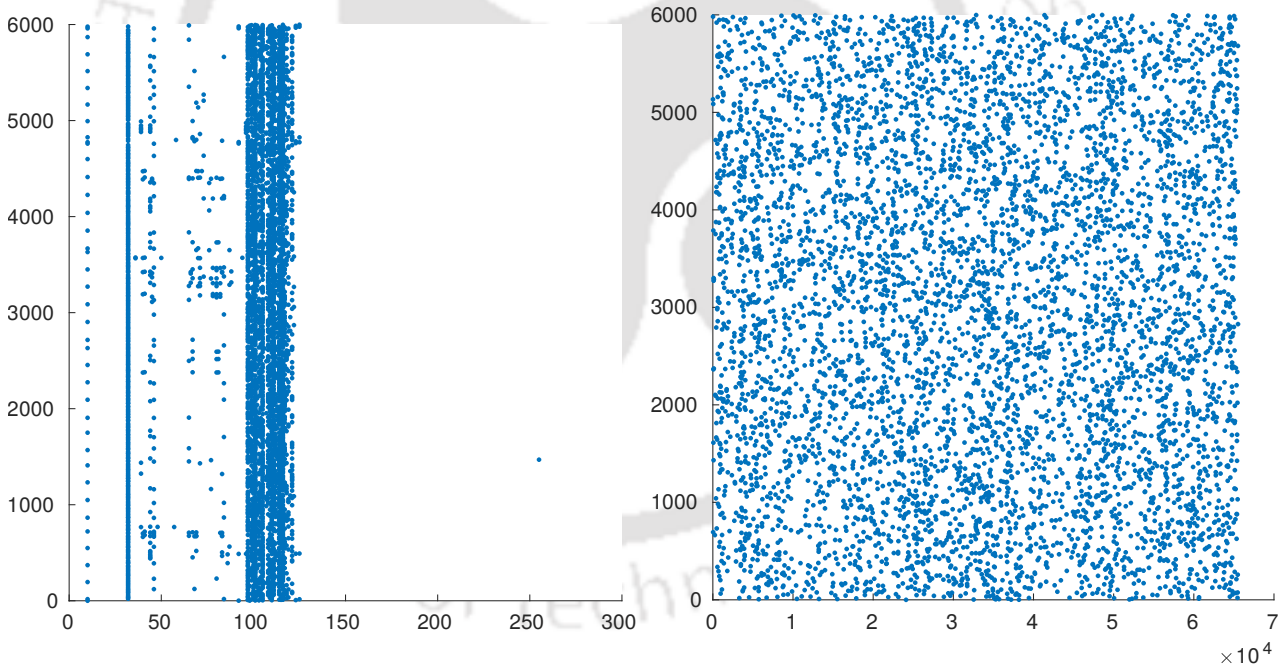
Initialize $D = \{\}$, k = size(K)
Find $\hat{y}_{max} = \max_{1 \le i \le N}(\hat{y}_i)$, $\hat{y}_{min} = \min_{1 \le i \le N}(\hat{y}_i)$
Calculate $\epsilon = \frac{(\hat{y}_{max} - \hat{y}_{min})}{N_c}$
For each $\hat{y}_i$ in $\hat{\mathbf{y}}$
  Calculate $d = \lfloor \frac{\hat{y}_i - \hat{y}_{min}}{\epsilon} \rfloor$
  $D(\pi(d)) \leftarrow D(\pi(d)) \cup \{i\}$
For $m = 1$ to M
  Initialize $j = 0$, $p = \mathbf{P}(m)$
  Calculate $\hat{p} = ASCII(p)$, $l = size(D(\pi(\hat{p})))$
  Generate random number r, $0 \le r \le 1$
  While $r \le \eta$, do $j = (j + 1)\%l$
Else Calculate $f_k = half(K(m\%k))$, where $half(n)$ converts n to Half-Precision Floating Point format
$t = ROL(D(\pi(\hat{p}), j + 1), m)$ where $ROL(n, b)$ rotates n left by b-bits
$c = t \oplus f_k$
$\mathbf{C}(m) = c$
Return $\mathbf{C}$

In the decryption module, an interval, in which an encrypted value lies, is computed using the same symmetric key. By applying inverse permutation map ($\pi^{-1}(k)$), corresponding indices are converted back to clear-text ($\tilde{P}$). The message can be decoded by converting $\tilde{P}$ into characters. Proposed decryption scheme is presented in Algorithm 10.

Algorithm 10: Proposed decryption algorithm

Input: $\hat{\mathbf{y}}$,$\mathbf{C}$, $\mathbf{K}$
Output: $\tilde{\mathbf{P}}$

Initialize $k = size(K)$
Find $\hat{y}_{max} = \max_{1 \le i \le N}(\hat{y}_i)$; $\hat{y}_{min} = \min_{1 \le i \le N}(\hat{y}_i)$
Calculate $\epsilon = \frac{(\hat{y}_{max} - \hat{y}_{min})}{N_c}$
For $m = 1$ to M
    Initialize $c = \mathbf{C}(m)$
    Calculate $f_k = half(K(m\%k))$, where $half(n)$ converts n to Half-Precision Floating Point format
$t = c \oplus f_k$
$\hat{C} = \text{ROR}(t, m)$, where $\text{ROR}(n, b)$ rotates n right by b-bits
Calculate $\hat{k} = \lfloor \frac{\hat{y}_c - \hat{y}_{min}}{\epsilon} \rfloor$
    Compute $k = \pi^{-1}(\hat{k})$
    $\tilde{\mathbf{P}}(m) \leftarrow CHAR(k)$
Return $\tilde{\mathbf{P}}$



(a) Initial 6000 ASCII characters distribution of plain-text (b) Corresponding cipher distribution of the plain-text

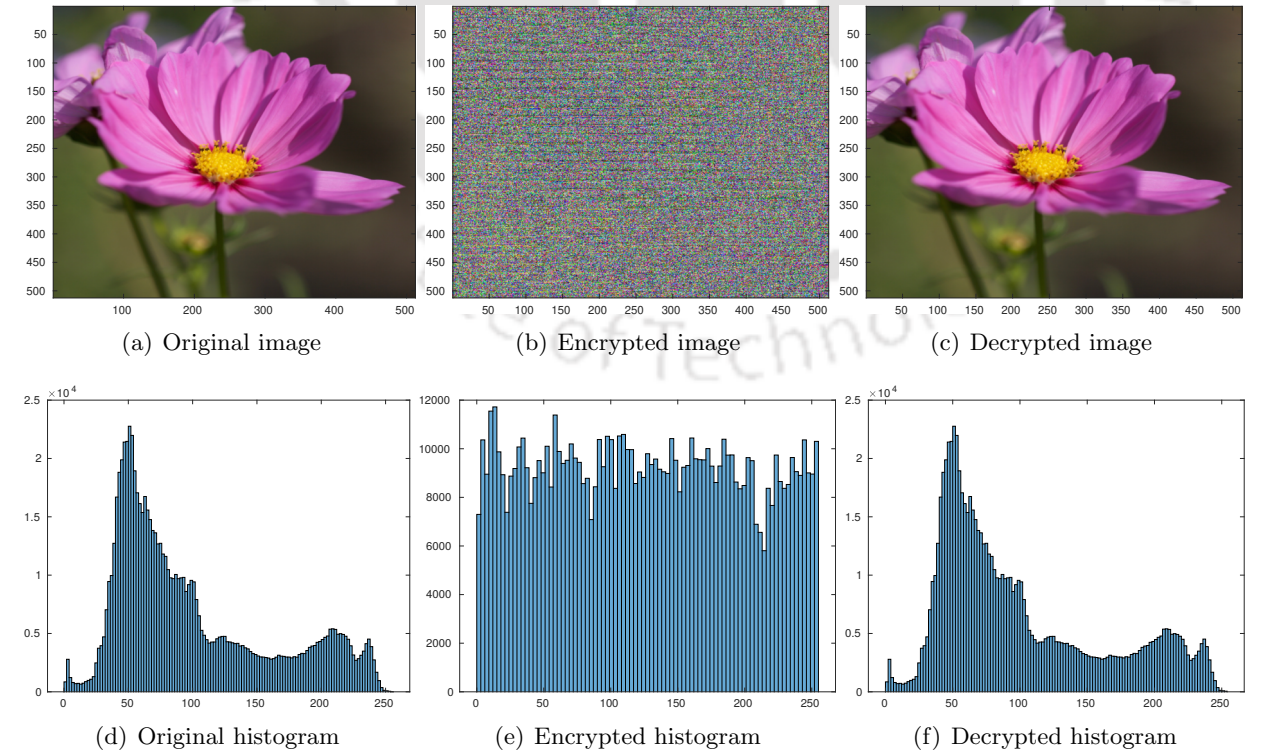
Figure 4.11: Text data encryption example for 100KB plain text based on the key value listed in Table 4.1

Figure 4.11 showcases encryption of text data, where Figure 4.11(a) shows ASCII distribution of 1st 6000 characters (plain-text) and the Figure 4.11(b) is the generated ciphertext after applying our proposed encryption algorithm. In addition to the plain text encryption, the proposed method can also be applied to images with RGB values ranging from 0 to 255. The parameter N_c in Algorithm 9

(a) Original image (b) Encrypted image (c) Decrypted image

(d) Original histogram (e) Encrypted histogram (f) Decrypted histogram

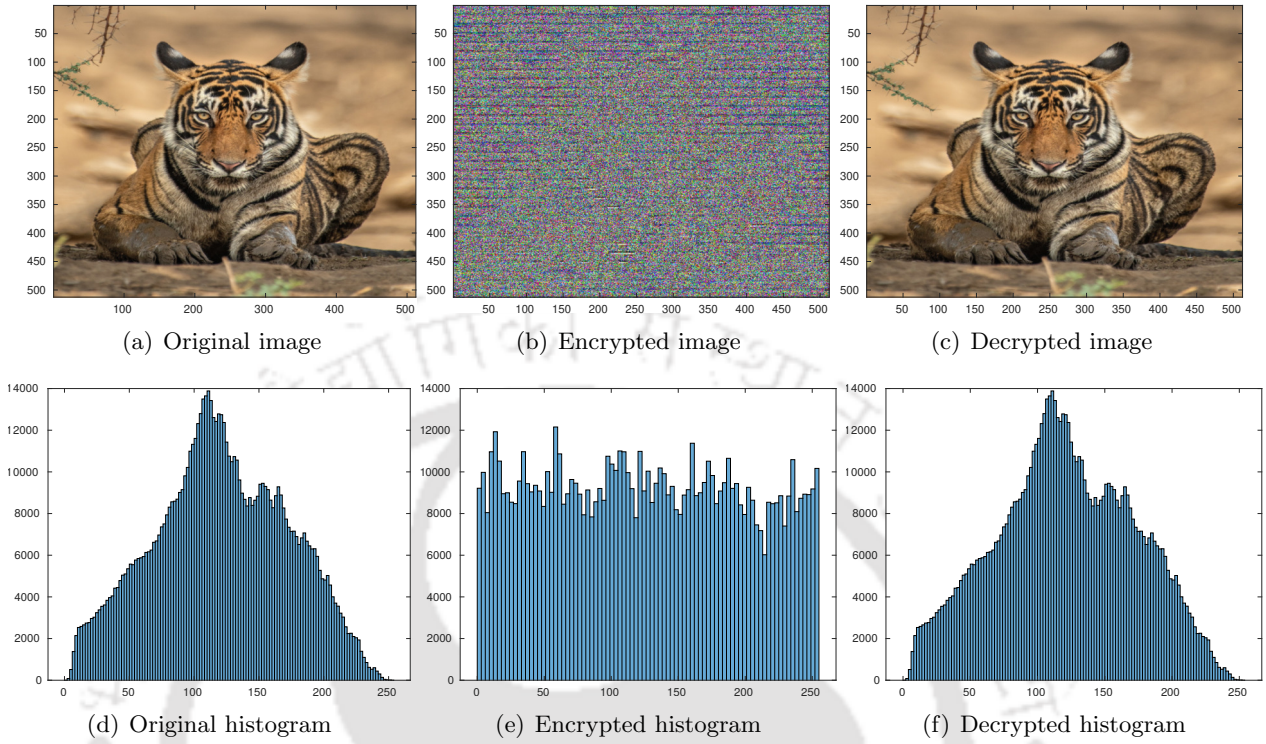
Figure 4.12: Encryption decryption example image 1 for key value of Table 4.1



(a) Original image (b) Encrypted image (c) Decrypted image

(d) Original histogram (e) Encrypted histogram (f) Decrypted histogram

Figure 4.13: Encryption decryption example image 2 for key value of Table 4.1



(a) Original image (b) Encrypted image (c) Decrypted image

(d) Original histogram (e) Encrypted histogram (f) Decrypted histogram

Figure 4.14: Encryption decryption example image 3 for key value of Table 4.1

can be set to 256 and a serialized image is fed as an input for image encryption. This algorithm can be extended for video encryption as well by treating each frame as an independent image. The proposed encryption and decryption algorithms are demonstrated on three set of images. The original, encrypted and decrypted images are illustrated with their corresponding histograms presented in Figure 4.12, 4.13 and 4.14. Parameters of the key used for encryption and decryption in both types of applications, text and image, are $m_1 = 0.2944$, $m_2 = 0.8756$, $m_3 = 0.0947$, $l_1 = 0.508$, $l_2 = 0.254$, $l_3 = 0.127$, $k_1 = 0.005$, $k_2 = 0$, $k_3 = 0.0008$, $I_1 = 9.526 \times 10^{-3}$, $I_2 = 1.625 \times 10^{-3}$, $I_3 = 1.848 \times 10^{-4}$, $g = 9.81$. Selection of these parameters for a key is accomplished using bifurcation diagram shown in Figure 4.7, 4.8 and 4.9. Although, histograms of input images are vastly different, but histograms of encrypted images are similar in nature. This makes proposed scheme less vulnerable as compared to other methods, because it is hard to determine input just by using an encrypted histogram without knowing key. It is to mention that the proposed scheme has wide range of applicability.

The encryption and decryption algorithm are validated with text and various image inputs. The correctness of the decryption can only proven is a message encrypted with a key and generate cipher-text and operating over the same ciphertext if the decryption produces identical message which is

denoted by decryption(encryption(message))=message. From Algorithm 9, step no 4 to 8 are followed chronologically to generate ciphertexts. In step 4 intervals are calculated in which plaintext to ciphertexts mapping would be performed, then in step 5 then actual mapping from ASCII to chaotic map is completed. Step 6 is for floating point to integer conversion and step 7 and 8 is for rotating the intermediate ciphertexts left by certain bit position and XORing with the key to generate the final ciphertext. Now for the decryption algorithm we can observe that above mentioned are computed in opposite sequence. In decryption operation first we XOR with symmetric key to nullify the XOR operation done in encryption side and subsequently rotate right by the same bit position and lastly map to the range to obtain message text.

4.4.3 Complexity Analysis

The computational complexity of the proposed scheme is described below.

- The computation of state variables (θ) for a duration t and time step ΔT requires N iterations, where $N = \frac{T}{\Delta t}$. In each iteration, only six equations are analyzed to find six state variables $(\theta_1, \theta_2, \theta_3, \dot{\theta_1}, \dot{\theta_2}, \dot{\theta_3})$ in $O(1)$ time. Thus, the time required for generating an asymptotical map is $O(N)$.

- The space complexity for storing state variables is $O(N)$.

- Baptista type partitioning takes constant time, if range of the variables is already computed in the previous step. For assigning variables to corresponding intervals, $O(N)$ time is required for each variable. The $K_c = 21$ variables constituting symmetric key in our proposed method, $K_c O(N)$ time is required.

- The encryption of a plain-text having M characters utilizes M iterations. In each iteration, an encrypted value is selected randomly in an interval in a constant time. Therefore, time complexities for setting up of an encryption module and processing each plaintext through are $O(N)$ and $O(M)$, respectively.

- Space complexity of an encryption module is $O(N)$.

- The time complexities of setting up decryption module and its execution are $O(N)$ and $O(M)$, respectively. The information rate of the proposed cryptosystem is defined as the ratio of the

size of plain-text, SZ_{pt}, to the size of cipher-text, SZ_{ct}, which is defined below.

$$R = \frac{SZ_{pt}}{SZ_{ct}} = \frac{8 * M}{\lceil \log_2 N \rceil * M} = \frac{8}{\lceil \log_2 N \rceil} \tag{4.33}$$

The overall time complexity of the complete encryption and decryption methods is the sum of individual complexities mentioned above, which add upto $O(N)$. Therefore, it can be utilized in many applications efficiently to make them secure. Since, both time and space complexities of the proposed method are linear, it offers a very good scope for its parallelization to boost performance at hardware level as well. Due to its immunity to quantum computing based attacks, scalability and being computationally simple, our proposed method becomes a prominent candidate to provide much needed security to various wide range of applications, such as IoT based smart systems, high throughput low latency systems etc.

4.5 Hardware Implementation

The proposed design has been implemented using System Verilog HDL on Digilent Zed-Board having Xilinx Zynq-7 FPGA. Its synthesized netlist is generated by Synopsis Design Compiler (DC) employing SCL's 180nm design libraries. For simplicity and to validate the proposed algorithms, permutation map ($\pi(k)$) is fixed as an identity map and maximum three state variables are stored in a particular interval for each character during an encryption phase. As a result, it is required to search over total number of characters, N_c, for an interval during decryption cycle. For evaluating equation 4.24, its RHS expression is converted into a postfix format and the resultant postfix expression is stored in a ROM on an FPGA board. Later, this expression is evaluated using a stack-based postfix evaluation technique. The angle combinations of sinusoidal terms present in the expressions are cached in an LUT and are appropriately decoded while evaluating these expression. The overall block diagram of this scheme is presented in Figure 4.15.The hardware design can be segregated into three main parts, viz. LUT and Memory, ALU and Control Unit. A brief description of these units are elaborated in the following subsections.

4.5.1 Look Up Table (LUT) and Memory

In the FPGA implementation, five LUTs and a stack memory are utilized for solving differential equations by evaluating postfix expressions, which depends on angle combinations presented in Table 4.3. These five LUTs are used to store *Parameters, Angle Combinations, State Variables, Numeric*

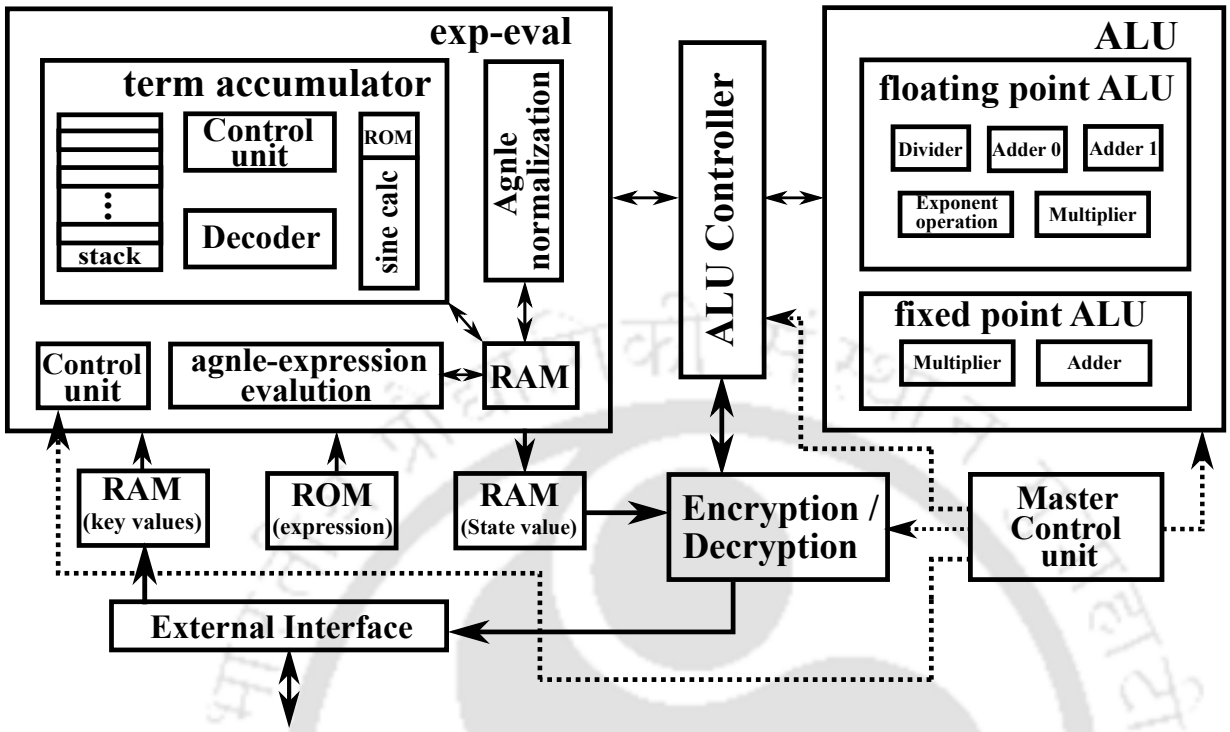


Figure 4.15: Overall block diagram of proposed scheme

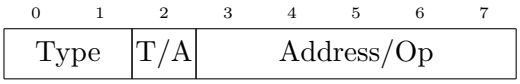
Constants and Postfix Evaluations. Parameters represented in Table 4.1 are employed for the above mentioned purpose.

Table 4.3: LUT Details

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

As it is mentioned earlier, in the implementation of postfix evaluations, RHS expressions in equation 4.24 are converted into postfix format. The resultant postfix expression constitutes numeric constants, state variables or parameters, operations and sinusoidal terms. Each of these terms is encoded to a 8-bit code and an entire postfix expression is stored in an LUT in the coded format. During postfix

evaluation, these codes are decoded and an appropriate data value is fetched from a corresponding address. The 8-bit encoding format can be expressed as shown below.



The mathematical model of triple pendulum system presented in equation 4.21, 4.22 and 4.23 are parsed and evaluated using information presented in Table 4.4. Thereafter, stack memory is configured to execute encryption and decryption modules.

Table 4.4: Encoding for mathematical operations using 8-bit postfix format

Type	T/A	Address/Operation	Remarks
00	X	XXXXX	Constant terms
01	0	0x00 to 0x1F (addr.)	Parameters from Table I & II
	1	0x00 to 0x1F (addr.)	State variables
10	0	0x00 to 0x1F (addr.)	Sine values from Table IV
	1	0x00 to 0x1F (addr.)	Cosine values from Table IV
01	X	XX000 (op.)	Exponentiation
		XX001 (op.)	Multiplication
		XX010 (op.)	Division
		XX011 (op.)	Addition
		XX100 (op.)	Subtraction
11	1	11111 (op.)	End of expressions

X = don't care; op. = operation; addr. = memory location of LUT

Stack: A stack consisting of two immediate registers and a memory is implemented for the evaluation of postfix expressions. The immediate registers store top two values of the stack. An operation is performed in an expression employing these two immediate registers and its result is stored in the first register. Another register is again filled with a value popped from top of the stack for next operation. It continues until all the operations related with a mathematical expression are completed.

When a complex operation requires more resources than two immediate registers, then this operation is performed in two or more than two steps. In this operation, value stored in the second immediate register is pushed onto the stack and, corresponding memory location and program counter are updated. This process repeats until a complex operation is executed producing a correct output. This approach reduces resource utilization while performing a complex operation. Another advantage of using stack-based push-pop architecture is that it can be used for solving any differential equa-

tion expressed in the form of state-space equation. For this operation, only angle combinations and mnemonics stored in the stack need to be changed keeping hardware architecture identical for all the operations. The brief discussion on Arithmetic and Logic Unit (ALU) is presented in the following section.

4.5.2 Arithmetic and Logic Unit (ALU)

In the proposed design, an arithmetic and logical unit (ALU) consists of two addition, one multiplication, one exponentiation and one division modules. All these modules perform floating point operations in the standard IEEE-754 format. The datapath in the addition module is split into four stages. The clock frequency of this operation is fixed at 100 MHz to meet timing constraints of the FPGA employed in validation of the proposed design. The datapath in multiplication module is divided into five stages to achieve above mentioned clock frequency. Moreover, if required, the multiplication module can be configured to function in a pipelined mode. Exponentiation module evaluates value of a floating point input raised to an integer as an exponent. It instantiates multiplication module internally for the evaluation of an exponent. Datapath of division module is sliced into multiple stages to meet the target clock frequency of 100 MHz. However, the division operation requires significant clock cycles as compared to other operations, therefore, the design of proposed algorithms is optimized to keep the number of division operations as less as possible.

4.5.3 Control Unit

The control unit of the proposed design is essentially a finite state machine (FSM), which performs required tasks in a well-defined sequence. In order to implement it efficiently, the complete FSM is factored into smaller state machines hierarchically. The top level FSM serves as a controller for all the smaller FSMs. Detailed descriptions for these state machines are given below.

(i) **Top Level State Machine:** Figure 4.16 exhibits a simplified FSM of the top module. Description of each state is given below.

- DEFAULT: It is an initial state of FSM after reset.
- RECEIVE_KEY: It receives encryption or decryption keys through USB or Ethernet interfaces.

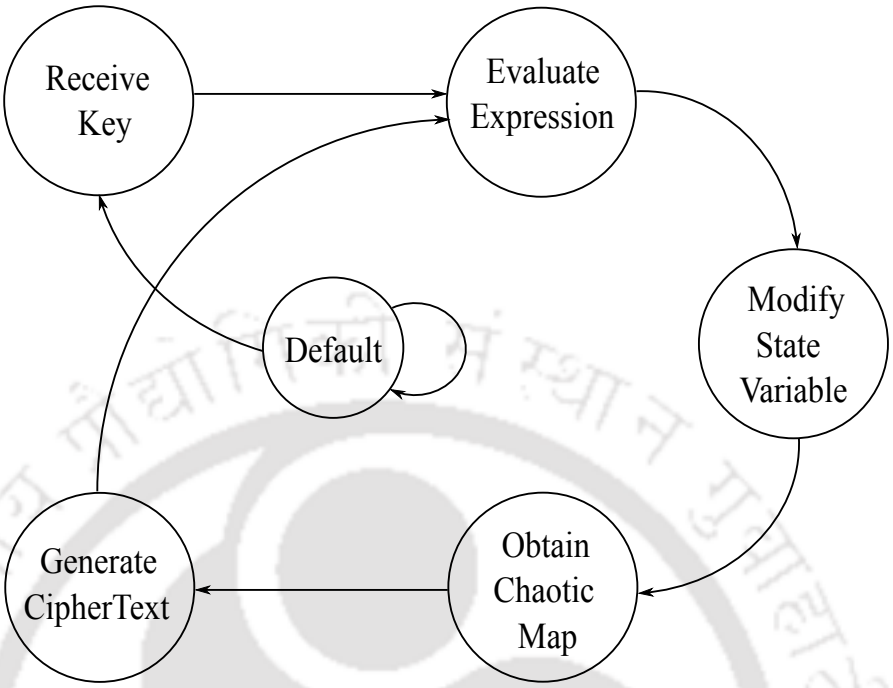


Figure 4.16: Top-Level FSM

- EVALUATE_EXPRESSION: It evaluates state-space expression using given initial conditions for a chaotic dynamic system.
- MODIFY_STATE_VARIABLES: In this state, state variables are updated based on the previous state. It involves multiply and addition modules to perform updated operations.
- OBTAIN_CHAOTIC_MAP: It evaluates a chaotic map through a linear combination of all state variables and calculates a linear combination of θ, $\dot{\theta}$ and $\ddot{\theta}$. It is to mention that θ corresponds to the chaotic map. Further, it calculates intervals, in which θ lies and adds it to corresponding address in LUT.
- GENERATE_CIPHERTEXT: It generates a ciphertext for a particular plaintext string using a generated chaotic map and utilizes LUT to convert plaintext to ciphertext.

(ii) **Evaluate State-Space Expression:** Figure 4.17 exhibits an FSM of a state space solver module and its detailed description is presented below.

- DEFAULT: This is an initial state of FSM after reset.
- FETCH_INITIAL_CONDITIONS: It fetches required initial conditions, θ and $\dot{\theta}$, from Block-RAM (BRAM).

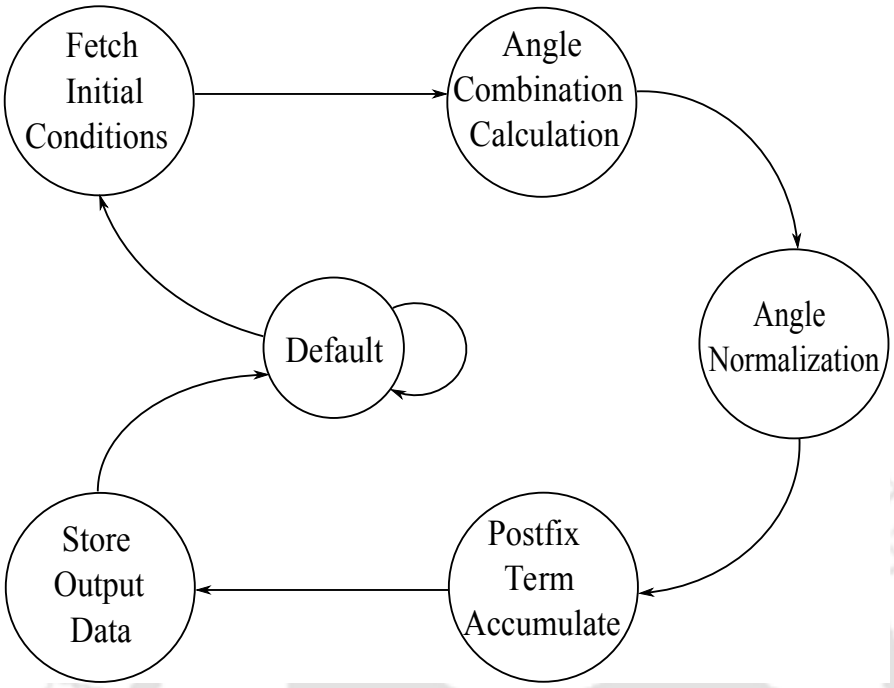


Figure 4.17: FSM for State-Space solver

- ANGLE_COMBINATION_CALC: This state generates linear combinations of state variables.
- ANGLE_NORMALIZATION: In this state, trigonometric inputs are normalized in the range of $[-\pi, \pi]$.
- POSTFIX_TERM_ACCUMULATE: It evaluates a state-space expression stored in the postfix form.
- STORE_OUTPUT_DATA: This state stores state-space expression obtained in the previous state in the memory.

(iii) **Postfix Expression Evaluation:** Figure 4.18 presents an FSM of postfix expression evaluation, which outputs encrypted/decrypted values. A description of each state is described below.

- DEFAULT: This is an initial state of FSM after reset.
- READ_POSTFIX_CODE: In this state, mnemonic of the next term of a postfix expression is fetched from the stack.
- DATA OR OPERATOR: This state checks whether a fetched code corresponds to data or an operator.

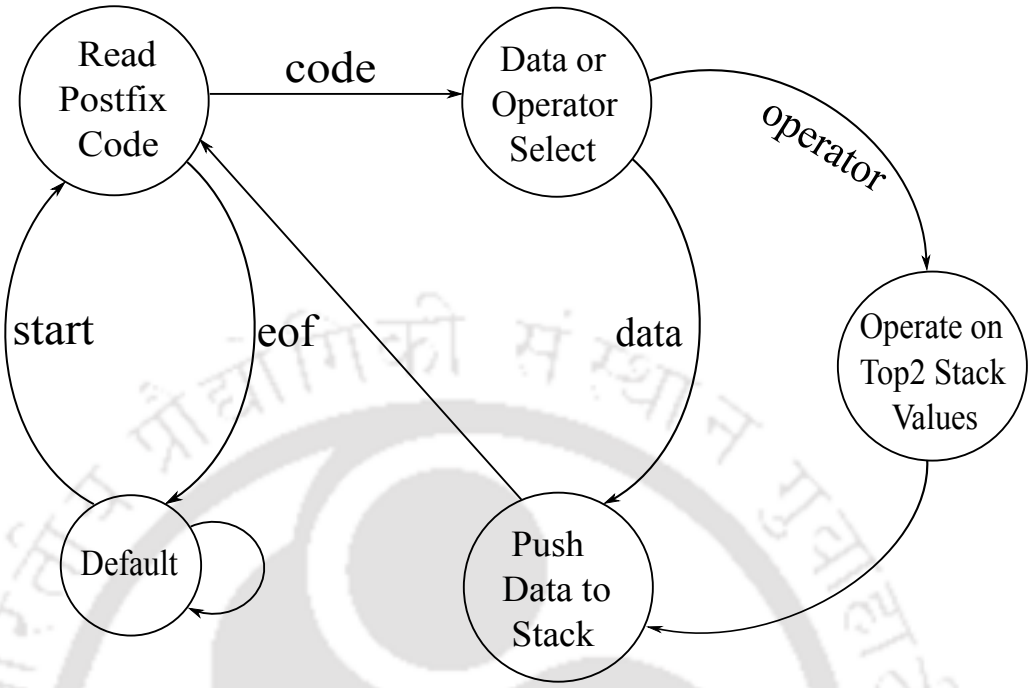


Figure 4.18: FSM for Postfix Evaluator

- OPERATE_ON_TOP_TWO_STACK_VALUES: It performs an operation based on top two data values stored in a stack.
- PUSH_DATA_TO_STACK: It pushes data in a postfix expression or result of an ALU operation onto the stack.

It is to mention that the differentiating factor between encryption and decryption operations in *Postfix Expression Evaluation FSM* depends on postfix expressions, which are stored in postfix evaluation LUTs. Employing the above mentioned FSMs, encryption and decryption algorithms proposed in this chapter are implemented and validated against various benchmarks. In the next section, implementation results of these methods on various platforms are compared and discussed in detail.

4.6 Results and Analysis

In this section, we present hardware implementation results and relevant tests for the randomness quality of a chaotic map generated by the proposed approach.

4.6.1 FPGA and ASIC Implementation Results

The detailed analysis of the proposed encryption and decryption algorithms is presented in previous section to validate their correctness and efficacy. Later, these methods are implemented on hardware using System Verilog and is synthesized using Xilinx Vivado [191]. It is to mention that the results obtained using hardware implementation of these algorithms match with their MATLAB based implementations accurately.

Table 4.5: Comparison With Existing Chaos Based Cryptosystem

Work	LUT/FF/Slice/BRAM/ DSP/ BUFG	Comparative Resource Utilization	Power (mW)	Delay (ns)	PDP[a] (nJ)	Throughput (MBps)
Adrián et al. [192]	10943/3629/ 3190/*/144/*	**1.825**×	332	192	63.744	1000
Pérez-Resa et al. [193]	16978/11127/ 5636/77/*/*	**3.446**×	775	8	6.2	1000
Richard-1 et al. [194]	7426/11285/ 3268/343/*/*	**2.275**×	∼	∼	∼	41.1
Richard-2 et al. [194]	3587/5592/ 1596/170/*/*	**1.115**×	∼	∼	∼	109.6
Proposed	3042/3519/ 3237/5/8/2	**1**×	186	1.592	0.296	2396.164

∼: not specified; *: Resource not utilized; [a]: Power-delay-product.

Table 4.5 exhibits a detailed comparison and resource utilization of our hardware implementation with other prior art work, such as Adrián et al. [192], Pérez-Resa et al. [193] and two variants of Richard et al. [194] etc., reported in the literature. It can be observed that the proposed design consumes 1.825×, 3.446×, 2.275× and 1.115× lesser resource and produces 2.396×, 2.396×, 58.3× and 21.863× greater throughput than [192], [193] and two variants presented in [194], respectively. Further, the proposed scheme is 1.785× and 4.167× power efficient and 120.6× and 5.025× less delay while compared with [192] and [193], respectively. It is to mention that our proposed method exploits maximum hardware resources, memory and LUTs, in storing state variables only. Although it enables proposed design to be memory optimized and most of the static power is consumed for the memory. Since FPGA is the target device, a few DSP slices (DSP48) are employed during synthesis of the proposed design, which enable efficient implementation of trigonometric functions. Therefore, it can be summarized that the proposed scheme is resource optimal, delay efficient and presents better throughput than other state-of-the-art methods.

It can be observed further that the power consumption of our proposed design is 186 mW. Its

Table 4.6: Actual resource utilization for proposed scheme on SCL's 180nm technology node at 250MHz

Parameters	Value
Total std. cells	6229
Combinational cells	5086
Sequential cells	1105
Number of buf/inv	1075
Number of references	38
Combinational area	113800.501382 μm^2
Buf/Inv area	9217.829951 μm^2
Non combinational area	83402.028854 μm^2
Net Interconnect area	6533.662340 μm^2
Total cell area	197202.530237 μm^2
Total area	**203736.192577** μm^2
Internal Power	51.1217 mW
Switching Power	10.7619 mW
Leakage Power	1.0772 μW
Total Power	**61.8836 mW**

throughput and theoretical operating frequency are 2396.164 MBps and 628.14 MHz, respectively. It has the least power-delay product when compared with other designs, which indicates that our proposed design is energy efficient as well. Thus, the high throughput of our design along with a very high energy efficiency and very high strength against attacks make it a suitable candidate to be used in the wide range of electronics systems ranging from smart IoT devices to low latency high performance systems.

Further, the proposed design is realized employing 180nm Bulk CMOS technology available with Semi-Conductor Lab, India. Table 4.6 presents actual physical area, power and resource utilization estimated using above mentioned technology node at 250MHz clock. The chip layout of the proposed design illustrated in Figure 4.19 is developed by using Electronic Design Automation (EDA) tool, IC Compiler, provided by Synopsys Inc. Similarly, the synthesized design netlist is generated by employing Design Complier provided by Synopsys Inc. It is to mention that all the design parameters exhibited in Table 4.6 are extracted through IC Compiler.

In order to validate usage of our proposed design in various applications, its power profile is analyzed by varying operating frequency from 1KHz to 500 MHz, which is presented in Figure 4.20. The total power measured at a particular frequency is the sum of static, dynamic and leakage power. It can be also be observed in Figure 4.20 that the static power contributes significantly to the total

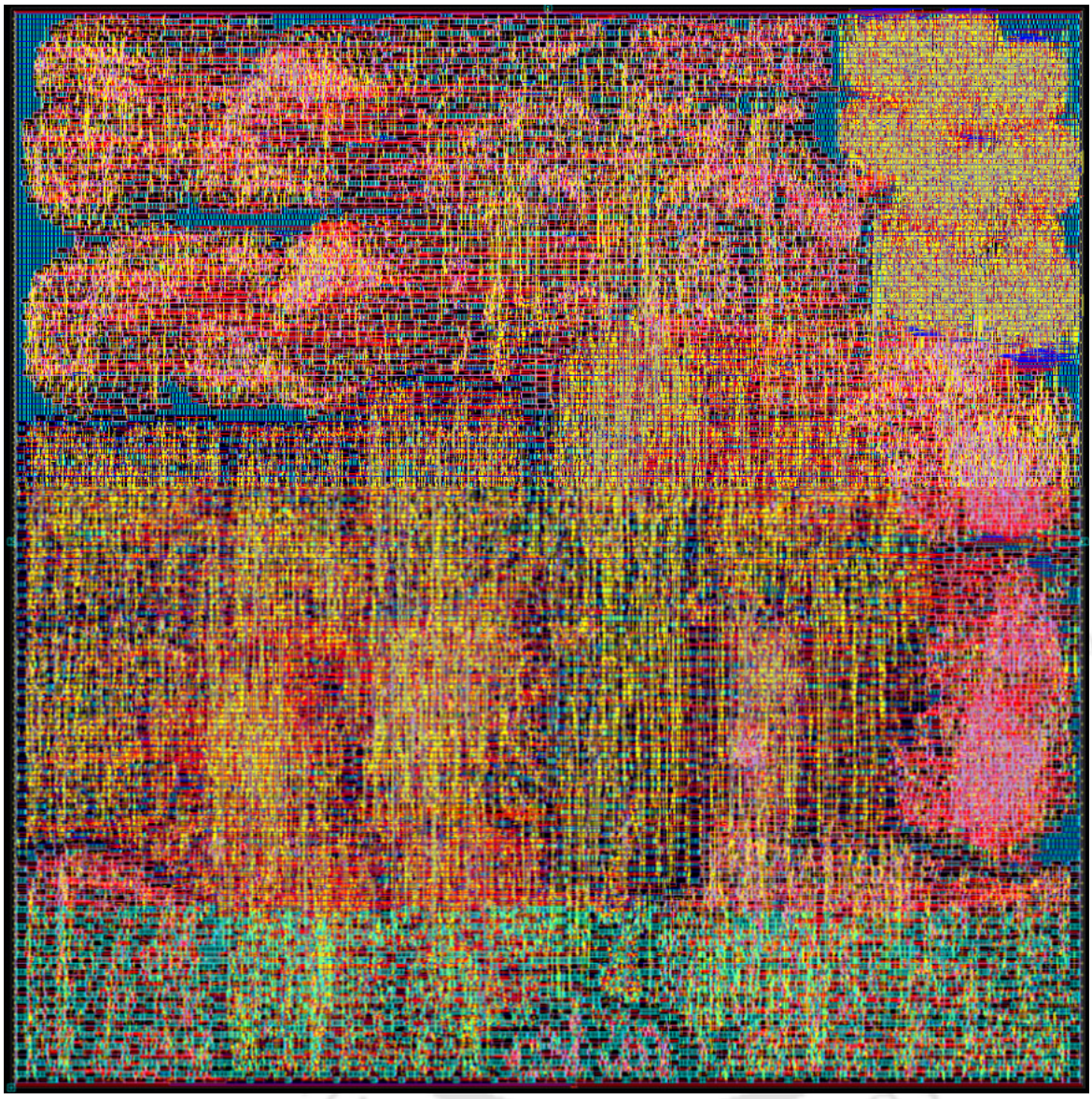


Figure 4.19: Chip layout at 180nm Bulk CMOS technology node

power. Below 100 KHz, leakage power becomes comparable to the dynamic power, which is not recommended for reliable circuit performance at 180nm technology. Above 100 KHz, leakage power is not significant as compared to static and dynamic power. It is to further mention that the leakage power does not rise substantially at 500 MHz as well. This implies that our proposed design can be used reliably at more than 100 KHz to high operating frequencies at 180-nm technology node. With the smaller technology nodes, the operating range is anticipated to be different.

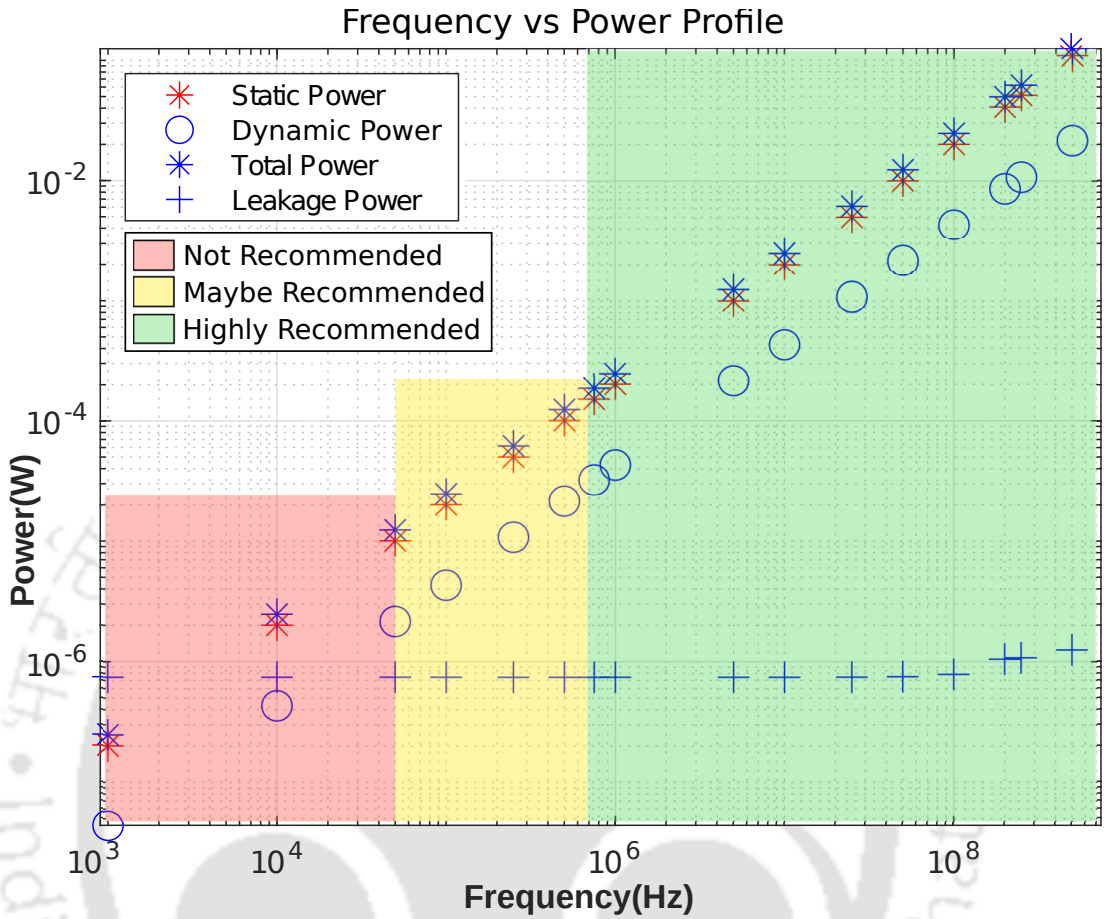


Figure 4.20: Power profile of the proposed design at various operating frequencies (1KHz to 500MHz)

4.6.2 Chaos Analysis for Security Estimation

In order to test randomness of a chaotic map generated in the proposed method, various statistical tests are performed. These statistical tests are generally employed to validate randomness in pseudo random-number generators. As we know, these chaotic maps are used to generate keys for encryption and decryption schemes. Therefore, the statistical tests can ensure whether the keys generated in our proposed scheme are ideally indistinguishable from any true random sequence or not. The statistical tests, which are conducted to support our claims, are described below.

Test for Randomness

NIST suite [130] is employed to test randomness properties of a chaotic map. The information about tests performed on the chaotic map generated during analysis and the corresponding results obtained are presented in Table 4.7, respectively. In the NIST test, our proposed chaotic generator

is compared with two recently published methods described in [195] and [196] to test its randomness. In [195], a sine chaotification model is employed to generate 1-D chaotic map, whereas, [196] presents physically unclonable functions (PUF) based chaotic maps. The [195] and [196] are used so that our proposed method is compared fairly with widely used chaotic schemes. From Table 4.7, it can be concluded that the proposed chaotic map generator performs well among all the tests. This is largely due to nonlinear dynamics of chaotic map. The outcome of these tests indicate that the proposed chaotic map is indistinguishable with respect to any other random sequence generators depicted in [195] and [196].

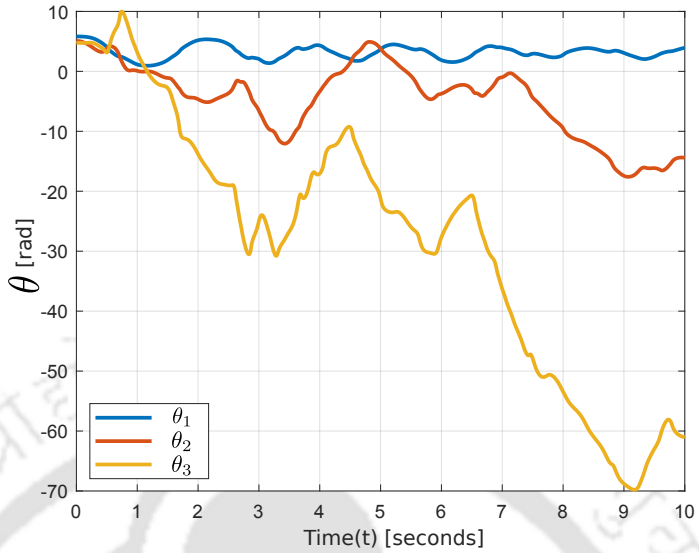
Table 4.7: Comparison of proposed scheme with other existing ones' on NIST $SP800-22$ test suite

NIST Tests	p-value ($0.01 \leq p\text{-}value < 1$)		
	Hua et al [195]	**Srisubha et al [196]**	**Proposed**
Frequency	0.364146	0.49	0.394808
Block Frequency	0.427274	0.33	0.783942
Cumulative Sums	0.670408	0.59	0.197764
Runs	0.213309	0.45	0.335584
Longest Run of 1's	0.204076	0.53	0.284067
Rank	0.819544	0.39	0.197764
DFT/Spectral	0.407091	0.41	0.769057
Non-periodic T.M.	0.472933	0.49	0.278074
Overlapping T.M.	0.602458	0.61	0.554566
Universal Statistical	0.772760	0.29	0.477911
Approx. Entropy	0.287306	0.60	0.365177
Random Excursions	0.442347	0.38	0.513954
Random Excursions Variant	0.356540	0.35	0.242148
Serial	0.175321	0.57	0.156848
Linear Complexity	0.264458	0.58	0.129930

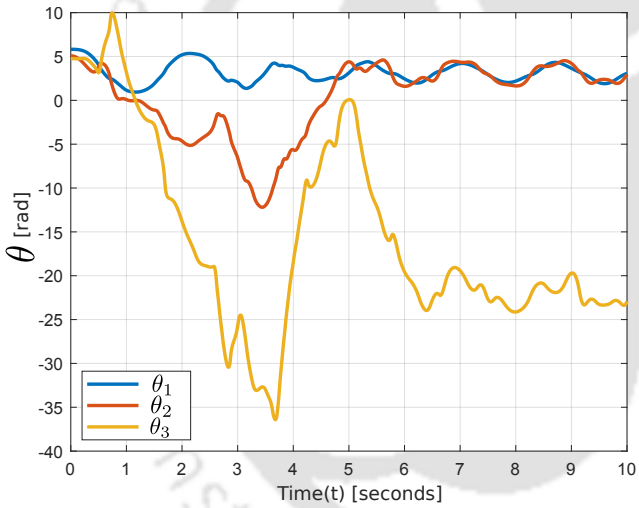
Test for Chaos

There are a few essential conditions, which need to be met by any chaos based cryptosystem. These conditions include ergodicity, sensitivity to parametric values and sensitivity to initial condition to be as unpredictable as possible. For completeness, these essential conditions are described below.

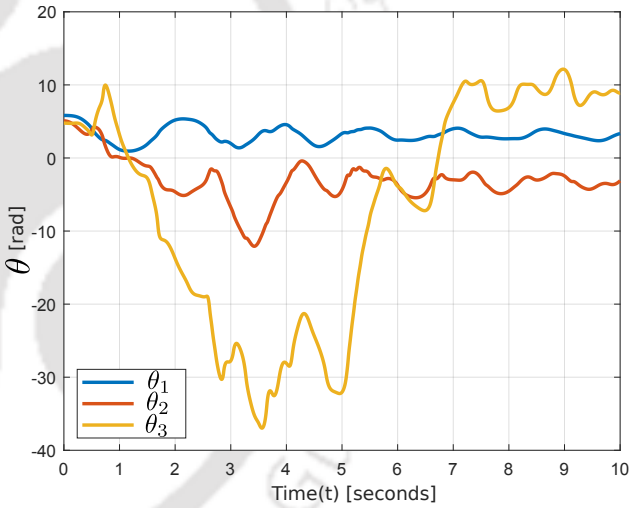
(i) **Sensitivity to Parametric values:** A small perturbation in one of the system parameters, for example m_1, is enough to create two exponential diverging trajectories starting at the same initial point. This is reflected in Figure 4.21(a) and Figure 4.21(b) for the parameters given in



(a) θ values for $m_1 = 0.2944$; $\theta_1(0) = 0.4603$



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



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

Figure 4.21: Sensitivity to parameter value m_1 and θ_1 with $\triangle m = 10^{-6}$ and $\triangle\theta_1 = 10^{-6}$

Table 4.1.

(ii) **Sensitivity to Initial Condition:** Two trajectories starting at two different but arbitrarily close initial points denoted by θ_1 diverge from each other at an exponential rate. This is illustrated in Figure 4.21(a) and Figure 4.21(c) for the initial conditions given in Table 4.1.

(iii) **Ergodicity:** Ergodicity is the phase space exploration either uniformly or randomly by a chaotic system. Here, phase space is defined as a relationship between angular velocity, ω, and the angle of pendulum's bob, ϕ. The triple pendulum system exhibits ergodic nature because phase space

is sparse and is not cluttered in any specific region. The phase space of the bottom bob of the proposed triple pendulum system illustrated in Figure 4.22 under conditions given in Table 4.1 exhibits a trajectory, which satisfies ergodicity criteria. This leads to a conclusion that a system under specific conditions can be considered ergodic.

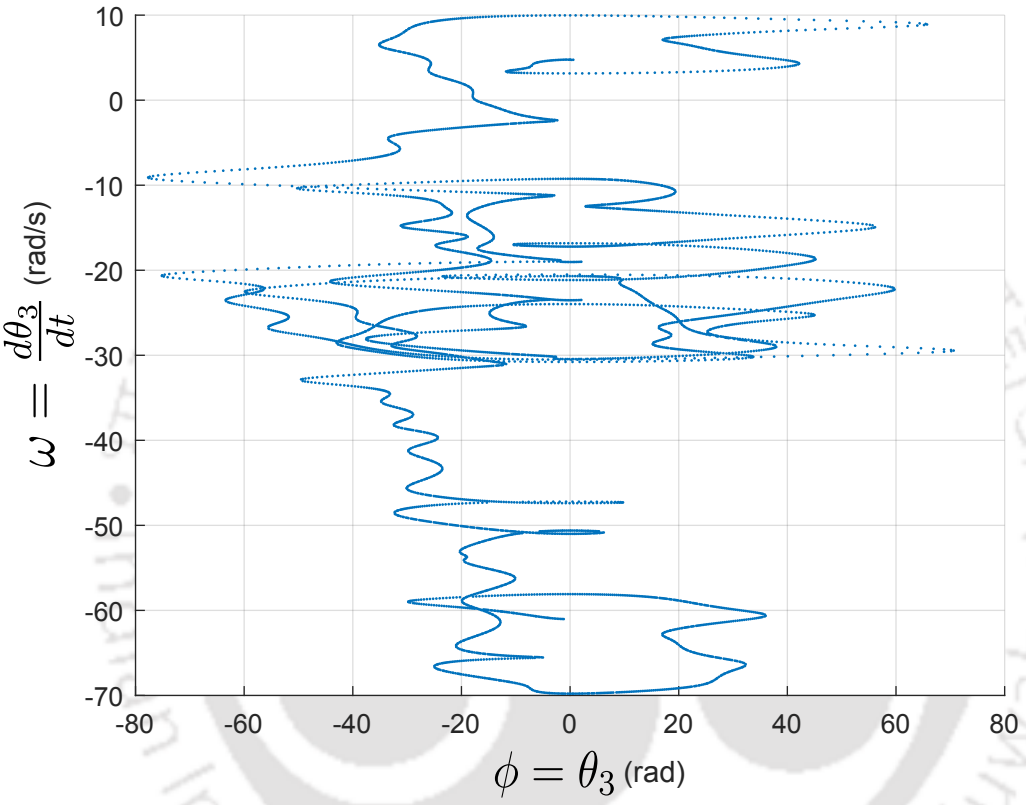


Figure 4.22: Phase space plot for ergodicity test for θ_3

The proposed triple pendulum based cryptosystem holds above mentioned properties, which are clearly evident in Figure 4.22 and Figure 4.21. It is to mention that the map generated using proposed compound triple pendulum model possesses essential chaotic properties to be employed in a general cryptosystem.

Collision Test

Collision resistance is the property of cryptographic algorithms, which makes it difficult to find two inputs that give the same encrypted output. It is well known that finding collisions is computationally very hard. In this chapter, it is demonstrated that the proposed algorithm is fully collision resistant. As it is discussed earlier, an entire range of $\hat{\mathbf{y}}$ are partitioned according to number of characters. For example, $N_c = 256$, i.e. $[\hat{y}_{min} + k\epsilon, \hat{y}_{min} + (k+1)\epsilon)$ for $k = 0, 1, ..., 255$, which are mapped to different

intervals. In the proposed method, two different inputs having different characters are mapped to different intervals always. Thus, it can be considered as collision resistant.

Based on the experimental and analytical results, it can be emphasized that the proposed scheme is better as compared to any other chaos based PQC algorithms. Because of smaller key size, optimal area, high throughput and low power consumption, it can also be used efficiently to enhance security of IoT devices as well as any low latency high performance secure application.

4.7 Conclusion

The work presented in this chapter discusses a novel symmetric key cryptography scheme based on a nonlinear triple pendulum chaotic map. The detailed analysis and standard tests of the proposed method classify it to be considered as a potential post quantum cryptography (PQC) algorithm. The proposed encryption scheme is compared with various other chaos based schemes over standard security parameters to prove its effectiveness. The highly nonlinear and interdependent unpredictable nature of our proposed scheme makes it difficult to breach the security without complete knowledge of its key. It is also unbreakable practically with the partial knowledge of its key because of high sensitivity of chaotic maps to key values. Design methodology and analytical proofs of the generator are presented here, such as bifurcation diagrams and Lyapunov exponent test to verify its chaotic property. The generated chaotic map is ratified as a cryptographically secure pseudo random number generator through various standard benchmark tests viz. sensitivity to parametric and to initial values, ergodicity, collision test, NIST randomness test etc. The proposed chaotic generator is utilized to develop a symmetric key encryption scheme and is realized on FPGA platform whose overall power consumption, delay and throughput are 0.186 W, 1.592 ns and 2396.164 MBps, respectively. The power consumption, resource utilization and throughput of the encryption scheme are 1.785×, 1.825× and 2.396× better when it is compared to the best known algorithm among the contemporary methods. The proposed cryptosystem is further implemented using 180nm Bulk CMOS technology occupying 0.20374 mm^2 die area and consuming 61.88 mW power at 250MHz at 180nm. It is shown that this system can operate from low (at least 100KHz) to high operating frequencies efficiently without showing any performance deterioration. The efficient area utilization of the proposed post quantum cryptosystem with high throughput at a very low power enables us to utilize it not only in the IoT based systems but also in the high performance applications to provide them desired strength against

any malicious attack.

5

Lattice Based Cryptosystem

Contents

Objective

Lattice based partially homomorphic encryption (PHE) schemes provide strong resistance against quantum as well as classical computer based adversary security attacks. Here, we present a software hardware co-design of two variants of an PHE scheme employing an ARM-SoC and an FPGA, which provides necessary acceleration to PHE methods in the above-mentioned ecosystem. First PHE variant is designed for generic homomorphic encryption, while second variant is aimed for resource optimal lightweight IoT driven applications. A robust and reliable low latency data transfer protocol is developed between FPGA based accelerator IP and ARM-SoC host system for their seamless assimilation. Moreover, a Linux based embedded operating system is customized with board support package (BSP), device-tree and first stage boot loader (FSBL) to enable all the essential functionalities required to communicate cross platform devices for performing encryption and decryption. The proposed PHE variants are realized using Verilog HDL on multiple FPGA platforms and are efficient alternatives to enhance security in edge enabled IoT devices.

5.1 Introduction to Lattice Homomorphic Cryptography

With the advent of Internet-of-Things (IoT), device to device communication, edge analytics etc., conventional systems are being transformed into data driven smart systems. Data generated using IoT enabled devices in a smart world is huge in nature and requires special techniques or devices to handle and process it. In the pool of peer-to-peer communicating devices, it is imperative to transfer data securely, otherwise the loss of vital information may affect data communication adversely. It is anticipated that the transformation to Internet-of-Everything (IoE) would increase complexity of the systems immensely and emphasis on the security of the systems would enhance greatly along with the need of real time data analytics. Further, the rise of quantum computing also makes the systems vulnerable. Thus, in order to address above mentioned issues, the need of the hour is to design area, form factor, speed and power efficient post quantum computing (PQC) secure systems. Here, we depict resource efficient realization of a hardware accelerator of a lattice based homomorphic algorithm on an FPGA.

The lattice based fully homomorphic encryption (FHE) scheme proposed by Gentry [78] present solutions for the security in the cloud computing and preserves data privacy against various malicious adversaries. FHE can perform computations over ciphertexts without decryption and reviling any

personal information, where one can evaluate mathematical functions on a ciphertext and modify it without decryption. If ψ is a mathematical function to be performed over plaintexts m_1 and m_2, then $\psi(m_1, m_2)$ can be calculated from their encrypted ciphertext ($Enc(m_1)$ and $Enc(m_2)$). Broadly there are two categories of homomorphic cryptography schemes, FHE and somewhat homomorphic encryption schemes. The main difference between them is that, FHE can evaluate a mathematical function with arbitrary complexity, while the latter one can only compute simple mathematical operations. The initial formulations of an FHE scheme is described in [78] and various similar classes of algorithms can be found in [197–204]. Most of the above mentioned algorithms are converted to FHE from somewhat homomorphic scheme by using bootstrapping for various key lengths. Thereafter, some alternative FHE schemes based on various hidden subspace membership (HSM) problems have been formulated without employing bootstrapping can be found in [205–208].

A well-known HSM problem, Learning with errors (LWE), based homomorphic encryption scheme is initially propose in [209, 210]. Here, the plaintext is recovered by finding the inner product of the ciphertext and a secret key. It is to mention that homomorphic multiplication of two ciphertexts are performed by their tensor product, and homomorphic key for the decryption of recrypted ciphertext is generated using an original secret key. In this process, the final ciphertext is larger as compared to input plaintext, and to reduce its size, relinearization is employed. Relinearization of ciphertext requires one more set of evaluation key consist of L matrices, where L is the circuit depth. Another limitation of the schemes proposed in [206,207] is the inclusion of noise while performing homomorphic multiplication, which can be addressed by introducing a modulus switching technique [211] in the above mentioned schemes.

An LWE-based FHE scheme is proposed in [208], which is realized by incorporating approximate eigen-vectors. In this scheme, relinearization is not required while performing homomorphic addition and multiplication. Therefore, this method do not require bootstrapping and L-depth evaluation keys, which reduce hardware and computational overheads. Further, ring-LWE homomorphic encryption schemes are developed [212, 213] modifying the above mentioned scheme, where bootstrapping is an optional feature only to enhance security. It is to mention that all these methods are compute intensive, thus, the need of the hour is to expedite their computations either by employing standard means or by a suitable custom approach. In order to address this issue, many hardware implementations of FHE schemes are presented in the literature, which are summarized below.

5.1.1 Brief Literature Review

In the last decade, a few hardware accelerators are proposed for FHE schemes, which can be found in [214–225]. A hardware implementation of Gentry-Halevi FHE algorithm [200, 201] using 90-nm CMOS technology is presented in [215], which takes 18.1 ms, 16.1 ms and 3.1 ms for encryption, decryption and recryption respectively. The architecture proposed in [215] consumes a large number of resources (around 30M gates) and takes more time to generate output. In [214], an FHE scheme utilizing iterative number theoretic transform (NTT) modular polynomial multiplier technique is implemented on an FPGA platform. Although the work presented in [214] is one of the latest works and depicts fast ring-LWE (RLWE) based FPGA implementation, but this work is not resource optimal, which makes it less suitable for low power security applications.

An accelerator for somewhat FHE (SFHE) proposed in [219] mainly focuses on the optimization of NTT base polynomial multiplier using Cooley-Tukey FFT technique, and not only consumes higher resources but also exhibits higher delay. A Brakerski/Fan-Vercauteren (BFV) FHE accelerator is realized in [197,216] on Xilinx Virtex-7 using PCIe interconnect protocol and simple encrypted arithmetic library (SEAL). In [216] implementation, there are three main components, i.e. Montgomery modular multiplier, SEAL based four step for encryption-decryption modules and iterative NTT operation module, to perform partial encryption-decryption. Thereafter, SEAL software is deployed to complete remaining operations of encryption and decryption, which makes the method resource intensive and become complex while hardware realization.

Further, integer multiplication architectures employing FFT and Low Hamming Weight (LHW) designs are proposed in [217] to gain significant speedup while implementing them on Xilinx Virtex-7 FPGA platform. In [217], partial modifications of FHE are proposed by realizing an integer-FFT multiplier with LHW parameters and a large number of DSP blocks and LUTs are required to implement this multiplier, which is not suitable for low power IoT applications. A 16 to 32-bit Torus fully homomorphic encryption (TFHE) scheme on CPU as well as on GPU is proposed in [218] to reduce computational time of data encryption and decryption. It is to mention that FHE method proposed in [218] is optimized for 16 to 32-bit multi-CPU cores and GPUs only. Thus, its hardware realization is complex and resource intensive. A general NTRU like lattice FHE scheme designed using AXI-interconnect is presented in [220], and an FFT/iFFT based method is depicted in [221]. The work presented in [220] proposes to use this scheme on generic high level FPGAs theoretically.

In [221] feasibility of various multiplication methods on Virtex-7 class FPGAs for FHE is delineated and it is observed that it requires huge number of DSP blocks and LUTs for its realization.

An integer multiplier designed using NTT and inverse-NTT (INTT) based on radix-r butterfly units is stated in [222] to implement FHE algorithm. The above method design works for of large integer multiplications and it manipulates data by BRAM and SRAM memory operations. It is implemented using 90-nm CMOS technology, which spans substantial area and consumes more cycles because of repeated memory access. This makes it infeasible for low power IoT devices. A hardware accelerator implementation of integer-based FHE utilizing Comba multiplication technique is presented in [223], which exhibits an architecture only for the encryption and takes 159 seconds to encrypt a large integer input using the proposed technique. It is also infeasible to implement decryption and homomorphic operations using the above mentioned method. Thus, the proposed method cannot be employed in the realization of real time IoT applications.

5.1.2 Motivation and Overview of the LWE PHE Accelerator Design

From the above mentioned works it can be observed that the hardware implementations consume large resources, which are not suitable for low power IoT system design. Delay, area and power are significantly higher in most of these methods, which are the limiting factors in the real-time applications. Thus, there is a need of low latency resource optimal methodology for encryption and decryption for IoT applications. Further, as per best of our knowledge, there is no generic methodology found in the literature to integrate PHE hardware accelerator IP with an ARM-SoC in software-hardware co-design. Thus, in the proposed work, the primary focus is to develop a simple lightweight LWE based PHE accelerator IP core on FPGA platform, which consumes lower resource and exhibits less delay with significant throughput in the real-time operations.

Here in this chapter we develop an LWE PHE algorithm and optimally realized on FPGA platform. In this regard, two variants of PHE hardware accelerators are proposed based on LWE-HSM problem, which make the system secure against classical as well as quantum computer based attacks. Although, both the designs are resource optimized, the first variant aims more on the security aspect, while the second variant is designed mainly for high-speed low-power IoT applications. The other notable contribution here is to develop generic custom AXI slave IP for seamless integration of programmable system (PS) and custom FPGA core design. Moreover, an embedded Linux kernel is customized, which is hosted on an ARM processor to facilitate easy integration of cross platform communication

devices via standard Ethernet protocol, efficient memory management etc.

In this chapter first the preliminaries required to describe the proposed PHE algorithm is explained along with its mathematical formulations. Thereafter resource optimal FPGA implementation details of both the variants of the LWE PHE algorithms and their mathematical formulations are elaborated. Later a detailed ARM based Zynq-SoC setup, custom embedded operating system and integration of FPGA accelerator with the ARM-host are explained. Lastly, discussion of the results obtained during analysis of the proposed scheme and its comparison with already existing contemporary hardware implementations are reported.

5.2 Formulation of First Variant of LWE PHE Algorithm

Various notations used for the description of algorithms are presented in the acronym section of the thesis for ceaseless navigation through the chapter and reference to mathematical terms. Also for easy understanding of the proposed method, few basic definitions are presented below for completeness.

Learning With Errors: For $\{p, n\} \in \mathbb{N}$, the probability distribution is $\mathcal{X}$ over $\mathbb{Z}$ and $\mathbf{s} \in \mathbb{Z}_p^n$ is a secret vector from $\mathbf{a}_i \xleftarrow{\$} \mathbb{Z}_p^n$, $e_i \xleftarrow{\$} \mathcal{X}$. If a given polynomial times many samples of the form $(\mathbf{a}_i, \mathbf{b}_i) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$, where $\mathbf{b}_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \pmod p$, then $LWE_{n,p,\mathcal{X}}$ problem is to determine an output vector $\mathbf{s} \in \mathbb{Z}_p^n$ with overwhelming probability.

Hidden Subspace Membership: For positive integers $n, q, \ell \in \mathbb{N}$, where q is prime and $\ell \geq 1$, let $\mathcal{S}$ be an ℓ-dimensional subspace of the vector space $\mathcal{V} := \mathbb{Z}_p^n$. Let $\mathcal{N} = \{0, \mathcal{X}^{n-\ell}\}$ be a noise distribution on $\mathcal{V}$, where $\mathcal{X} = \mathcal{X}_\alpha$ is zero mean distribution on $\mathbb{Z}_p$ and 0 is all zero vector of size ℓ. Further, the HSM problem denoted by $HSM_{\ell,n,q,\mathcal{N}}$ can be defined by equation 5.1, where a probabilistic polynomial time adversary (PPT adversary) $\mathcal{A}$ can guess the value of $c \in \{0, 1\}$ with a significant probability.

$$Adv_{\ell,n,q,\mathcal{N}}^{HSM,\mathcal{A}}(\lambda) := |Pr[HSM_{\ell,n,q,\mathcal{N}}^{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \tag{5.1}$$

The $LWE_{n,p,\mathcal{X}}$ is a specific case of HSM problem, namely, $HSM_{n+1,n,p,\mathcal{N}}$, where $\mathcal{N} = (\mathbf{0}, \mathcal{X})$. Therefore, if an adversary is able to solve $HSM_{l,n,p,\mathcal{N}}$ problem, it implies that same strategy can be employed to solve $LWE_{n,p,\mathcal{N}}$ problem.

The security of LWE based encryption schemes can be assured by its equivalence in mathematical hardness with the shortest vector finding problem in lattice architectures. Lets assume a discretized LWE based PHE, where $\mathcal{X}$ is mean noise distribution profile with a Gaussian distribution with $\sigma \geq$

$2\sqrt{n}$ (n is dimension of the field). Therefore, there certainly exists a quantum reduction method to solve an LWE problem, which approximates to decisional Shortest Vector Problem (GapSVP$_\gamma$) [226, 227], where approximation factor is denoted by γ. Further, [228] and [229] exhibit classical reductions of LWE problem from the worst-case lattice problems using an exponential modulus and a polynomial modulus, respectively. The time complexity of best known algorithms for GapSVP$_\gamma$ problem is $2^{\Omega(n/log\gamma)}$ [230, 231] to find any solution with non-negligible probability. Therefore, it can be understood that the encryption algorithms based on LWE problems are secure against both quantum as well as classical computer generated cryptographic attacks, because these attacks consume exponential time to analyze the network. The following subsection describes key generation, which is the most important part of an PHE scheme followed by encryption, decryption and recryption operations. It is to mention that the proposed scheme is derived from [232, 233].

5.2.1 Key Generation

Lets consider p to be an odd prime number and $\mathcal{I}$ is an ideal of the polynomial ring $\mathbb{F}_p[x_1, \ldots, x_l]$. Here, $\mathbb{F}_p[x_1, \ldots, x_l]_r$ is a vector space with dimension $D=\binom{l+r}{r}$ for $r \in \mathbb{N}$. $\mathcal{I}_{\leq r}$ represents set of a polynomial in $\mathcal{I}$ with degree $\leq r$, such that $\mathcal{I}_{\leq r}$ is a d dimensional subspace of $\mathbb{F}_p[x_1,....,x_l]_r$. $\mathcal{A}$ denotes the set of distinct points $[a_1, ..., a_n]$ in $\mathbb{F}_p^l$, such that $d \leq n \leq D$. The polynomials $f \in \mathbb{F}_p[x_1, \ldots, x_l]_r$ from the given basis $\mathcal{B}_{\mathcal{I}_{\leq r}}=(f^1, \ldots, f^d)$ of subspace $\mathcal{I}_{\leq r}$ are evaluated at points in set $\mathcal{A}$ to get the basis $\mathbf{B}$ for $\mathcal{S}_{\mathcal{I}_{\leq r}}$, where matrix $\mathbf{B} \in \mathbb{Z}_p^{n \times l}$. Following Lemma 1, constraints of set $\mathcal{A}$ can be laid out. It can be seen that a basis for the perpendicular subspace $(\mathcal{S}_{\mathcal{I}_{\leq r}})^\perp$ has a specific form in equation 5.1. The numerical method to obtain basis matrix for the subspace $(\mathcal{S}_{\mathcal{I}_{\leq r}})^\perp$ is described in Algorithm 12. It is to mention that basis matrix is an essential part of the secret key matrix ($\mathbf{s}^{HSM}$).

Lemma 1:

The $HSM_{\ell,n,q,\mathcal{N}}$ problem for an ℓ-dimensional subspace $\mathcal{S}$ can be reduced to $LWE_{\ell,q,\mathcal{X}}$ problem with multiple secrets $\{S_i\}_{1 \leq i \leq n-\ell}$. A basis for the subspace $\mathcal{S}$ can be formulated in terms of a matrix as equation 5.2.

$$B = [\frac{B_1}{B_2}] \in \mathbb{Z}_q^{n \times \ell} \tag{5.2}$$

Here, $B_1 \in \mathbb{Z}_q^{\ell \times \ell}$ is a full rank matrix and every row of $B_2 \in \mathbb{Z}_q^{(n-\ell) \times \ell}$ is a linear combination of the rows of B_1. Then, for $v = (v_1, v_2) \in \mathcal{S}^\perp$, the equation 5.3 can be depicted.

Algorithm 11: keyGen (B,n,d)

Input: **B**=Basis matrix of subspace $\mathcal{S}_{\mathcal{I}_{\leq r}}$,
$n = |\mathbf{A}|$, d=dimension $(\mathcal{S}_{\mathcal{I}_{\leq r}})$.
Output: Matrix $\mathbf{s}^{HSM} \in \mathbb{Z}_p^{(n-d)\times n}$
set $\mathbf{I}_{n-d}$ = Identity matrix of size $(n-d)$
set $\mathbf{B1} \in \mathbb{Z}_p^{n\times n}$ and $\mathbf{B2} \in \mathbb{Z}_p^{(n-l)\times n}$, such that $\mathbf{B}= \left[\begin{smallmatrix} \mathbf{B1} \\ \mathbf{-B2} \end{smallmatrix} \right]$
set integer $i = 1$
repeat for $1 \leq i \leq (n-d)$
  begin
    $\mathbf{v}_1 = \mathbf{I}_{n-d}$(i,:)*$\mathbf{B2}$
    $\mathbf{v}_2 = (\mathbf{B1}^{-1})$*$\mathbf{v}_1$
    $\mathbf{s}^{HSM}$(i,:)= $[\mathbf{v}_2 \mid \mathbf{v}_1]$
  end

$$v_1 B_1 = -v_2 B_2 \tag{5.3}$$

This implies that for a given $v_2 \xleftarrow{\mathcal{S}} \mathbb{Z}_q^{n-\ell}$, we can find $v_1 \in \mathbb{Z}_q^{\ell}$, such that equation 5.3 is satisfied. Therefore, we can find a basis $\{\mathcal{S}'_{\ell+1}, \ldots, \mathcal{S}'_n\}$ for $\mathcal{S}^{\perp}$, such that

$$\mathcal{S}'_{\ell+1} = (\mathcal{S}^1_{\ell+1}, \ldots, \mathcal{S}^{\ell}_{\ell+1}, 1, 0, \ldots, 0)^T,$$

$$\mathcal{S}'_{\ell+2} = (\mathcal{S}^1_{\ell+2}, \ldots, \mathcal{S}^{\ell}_{\ell+2}, 0, 1, \ldots, 0)^T,$$

$$\vdots$$

$$\mathcal{S}'_n = (\mathcal{S}^1_n, \ldots, \mathcal{S}^{\ell}_n, 0, 0, \ldots, 1)^T$$

for $\mathcal{S}^i_j \in \mathbb{Z}_q$ and, $1 \leq i \leq \ell$ and $\ell+1 \leq j \leq n$. Given a sample $v = (v_1, \ldots, v_\ell, v_{\ell+1}+e_{\ell+1}, \ldots, v_n+e_n) \in \{\mathcal{S}+\mathcal{N}\}$ for $\ell+1 \leq j \leq n$, the following expressed by equation 5.4 can be met.

$$\langle V, \mathcal{S}'_j \rangle = \sum_{i=1}^{\ell} V_i \mathcal{S}^i_j - V_j + e_j \approx 0 \pmod q \tag{5.4}$$

where, $\sum_{i=1}^{\ell} V_i \mathcal{S}^i_j - V_j = 0$, hence, $V_j = \sum_{i=1}^{\ell} V_i \mathcal{S}^i_j$. Thus, a query to the sample oracle of LWE problem with multiple secrets $\{S_i\}_{1\leq i\leq n-\ell}$ can be answered with $(a, b_1, \ldots, b_{n-\ell}) \in \mathbb{Z}_q^{\ell+1}$, where $a = (V_1, \ldots, V_\ell)$ and $b_i = V_{\ell+i}$ for $1 \leq i \leq n-\ell$. It is to mention that LWE secret $\mathcal{S}_i$ is the vector $\mathcal{S}_i = (\mathcal{S}^1_{\ell+i}, \ldots, \mathcal{S}^{\ell}_{\ell+i})$ for $1 \leq i \leq n-\ell$.

If $\mathbf{0}_{(n-d)\times d}$ denotes a zero matrix of dimension $(n-d) \times d$ and **M1** be a matrix of dimension

$d \times (n-d)$, such that the elements of **M1** are sampled randomly from $\mathbb{Z}_p$, then the matrix $\mathbf{M} \in \mathbb{Z}_p^{n \times n}$ is formulated as given below.

$$\mathbf{M} = \left[\begin{array}{c|c} \mathbf{I}_d & \mathbf{M1} \\ \hline \mathbf{0}_{(n-d)\times d} & \mathbf{I}_{(n-d)} \end{array}\right] \tag{5.5}$$

If $\mathbf{M}^{-1}$ denotes inverse matrix of $\mathbf{M}$, then $\mathbf{M}^{-1}$ can be represented in the following manner mentioned below.

$$\mathbf{M}^{-1} = \left[\begin{array}{c|c} \mathbf{I}_d & \mathbf{\text{-}M1} \\ \hline \mathbf{0}_{(n-d)\times d} & \mathbf{I}_{(n-d)} \end{array}\right] \tag{5.6}$$

For the overall scheme, ideal $\mathcal{I}$, matrix $\mathbf{s}^{HSM}$, matrix **M1** and set of point $\mathcal{A}$ together form the secret key, whereas, d, l and p are the public parameters of the key.

5.2.2 Encryption

Algorithm 11 describes the steps of proposed encryption scheme. It inputs a message vector $\mathbf{m}$ and secret key *sk*, and converts it into a ciphertext matrix $\mathbf{C}$. For the encryption of a message bit $m \in \{0, 1\}$, a polynomial f of degree $\leq r$ is sampled uniformly from $\mathcal{I}$. This polynomial is evaluated at a point in $\mathcal{A}$ to obtain $f(\mathcal{A})$. Matrix $\mathbf{F}$ is formulated using random polynomials f at all the point in $\mathcal{A}$. Noise vector $\mathbf{e}$ is of the order l, where all the non-zero elements of $\mathbf{e}$ are constituted using uniform samples taken from discrete Gaussian distribution χ with zero mean and standard deviation $\sigma = \alpha p$, for some real $\alpha > 0$. For each message bit m, a ciphertext vector $\mathbf{c} \in \mathbb{F}_p^l$ is calculated employing equation 5.7. Its computational complexity is depicted in section 5.3.

$$\mathbf{c} = \{(m + f(\mathcal{A}) + \mathbf{e})\ \mathbf{M}\}\ mod\ p \tag{5.7}$$

5.2.3 Decryption

Algorithm 10 describes the scheme of recovering message vector $\mathbf{m}$ from ciphertext matrix $\mathbf{C}$ employing equation 5.8 given below. It also takes secret key *sk* as an input and generates $\mathbf{m}$. Computational complexity of this method is described in section 5.3.

$$\mathrm{m} = \left\lfloor \frac{1}{\lfloor \frac{p}{2} \rfloor} ((\langle \mathbf{s}^{HSM}, \mathbf{c} \rangle) \bmod p) \right\rceil \bmod 2 \tag{5.8}$$

Algorithm 12: Encrypt(m,sk)

Input: Message vector $\mathbf{m}$,
Secret key (Matrix $\mathbf{M1}$ and $\mathbf{F}$)
Output: ciphertext $\mathbf{C} = \{\mathbf{c}^i \in \mathbb{F}_p^l \mid 1 \leq i \leq l\text{-}d\ \}$
begin
  Sample polynomial $f \xleftarrow{\$} \mathcal{I}$, with degree$(f) \leq r$
  Evaluate f on $\mathcal{A}$: $f(\mathcal{A}) = \{f(a_1), ..., f(a_l)\}^T$
  Form vector $\mathbf{m}(i) = ((\mathbf{0}, \mathbf{m'}) \ . \ \lfloor \frac{p}{2} \rfloor)^T$, with $order(\mathbf{0}) = d$ and $order(\mathbf{m'}) = l - d$
  Noise $\mathbf{e}$=$(\mathbf{0}, \mathbf{e'})$, with $order(\mathbf{0}) = d$, $order(\mathbf{e'}) = l - d$, and $\forall e_i \in \mathbf{e'}$, $e(i) \xleftarrow{\$} \chi_\alpha$
  Ciphertext $\mathbf{C}(i,:) = \mathbf{c}^i$={($\mathbf{m}^i$ + f($\mathcal{A}$) + $\mathbf{e}$) $\mathbf{M}$} mod p
  repeat for $1 \leq i \leq l - d$
  end

Algorithm 13: Decrypt(C,sk)

Input: Ciphertext $\mathbf{C} = \mathbf{c}^i{}_{1 \leq i \leq l\ -\ d}$,
Secret key $\mathbf{s}^{HSM}$
Output: Plaintext $\mathbf{m}$={ $m_i \mid m_i \in \{0,1\}$, $1 \leq i \leq l - d$ }
begin
  Calculate $\mathbf{M}^{-1}\ \mathbf{c}^i$
  Calculate inner product: $x = \langle\ \mathbf{s}^{HSM}, \mathbf{M}^{-1}\ \mathbf{c}^i\ \rangle$
  Apply modular reduction: $x = (x \bmod p)$
  Message $m^i = \left\lfloor\ \frac{x}{\lfloor \frac{p}{2} \rfloor}\ \right\rceil \bmod 2$
  repeat for $1 \leq i \leq l - d$
  end

5.2.4 Recryption

As we know, arithmetic operations on encrypted data is known as recryption and it is applicable only on the schemes exhibiting homomorphism. The validation of homomorphic properties to exhibit recryption in the proposed PHE is given here. In the proposed PHE, arithmetic operations, such as addition, can be applied directly on the ciphertext, which translates further to the addition of underlying encrypted message bits. This helps in preserving security of message plaintext in homomorphic encryption scheme. It can be proved that the proposed algorithm is homomorphic with respect to the arithmetic operations and it is given below. For ciphertexts $\mathbf{c}_1$=({(m_1 + f_1($\mathcal{A}$) + $\mathbf{e}_1$) $\mathbf{M}$} mod p) and $\mathbf{c}_2$=({(m_2 + f_2($\mathcal{A}$) + $\mathbf{e}_2$) $\mathbf{M}$} mod p), $\mathbf{c}_{add}$ is expressed in equation5.9.

$$\begin{aligned}\mathbf{c}_{add} &= \mathbf{c}_1 + \mathbf{c}_2 \\ &= \{(m_1 + f_1(\mathcal{A}) + \mathbf{e}_1)\mathbf{M}\} \bmod p + \{(m_1 + f_2(\mathcal{A}) + \mathbf{e}_1)\mathbf{M}\} \bmod p \\ &= ((m_1 + m_2) + (f_1(\mathcal{A}) + f_2(\mathcal{A})) + (\mathbf{e}_1 + \mathbf{e}_2))\mathbf{M} \bmod p\end{aligned} \tag{5.9}$$

Here, $(f_1(\mathcal{A}) + f_2(\mathcal{A})) \in \mathcal{S}_{\mathcal{I}_{\leq r}}$ and the secret vector $\mathbf{s}^{HSM} \in \mathcal{S}^{\perp}_{\mathcal{I}_{\leq r}}$, thus, $\langle \mathbf{s}^{HSM}, (f_1(\mathcal{A}) + f_2(\mathcal{A}))\rangle = 0 \pmod p$. If $\mathbf{e}_1(j)$ and $\mathbf{e}_2(j)$ represent j^{th} entries of noise vectors $\mathbf{e}_1$ and $\mathbf{e}_2$, respectively, for $d+1 \leq j \leq n$, then decryption of $\mathbf{c}_{add}$ can be expressed by equation 5.10.

$$\begin{aligned}\langle \mathbf{s}^{HSM}, M^{-1}\mathbf{c}_{add}\rangle &= (\mathbf{m}_1 + \mathbf{m}_2).\lfloor\frac{p}{2}\rfloor + (\mathbf{e}_1(j) + \mathbf{e}_2(j))(\text{mod } p) \\ &= (\mathbf{m}_1 \oplus \mathbf{m}_2).\lfloor\frac{p}{2}\rfloor - \frac{1}{2}[(\mathbf{m}_1 + \mathbf{m}_2) - (\mathbf{m}_1 \oplus \mathbf{m}_2)] + (\mathbf{e}_1(j) + \mathbf{e}_2(j))(\text{mod } p)\end{aligned} \tag{5.10}$$

If $\mathbf{e}_{add} = -\frac{1}{2}(\mathbf{m}_1 + \mathbf{m}_2) - (\mathbf{m}_1 \oplus \mathbf{m}_2) + (\mathbf{e}_1(j) + \mathbf{e}_2(j)) \bmod p$, then Decrypt$(c_{add}, sk)$ recovers the message sum bit, if additive noise is bounded, i.e. $\mathbf{e}_{add} < 1 + 2B$, where $B < \lfloor p/2 \rfloor/2$.

It is to mention that Sage-Math [234] is employed for the verification and analysis of the proposed scheme. Its block level overview is shown in Fig. 5.1. A prime number p, degree of polynomial ideal $\mathcal{I}$ r, number of polynomial variables l and number of input message bits n are given as input to the encryption block. The proposed design is tested for a set of input parameters, for example p=131, l=4, r=2, n=10 and message bits={1,0,1,0,1}. Details of the hardware implementation of the proposed encryption, decryption and recryption schemes are given in the subsequent sections.

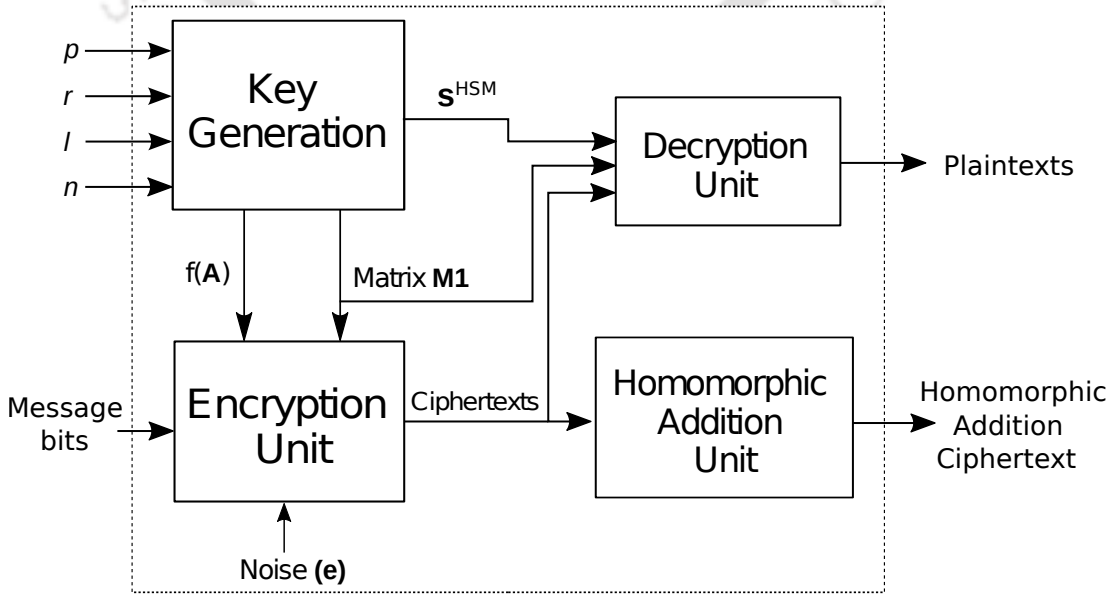


Figure 5.1: Proposed Software Flow of the PHE Scheme

5.3 Hardware Implementation of first PHE Scheme

The proposed design is implemented in Register-Transfer Level (RTL) using System Verilog HDL and Xilinx Vivado is employed for the logic synthesis. All the major functional blocks of the proposed PHE are presented below.

5.3.1 Encryption Unit

The encryption module is designed using a finite state machine (FSM), in which input message bits are processed bitwise. It can be seen from equation 5.11 that $\mathbf{M}$ is generated using $\mathbf{M1}$, $\mathbf{I}_n$, and $\mathbf{0}_n$. Storing the entire $\mathbf{M}$ matrix requires BRAM of depth $n \times n$. However, the structure of $\mathbf{M}$ can be used to optimize input memory requirements. The main information in $\mathbf{M}$ lies in the sub-matrix $\mathbf{M1}$. Thus, the storage is optimized by storing only $\mathbf{M1}$ as an input. It requires only $d \times (n-d)$ BRAM depth, which reduces memory footprint by $n(n-d)+d^2$. Further, the matrix-vector multiplication is optimized by exploiting the sparsity of $\mathbf{M}$. For each input message bit, using the rows of secret key matrix $\mathbf{F}$, $f(\mathcal{A})$ is generated as stated in the previous section and then added with noise vector $\mathbf{e}$. Later, this vector is multiplied with $\mathbf{M}$ to generate the final ciphertext. Note that while performing these operations, the dimension of ciphertext becomes more ($\geq n$) than the individual dimensions of matrices $\mathbf{F}$, $\mathbf{M}$ and $\mathbf{E}$. Here, $\mathbf{E}$ is the noise matrix. Therefore, a modular operation is performed using Algorithm 2 to bind the output in the range. Here x, p, and x_{mod_p} are the input, modularity, and output. The modular operation ensures that the elements of ciphertext are within $\lfloor -\frac{p}{2} \rfloor$ to $\lfloor \frac{p}{2} \rfloor$, which leads to the generation of an n-dimensional cipher vector by the encryption unit.

$$\mathbf{M} = \left[\begin{array}{c|c} \mathbf{I}_d & \mathbf{M1} \\ \hline \mathbf{0}_{(n-d)\times d} & \mathbf{I}_{(n-d)} \end{array}\right] \tag{5.11}$$

The main arithmetic computation in encryption is depicted by step 6 in Algorithm 11. Its hardware implementation is performed in three stages, which are vector-vector addition ($temp_{vec} = m_i + F[i;0] + e$), vector-matrix multiplication ($temp_{cipher} = temp_{vec} \times M$), and modular operation ($C[i;0] = temp_{cipher}$ **mod** p). Figure 5.2 presents an overall architecture of the encryption module, which contains basic arithmetic units, a matrix multiplier, and a modular function unit. Note that the term in square brackets can either be an individual element, an entire column, or a row of a particular matrix implemented on the hardware. Parentheses are used during algorithmic descriptions

Algorithm 14: Modular Operation

Input: x, p
Output: x_{mod_p}
procedure BEGIN
  $x_{temp} = |x|$
  while $(x_{temp} > p)$ **do** $(x_{temp} = x_{temp} - p)$
  if $(x < 0)$ **then** $(x_{temp} = p - x_{temp})$
  if $(x_{temp} > \lfloor \frac{p}{2} \rfloor)$ **then** $(x_{temp} = x_{temp} - p)$
  return $x_{mod_p} = x_{temp}$
  END
end procedure

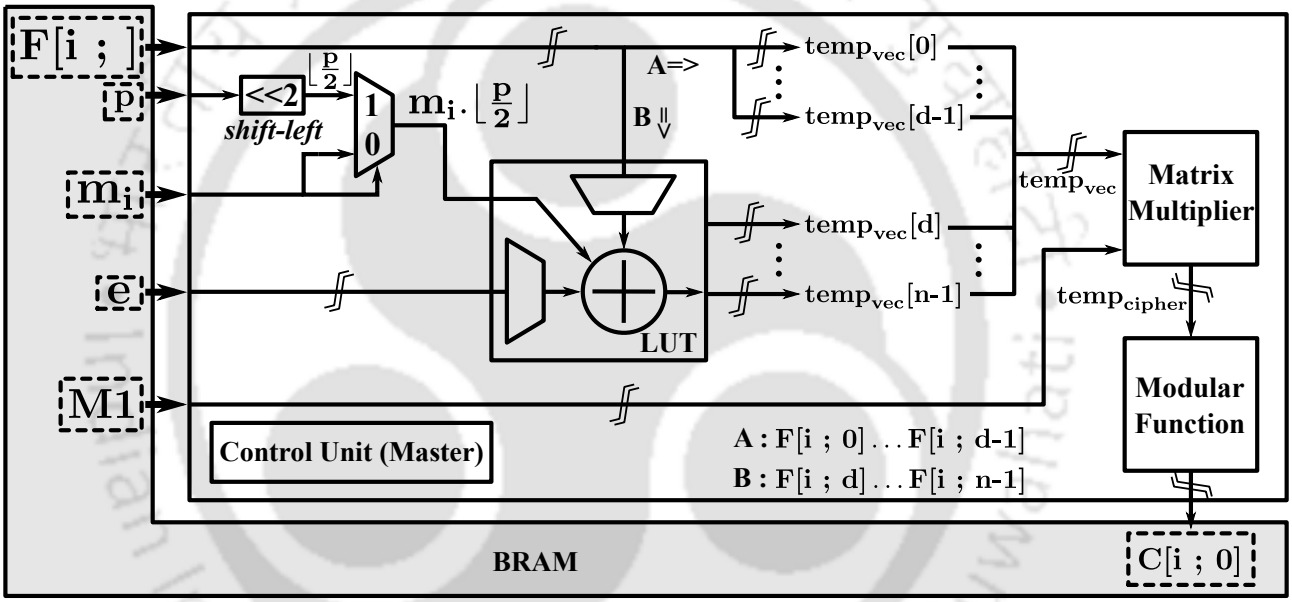


Figure 5.2: Overall Encryption Architecture of the First Scheme

and mathematical formulations, whereas square brackets are employed while describing hardware implementations.

Vector-Vector Addition

For optimizing memory storage, only nonzero elements of vector m_i and e are stored in BRAM, which reduces input BRAM depth by $n-1+d$. From Figure 5.3, the following relation can be derived by observing the structure of vectors $m(i)$, $F[i;]$, and e.

$$temp_{vec} = \begin{cases} F[i;k]; & 0 \leq k < d; \\ m_i \times \lfloor \frac{p}{2} \rfloor + F[i;k] + e'[k-d]; & d \leq k \leq (n-1-d) \end{cases}$$

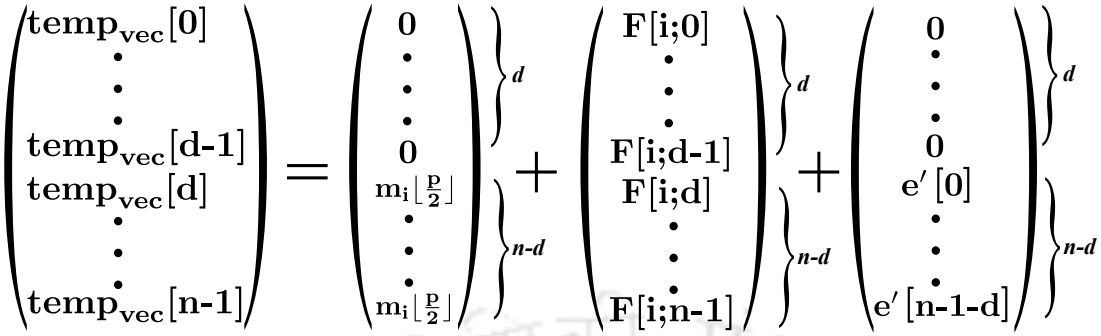


Figure 5.3: Vector Addition Operation

As we know, m_i is a single bit, and $m_i \times \lfloor \frac{p}{2} \rfloor$ is a basic MUX operation; vector addition is implemented using carry look-ahead adders available with DSP Slices in an FPGA.

Vector Matrix Multiplier Module

Here, the following equation represents vector-matrix multiplication. From matrix **M**, it can be deduced, $temp_{cipher}[k] = temp_{vec} \times [I_d[;k] \mid 0_{(n-d)\times d}[;k]]$ for $0 \leq k < d$. Here $\mid$ is a concatenation operator. This computation is optimized by introducing an identity matrix; therefore, no actual multiplication is required to compute $temp_{cipher}[k]$. Thus, $temp_{cipher}[k] = temp_{vec}[k]$ for $0 \leq k < d$ saves $(n \times d)$ multiplication and $(n-1) \times d$ addition operations.

$$[temp_{vec}[0] \ldots temp_{vec}[n-1]]_{1\times n} \times \left[\begin{array}{c|c} \mathbf{I}_d & \mathbf{M1}_{d\times(n-d)} \\ \hline \mathbf{0}_{(n-d)\times d} & \mathbf{I}_{(n-d)} \end{array}\right]_{n\times n}$$

Subsequently, to compute $temp_{cipher}[k] = temp_{vec} \times [M1_{(d\times(n-d)}[;k] \mid I(n-d)[;k]]$ for $d \leq k < n$, the MAC DSP unit of FPGA is employed. The architecture of this multiplier is depicted in Figure 5.4. Vector-matrix multiplication is implemented by choosing corresponding elements of $temp_{vec}$ and $I_{(n-d)}$.

Modular Operation

Here, the modular operation is described using modular function flow and its bitwise architecture to provide complete details of the architecture. Note that a generic modular block is realized instead of Montgomery or Barret reduction [235, 236].

Modular Function Flow: Figure 5.5 depicts the flow diagram of modular function implementation. Here, x and y are 32-bit input and output, such that $y = x \ mod \ p$. Internal one-bit register

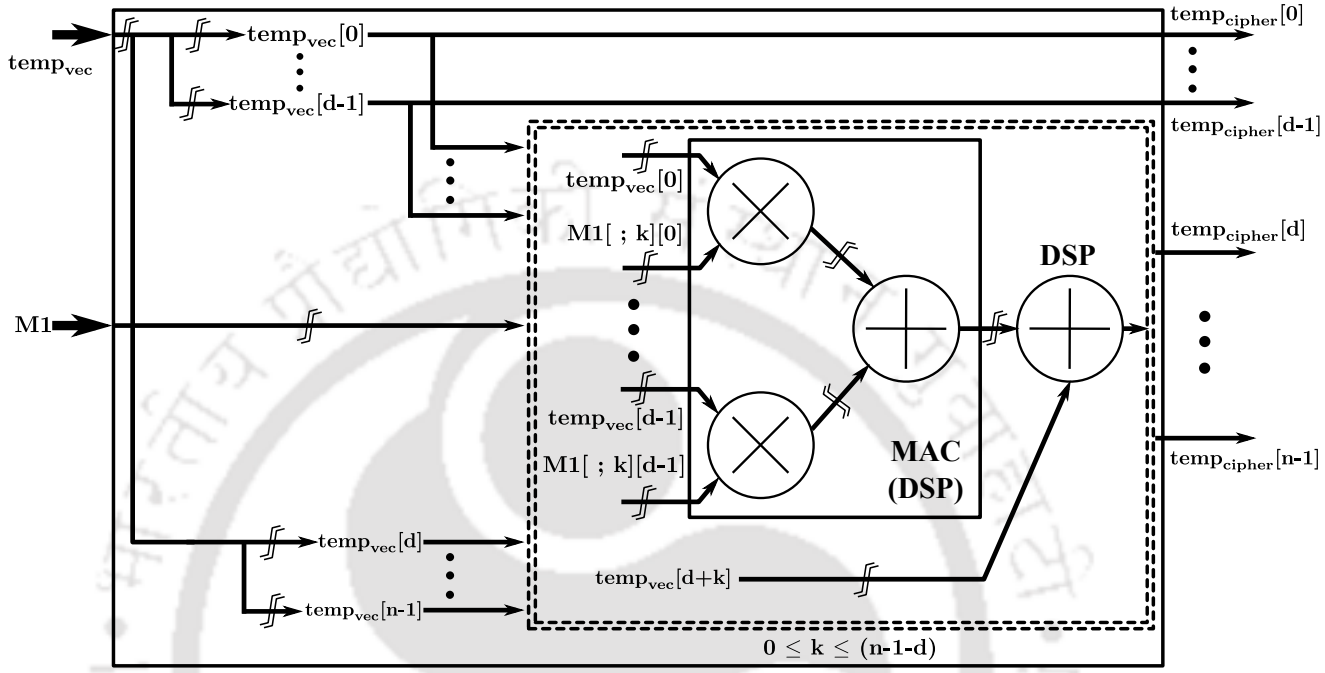


Figure 5.4: Matrix Multiplier Module Architecture

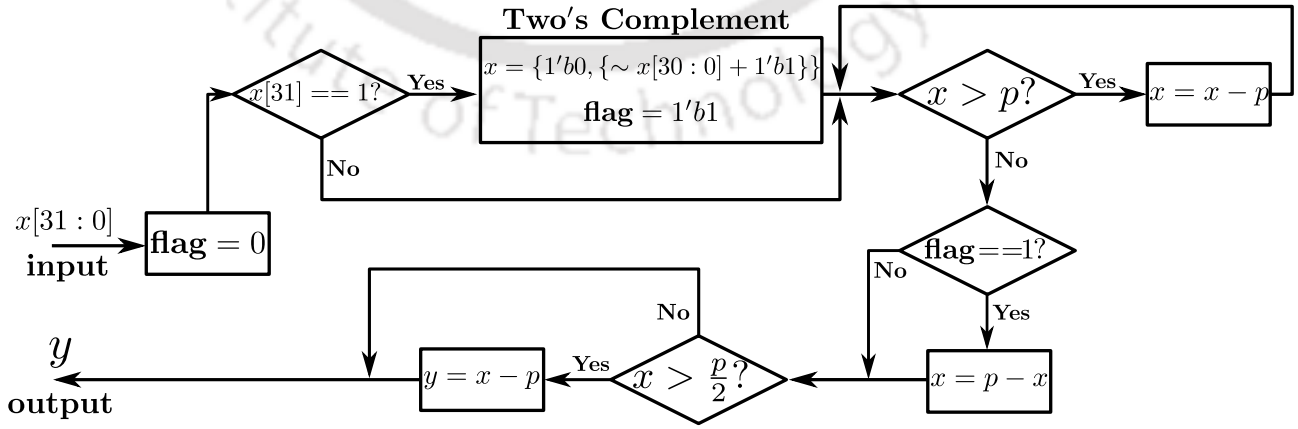


Figure 5.5: Modular Function Flow

$flag$ is employed to indicate whether x is a positive or a negative number. This information is used later to calculate modular operation appropriately.

Modular Function Bitwise Architecture: Figure 5.6 represents the bitwise architecture of the modular function block. Here, x is a 32-bit input register; $flag$ is a one-bit internal register, and $x \ mod \ p$ is a 32-bit output register. The dotted rectangles are temporary registers used to store intermediate results, and the blocks marked as LUT are implemented on the FPGA fabric. The $2'$s complement unit is utilized to obtain the absolute value of negative input x. For $p = 131$, 32-bit input register bit-width is sufficiently large to accommodate all the internal computations to avoid overflow or underflow. Here, p is chosen based on the analysis provided in [233].

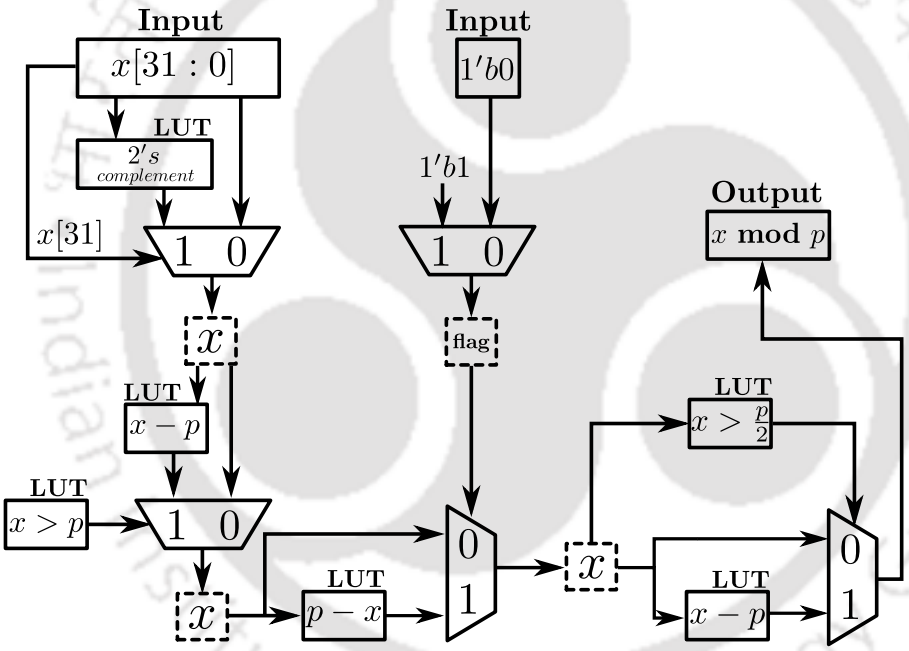


Figure 5.6: Modular Function Bitwise Architecture

The time complexity of encryption depends on three vector additions, one matrix multiplication, and one modular operation. The time complexity of addition is $O(1)$ because of the usage of n parallel adders. The complexity of matrix multiplication is $(O(n) + 1)$ due to n parallel MAC units, whereas modular operation requires $O(log_2 n)$ steps [174]. Therefore, the overall computational complexity of the proposed encryption unit is $O(1) + (O(n) + 1) + O(log_2 n) \approx O(n)$.

5.3.2 Decryption Unit

The decryption unit comprises a control unit, a matrix-vector multiplier, an inner-product calculator, a modular operation block, and a division and rounding-off module, as shown in Figure 5.7. The

input to this decryption block is a ciphertext vector c_i, a secret key vector s^{HSM}, and a matrix **-M1**. For matrix-vector multiplication, rather than storing the entire $\mathbf{M}^{-1}$, only **-M1** is stored in BRAM, similar to the encryption block. This reduces memory footprint by $n(n-d)+d^2$ and improvises resource utilization of the first method.

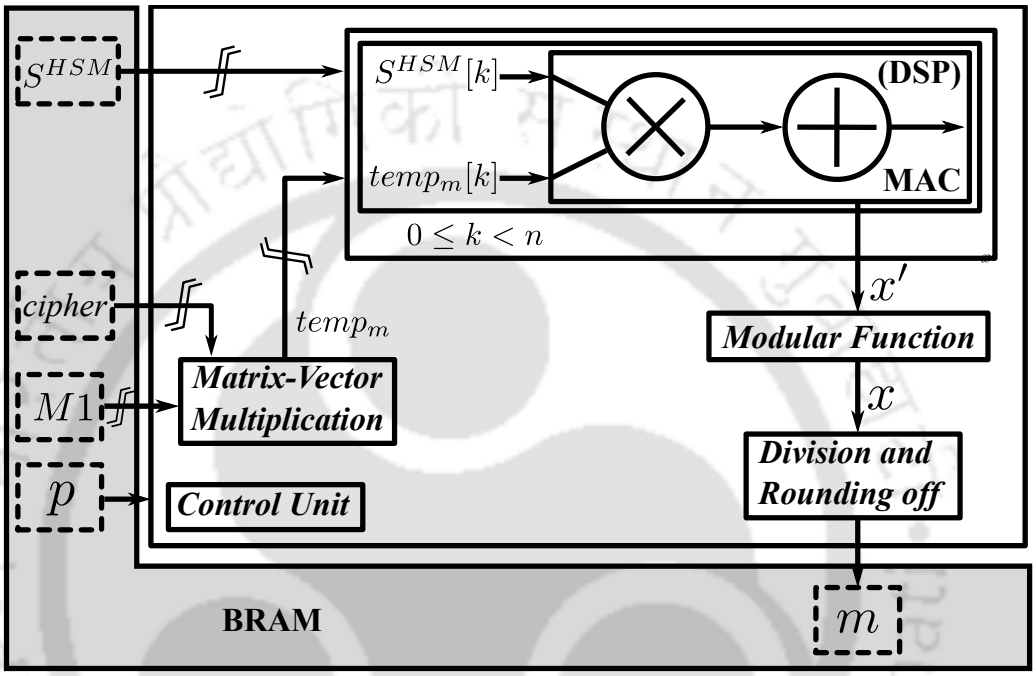


Figure 5.7: Overall Decryption Module Architecture

Matrix-Vector Multiplication

$$M^{-1} \times cipher = \left[\begin{array}{c|c} \mathbf{I}_d & \mathbf{M1}_{d\times(n-d)} \\ \hline \mathbf{0}_{(n-d)\times d} & \mathbf{I}_{(n-d)} \end{array}\right] \times \begin{bmatrix} cipher[0] \\ \vdots \\ cipher[n-1] \end{bmatrix}$$

It can be seen from the above equation that for computing matrix $M^{-1} \times cipher$ for $0 \leq k < d$, $temp_m[k] = [I_d[k;] \mid -M1_{d\times(n-d)}[k;]] \times cipher$ is performed. It is implemented using a multiply-and-accumulate (MAC) unit of the FPGA, and its architecture is depicted in Figure 5.8. Matrix-Vector multiplication with $I_{(n-d)}$ is performed using corresponding elements of the *cipher* vector. For $d \leq k < n$, $temp_m[k] = [0_{(n-d)\times d}[k;] \mid I_{n-d}[k;]] \times cipher$. Because of an identity matrix, $temp_m[k] = cipher[k]$ for $d \leq k < n$ saves $n \times (n-d)$ multiplications and $(n-1) \times (n-d)$ additions.

Vector-Vector Inner Product

Vector-vector inner product, $x = < s^{HSM}, temp_m >$, is implemented using one MAC unit having 32-bit carry look-ahead adders along with 32-bit multiplication units. The elements of s^{HSM} and

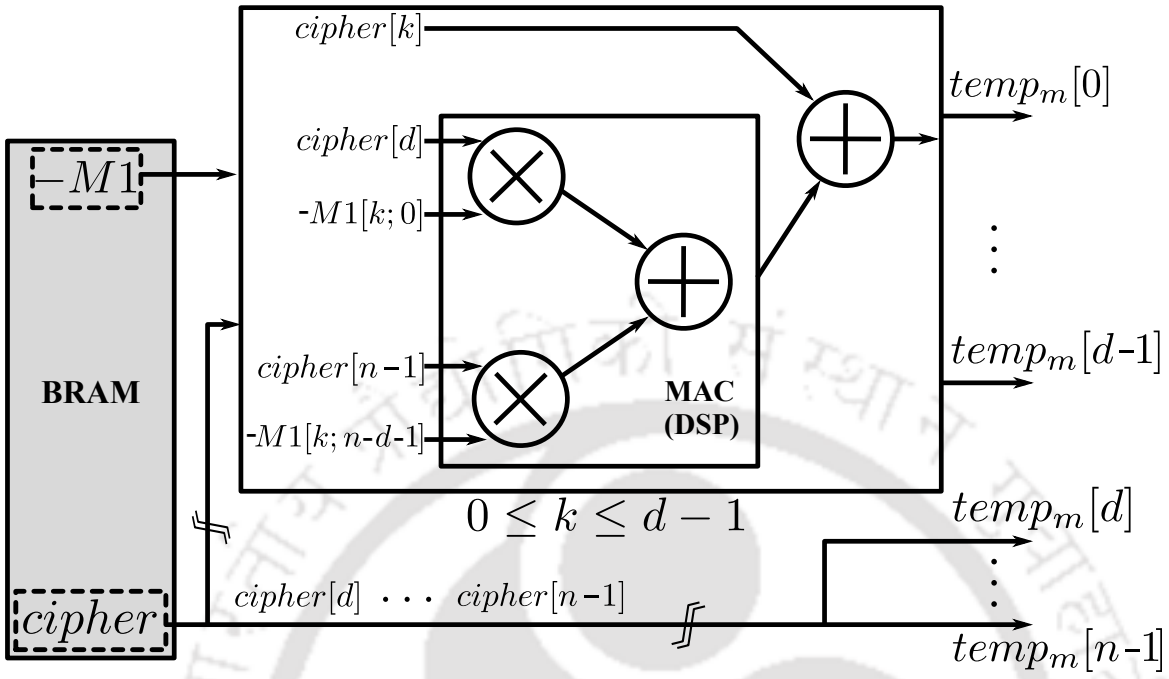


Figure 5.8: Architecture of Matrix-Vector Multiplication Module

$temp_m$ are provided as input to the MAC unit.

Modular Operation

The hardware implementation of the modular function unit used in the decryption block is similar to the modular function unit utilized in the encryption block. It is denoted as $x = x' \ mod \ p$.

Division and Rounding-off Module

The main function of the division and rounding-off module is implementing *step*-5 of Algorithm 10. The three major stages in the realization of division and rounding-off operation are converting x into a 32-bit single precision floating point number fp_x, its floating point division of by $\lfloor \frac{p}{2} \rfloor$, and finally, rounding off the division quotient to the nearest integer and taking *mod* 2 to the output. Detailed descriptions of these stages are given below.

Convert x into 32-bit single precision floating point number: A bitwise flow diagram of the conversion of 32-bit integer x to 32-bit single precision floating point number fp_x is illustrated in Figure 5.9. Here, $fp_x[31]$ is a sign bit, $fp_x[30:23]$ is an exponent and $fp_x[22:0]$ is a mantissa. First, fp_x is initialized to $32'b0$, and then the sign bit of fp_x is set. Variable $shift_x$ is used to mark the location of the first set bit in $x[30:0]$, which helps in calculating exponent bits $fp_x[30:23]$. Later,

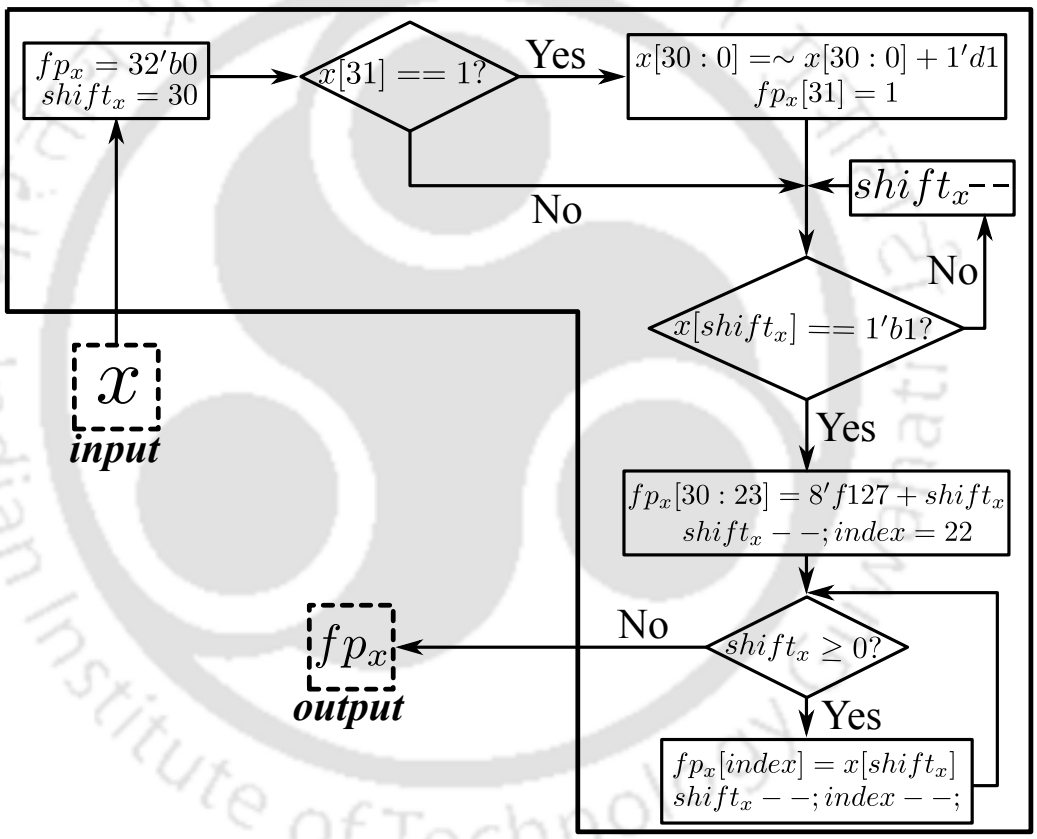


Figure 5.9: Conversion of x into Floating Point

$shift_x$ is used to assign $fp_x[22:0]$ starting from MSB.

Floating-point division of fp_x by $\lfloor \frac{p}{2} \rfloor$ is carried out by employing an internal 32-bit single precision floating point divider of DSP slices available on the FPGA.

Rounding off the division quotient to the nearest integer and taking *mod* 2**:** As given in *step*-4 of Algorithm 5, the result of the division is stored in 32-bit register y_{temp}. The following equation is employed for the floating point to integer conversion.

$$FI_{conv} = (-1)^{sign} \times 2^{E-127} \times (1 + \sum_{i-1}^{23} y_{temp}[23-1]2^{-i})$$

The position of decimal point, *decPoint* in y_{temp}, is calculated as given in *step*-5 of Algorithm 5. If *decPoint* is < -1, the absolute value of y_{temp} is less than 0.5, which rounds off to 0. If *decPoint* is -1, the absolute value of $y_{temp} \geq 0.5$ rounds off to 1. If *decPoint* is equal to 1, the absolute value of y_{temp} is 1. For *decPoint* > 0, the rounded-off integer value of y_{temp} is calculated as given in the bitwise flow diagram shown in Figure 5.10. Later, *mod* 2 on y is performed after rounding it off, and the LSB of output is considered as the final decrypted message bit.

Algorithm 15: : **Division & Rounding off**

Input: Integer x, odd prime p

Output: Integer $y = \left\lfloor \frac{x}{\lfloor \frac{p}{2} \rfloor} \right\rceil \bmod 2$

procedure BEGIN
  $\lfloor \frac{p}{2} \rfloor$ = right shift p by 1-bit
  convert x, $\lfloor \frac{p}{2} \rfloor$ to 32-bit single precision floating point
  32-bit floating point division: $y_{temp} = \frac{x}{\lfloor \frac{p}{2} \rfloor}$
  decPoint = $y_{temp}[30:23]$-127
  if(decPoint==0 || decPoint==-1) **then** y=1;
  else if (decPoint<-1) **then** y=0;
  else
  Integral y_{temp} int(y_{temp}) using {mantissa, decPoint}
    if (next bit of decPoint==1) **then** y = int(y_{temp}) + 1
    else y = LSB of int(y_{temp})
  return y
  end
end procedure

The modular operation illustrated in equation 5.8 is equivalent to selecting the least significant bit (LSB). Thus, the LSB of rounded integer produces a decrypted message m. The complexity of the decryption module depends on three modules: a matrix multiplier, a modular function block, and a division and rounding-off unit. Their computational complexities are $(O(n) + 1)$, $O(log_2 n)$,

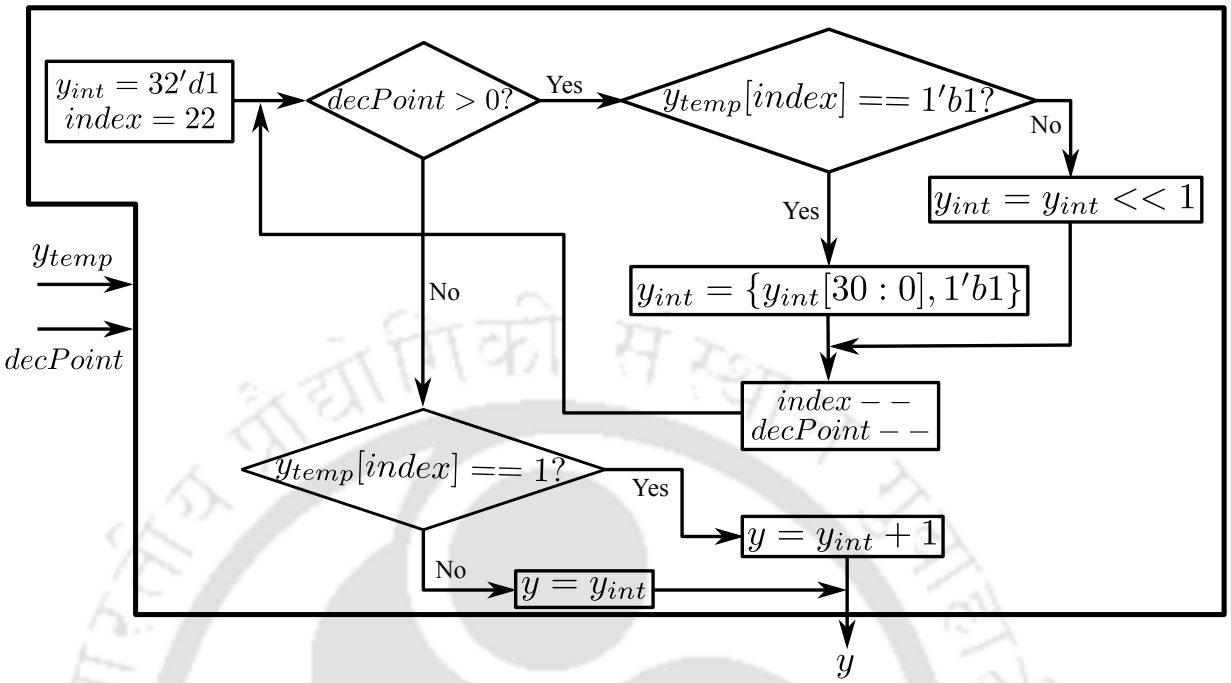


Figure 5.10: Rounding-off Division Quotient to Nearest Integer

and $O(\lceil n \rceil)$ [237] using n parallel blocks. Therefore, the overall complexity of decryption module is $(O(n)+1)+O(log_2 n)+O(\lceil n \rceil)) \approx O(n)$.

5.3.3 Recryption Unit

Here, Algorithm 2 presents hardware implementation steps of homomorphic addition for two input ciphertexts. This addition translates to bitwise-XOR operation on the input message bits. The homomorphic addition block performs arithmetic addition of input ciphertexts generating an intermediate vector named $\mathbf{x}_{add}$. Later, the modular function block executes a "**mod p**" operation on each element of $\mathbf{x}_{add}$ vector to obtain the final ciphertext $\mathbf{c}_{add}$. The cumulative complexity of homomorphic addition (with n addition blocks) and modular operation is $O(n)+O(log_2 n) \approx O(n)$.

Algorithm 16: : **Homomorphic Addition**

Input: $\mathbf{C} = \mathbf{c}^i{}_{1 \le i \le l\text{-}d}$
Output: $\mathbf{c}_{add}$
1: **procedure** BEGIN
2: $\mathbf{x}_{add} = \{\Sigma \mathbf{c}^i : 1 \le i \le (l-d)\}$
3: $\mathbf{c}_{add} := \mathbf{x}_{add}$ mod p
4: **return** $\mathbf{c}_{add}$
5: **end**
6: **end procedure**

Table 5.1: Timing, Resource-Utilization and Complexity of the First PHE Scheme

Unit	Operation	Clock Cycles	Delay (ns)	Complexity	Mbps	Resource	Utilization
Encryption	Matrix Multiplier	255	5100	$O(1)$ + $(O(n)$ +	5.814	LUT (as logic) LUT (as memory)	6881 741
	Modular Function	78	1560	1) + $O(log_2 n) \approx$		Register (as Flip Flops)	7058
	Total (5 message bits encryption)	344	6880	$O(n)$		DSP (DSP48E1) Block RAM	3 13.5
Decryption	Matrix Multiplier	229	4580	$(O(n)$ +	2.472	LUT (as logic)	6118
	Modular Function	544	10880	1) +		LUT (as memory)	822
	Division & Rounding Off	35	700	$O(log_2 n)$+ $O(\lceil n \rceil) \approx$		Register (as Flip Flops)	7865
	Floating Point Division	27	540	$O(n)$		DSP (DSP48E1)	3
	Total (5 ciphertexts decryption)	809	16180			Block RAM	12.5
Recryption	Adder Block	110	2200	$O(n)$ +	14.084	LUT (as logic)	938
	Modular Function	31	620	$O(log_2 n) \approx$ $O(n)$		Register (as Flip Flops)	365
	Total (5 ciphertexts)	142	2840				

The proposed PHE is emulated with parameters $p = 131$, $l = 4$, $r = 2$, $n = 10$, and input message bits= $\{1, 0, 1, 0, 1\}$ using Xilinx Zynq-7000 FPGA operating at 50 MHz. The detailed timing and resource utilization of encryption, decryption, and recryption units are shown in Table 5.1. The next section describes the proposed resource-efficient second PHE scheme.

5.4 Resource-Efficient Second PHE Scheme

In this section, we propose a lightweight variant of homomorphic algorithm based on lattice LWE based PHE algorithm discussed in [232]. For the description of algorithms, following notations are depicted. Here, λ is a security parameter employed in an algorithm. $\mathbb{R}$, $\mathbb{N}$ and $\mathbb{Z}$ denote real, natural and integer numbers, respectively. $x \xleftarrow{\$} \mathcal{S}$ states x is sampled from a set $\mathcal{S}$, which is uniformly random. $\mathbb{F}_q$ is a finite field of order q, where q is a prime number. Further, $\mathbb{F}_q[x]_{\leq d}$ denotes a polynomial set of x variables, whose degree is $\leq d$, such that $d \in \mathbb{N}$. Here $\{\mathbf{A}, \mathbf{B}, \dots\}$ denote tensor matrices. The i^{th} column and j^{th} row of $\mathbf{A}$ are illustrated by $\mathbf{A}(:, i)$ and $\mathbf{A}(j, :)$, respectively. A function $f(x) : \mathbb{N} \to \mathbb{R}$ is negligible, iff $\forall \omega \in \mathbb{N}$, $\exists$ an integer n_ω, such that $|f(x)| < \frac{1}{x^\omega}$ $\forall x > n_\omega$.

5.4.1 Algorithm Formulation

The details of basic building blocks of low power IoT variant of the proposed PHE scheme, viz. key generation, encryption, decryption and recryption are described below.

5.4.1.1 Key Generation

A set of binary matrices $\mathbf{L}$ and $\mathbf{R}$ is chosen as cryptographic keys, with dimensions $dim(\mathbf{L}) = m$ and $dim(\mathbf{R}) = n$, where $m < n$. For enhancing the strength of key sk, elements of the rows of $\mathbf{L}$ and $\mathbf{R}$ are chosen from the coefficients of an irreducible polynomial. One row of each of these matrices is selected randomly from the group of irreducible polynomials of degrees $(m-1)$ and $(n-1)$. The coefficients of these polynomials lie in $\mathbb{F}_q$, a finite field of order q, where q is a prime number. Further, $\mathbf{L}$ and $\mathbf{R}$ together form the secret key sk, which is the same for both encryption and decryption processes. sk is generated using a key generation algorithm presented in Algorithm 2. $x \xleftarrow{\$} \mathcal{S}$ states that x is sampled from a set $\mathcal{S}$, which is uniformly random. Further, $\mathbb{F}_q[x]_{\leq d}$ denotes a polynomial set of x variables, whose degree is $\leq d$, such that $d \in \mathbb{N}$ and λ is a security parameter [232]. The i^{th} column and j^{th} row of a tensor matrix $\mathbf{A}$ are illustrated by $\mathbf{A}(:, i)$ and $\mathbf{A}(j, :)$, respectively.

Algorithm 17: : **GenKey**(λ, m, n)

Input: λ, m, n
Output: sk: $(\mathbf{L}, \mathbf{R})$
procedure BEGIN
  $L \xleftarrow{\$} GL_m(\mathbb{F}_q)$, $R \xleftarrow{\$} GL_n(\mathbb{F}_q)$
  sk : ($\mathbf{L}$, $\mathbf{R}$)
  return sk
  END
end procedure

5.4.1.2 Encryption

As we know, the input to encryption module is $m \in \mathbb{F}_q^m$, secret key sk, and a pseudo-random number matrix D, which can be stated using equation 5.12. It generates ciphertext $\mathbf{C} = \{C^i \in \mathbb{F}_{q^l}^{m \times n} | 1 \leq i \leq \eta\}$ for $\eta \in \mathbb{N}$, which lies in $\mathbb{F}_q$. There may be different ciphers for the different random numbers for a message vector, which are represented by η.

$$D = \{X \in \mathbb{F}_{q^l}^{m \times n} | \sum_{i=1}^{n} X(:, i) = 0 \bmod q^l\} \tag{5.12}$$

Each column of $\mathcal{Q}$ is set as $(\mathcal{Q}^i(:,\beta) \leftarrow \mathcal{Q}^i(:,\beta) + \mathbf{m})$, where $\beta \in_{\$} [n]$. Further, C^i can be produced by equation 5.13.

$$C^i = L.\mathcal{Q}^i.R + c_i.N^i, \quad N^i \in \mathbb{F}_{q^l}^{m \times n} \tag{5.13}$$

As mentioned, random numbers are generated using BluXor PRNG [238]. After that, η ciphers are generated for every message, out of which only one cipher is free from the noise. This represents a set of output ciphertexts $\mathbf{C} = \{C^i_{1 \le i \le \eta}\}$, and a secret key is required to decrypt ciphers to extract messages. Algorithm 2 describes encryption steps to generate ciphertext from plaintext.

Algorithm 18: : **Encrypt** (m, sk, D)

Input: (m, sk, D)
Output: $\mathbf{C} = \{ C^i \in \mathbb{F}_q^{m \times n}, 1 \le i \le \eta \}$

procedure BEGIN
  sample $\mathcal{Q}^i \xleftarrow{\$} D$,
  set $\mathcal{Q}^i(:,\beta) \leftarrow \mathcal{Q}^i(:,\beta) + \mathbf{m}$ *for* $\beta \xleftarrow{\$} [n]$
  set $C^i = \mathbf{L} \cdot \mathcal{Q}^i \cdot \mathbf{R}$
  return $C = \{C^i_{1 \le i \le \eta}\}$
  END
end procedure

5.4.1.3 Decryption

The decryption module takes $\mathbf{C}$ and sk having $\mathbf{L}$ and $\mathbf{R}$ as inputs and generates corresponding plaintext m as an output. Since the proposed PHE scheme is a symmetric algorithm, an identical secret key is employed for the encryption and decryption. The decryption operation can be depicted using equation 5.14, where $\mathbf{m} = \mathbf{m}_i$, such that $\mathbf{m}_i \in \mathbf{F}_q^m$. The steps for generating plaintext from ciphertext are described in Algorithm 2.

$$\mathbf{m}_i = \sum_{j=1}^{n} (L^{-1}.C^i.R^{-1})(:,j) \tag{5.14}$$

Validation of Correctness of Decryption

We are aware, the decryption module should have a non-negligible probability to produce plaintext from the ciphertext. In the case of our proposed PHE algorithm, the decryption module is valid if the probability of ciphertext $\mathbf{C}$ to plaintext $\mathbf{m}$ is $\frac{1}{(\eta-1).q^{m(1-l)}+1}$ [232]. Let us consider A is an event of the decryption of $\mathbf{C}$ to $\mathbf{m}$, which lies inside $\mathbb{F}_q^m$, whereas B is an event where the decrypted outcome falls in the noisy domain. Thus, the outcome of B is $C^i = L.\mathcal{Q}^i.R + N^i = (L.\mathcal{Q}^i + \hat{N}^i)R$, where $\hat{N}^i$

Algorithm 19: : **Decrypt**(C,sk)

Input: $C = \{C^i_{1\leq i\leq \eta}\}$, secret key sk
Output: message $\mathbf{m}$
1: **procedure** BEGIN
set $\mathbf{m}_i = \sum_{j=1}^{n}(\mathbf{L}^{-1}C^i\mathbf{R}^{-1})(:,j)$
set $\mathbf{m} = \mathbf{m}_i$
2: **return m**
3: END
4: **end procedure**

and N^i have one-to-one correspondence and $\hat{N}^i \in \mathbb{F}_{q^l}^{m\times n}$. Therefore, the required probability can be estimated using equation 5.15.

$$P(\bar{B}|A) = \frac{P(A|\bar{B}).P(\bar{B})}{P(A)} \tag{5.15}$$

Here, $P(A|\bar{B})$ in equation 5.15 states the probability of getting $\mathbf{C}$ to $\mathbf{m}$ after decryption in the noiseless domain lying in $\mathbb{F}_q^m$. If there exists at least one noise-free encryption of $\mathbf{m}$ from $\mathbf{C}$, $P(A|\bar{B}) = 1$ and $P(\bar{B}) = \frac{1}{\eta}$. Further, $P(A)$ can be determined employing equation 5.16. $P(A|B)$ is the probability of decryption from the noisy elements, $\mathbb{F}_q^m$. Here, $P(A|B) = \frac{q^m}{q^{lm}} = q^{m(1-l)}$, which can be derived using equation 5.16.

$$P(A) = P(A|B).P(B) + P(A|\bar{B}).P(\bar{B}) \tag{5.16}$$

The probabilities of $P(A)$ and $P(\bar{B}|A)$ can be determined using equations 5.15 and 5.16, and are elaborated in equations 5.17 and 5.18. This proves the correctness of the decryption algorithm.

$$P(A) = q^{m(1-l)}(\frac{\eta-1}{\eta}) + \frac{1}{\eta} \tag{5.17}$$

$$P(\bar{B}|A) = \frac{1}{(\eta-1)q^{m(1-l)}+1} \tag{5.18}$$

5.4.1.4 Recryption Process and Validation of its Correctness

Since homomorphic addition can be applied directly on ciphertexts, it translates to the addition of underlying encrypted message bits without compromising the security of homomorphic encryption. The algorithm for recryption of the proposed PHE is illustrated in Algorithm 2 along with its

correctness below.

Algorithm 20: : **Recryption**

Input: $C_1 = \{C^i_{1(1\leq i\leq \eta)}\},\ C_2 = \{C^i_{2(1\leq i\leq \eta)}\}$

Output: C_{add}

procedure BEGIN

set $\pi \xleftarrow{\$} \mathcal{P}([\eta])$

set $C_{add} := \left\{C_1^i + C_2^{\pi(i)} : 1 \leq i \leq \eta\right\}$

return C_{add}

END

end procedure

Consider k ciphertexts, $C_1^{\pi_0(i)}$, $C_2^{\pi_1(i)}$, ..., $C_k^{\pi_{k-1}(i)}$, such that $\mathbf{Add}(\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k) = (C_1^{\pi_0(i)} + C_2^{\pi_1(i)} + \ldots + C_k^{\pi_{k-1}(i)})$, where decryption of the addition function yields a result in $\mathbb{F}_q^m$. From equation 5.18, it can be observed that the probability of single ciphertext decryption is $\frac{1}{(\eta-1)q^{m(1-l)}+1}$. Therefore, the overall decryption probability of all the elements of $\mathbf{Add}(\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k)$, which belong to the noiseless domain, is $\frac{1}{(\eta^k-1)q^{m(1-l)}+1}$. Let event A denote the decryption of $(C_1^{\pi_0(i)} + C_2^{\pi_1(i)} + \ldots + C_k^{\pi_{k-1}(i)})$, which belong to $\mathbb{F}_q^m$. Let event B denote at least one of $C_j^{\pi_{j-1}(i)}$ to be from the noisy domain, where $j \in [k]$ and $\bar{B} \subseteq A$. This leads to $P(\bar{B}|A)$ to be expressed as equation 5.19, where $P(\bar{B}) = \frac{1}{\eta}$ for each noiseless $\mathbf{C}_j$, $1 \leq j \leq k$, and $P(A|\bar{B}) = 1$ and $P(A|B) = \frac{q^m}{q^{lm}}$, if the sum of $(k-1)$ randomly chosen elements lies in $\mathbb{F}_q^m$. Thus, $P(A)$ can be determined using equation 5.20. Finally, $P(\bar{B}|A)$ illustrated in equation 5.21 can be estimated using equations 5.19 and 5.20. This validates recryption for a general case.

$$P(\bar{B}|A) = \frac{P(\bar{B})}{P(A)} \tag{5.19}$$

$$\begin{aligned} P(A) &= P(A|B).P(B) + P(A|\bar{B}).P(\bar{B}) \\ &= (1 - \frac{1}{\eta^k})q^{m(1-l)} + \frac{1}{\eta^k} \end{aligned} \tag{5.20}$$

$$P(\bar{B}|A) = \frac{1}{(\eta^k - 1)q^{m(1-l)} + 1} \tag{5.21}$$

$$\sum_{i=1}^{n}(L^{-1}(\mathbf{C}_1^i + \mathbf{C}_2^{\pi(i)})R^{-1}) = \mathbf{m}_1 + \mathbf{m}_2 \ (mod\ q) \tag{5.22}$$

$$P(\mathbf{C}_{add}) = (1 - \frac{1}{\eta^2})q^{m(1-l)} + \frac{1}{\eta^2} \tag{5.23}$$

In Algorithm 2, the recryption operation over two encrypted elements is presented ($k = 2$), where $\mathbf{C}_{add} = \mathbf{C}_1 + \mathbf{C}_2 = \mathbf{C}_1^i + \mathbf{C}_2^{\pi(i)}$, $1 \leq i \leq \eta$. Since $\mathbf{C}_1^i$ and $\mathbf{C}_2^{\pi(i)}$ lie in the noiseless domain $\mathbb{F}_q^m$, the overall recryption operation for two encrypted messages can be depicted by equation 5.22. Further, by substituting $k = 2$ in equation 5.20, $P(\mathbf{C}_{add})$ can be found in the range of $\mathbb{F}_q^m$ given in equation 5.23. Note that the expected number of decryptions needed to find output in $\mathbb{F}_q^m$ is $P(\mathbf{C}_{add})^{-1} = \frac{\eta^2}{(\eta^2-1)q^{m(1-l)}+1}$.

5.4.2 Hardware Implementation

The methodology of the second PHE algorithm and its detailed mathematical analysis are presented in [232]. Figure 5.11 presents a detailed architectural description of the second scheme. Its encryption process is the combination of two steps, viz., the formation of Q matrix and the matrix multiplication of secret key matrices L and R. Similarly, the decryption module is composed of a matrix multiplication unit of secret key L^{-1}, cipher matrix, and secret key R^{-1}. The basic hardware architecture of the matrix multiplier in encryption and decryption units is identical, and the only difference between these units is the inputs, key and cipher matrix representation, which are explained below.

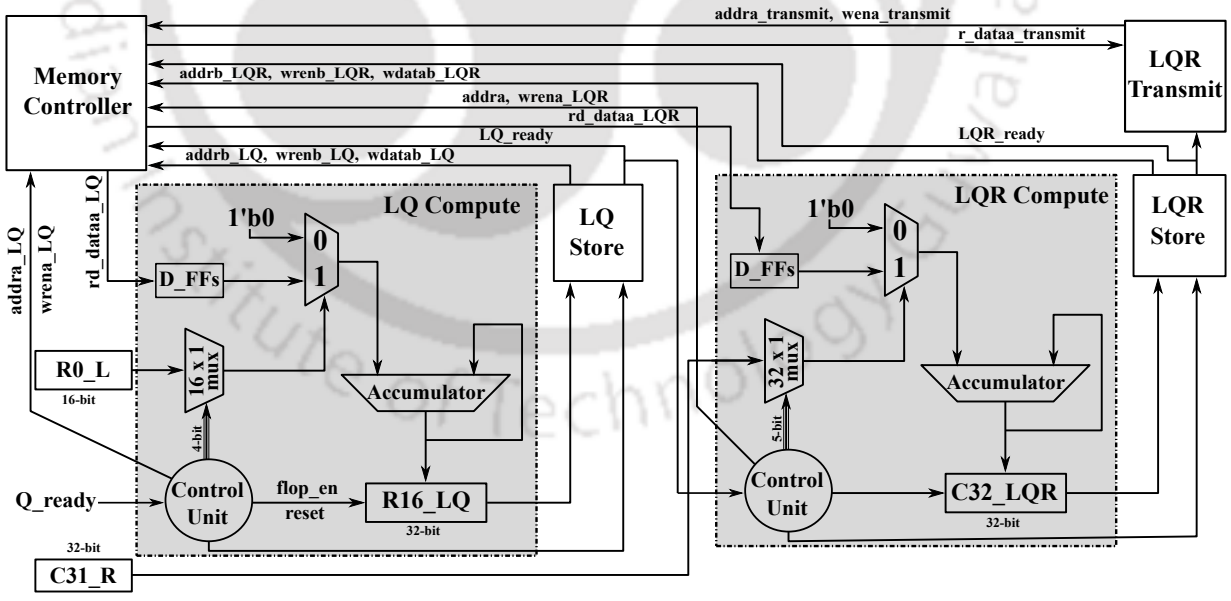


Figure 5.11: Architecture of Second PHE Scheme

As parameters, the encryption module requires a secret key sk comprised of two matrices, L and R. Subsequently, PRNG is employed to populate elements of a random matrix D. As mentioned earlier, $BluXor$ is selected as a PRNG [238] due to its lower resource consumption and has been validated with

NIST benchmarks. It generates 64-bit random strings and can simultaneously populate 64 matrix elements of D in a single clock cycle reducing computation time. After that, an intermediate matrix $\mathcal{Q}$ is formulated depending on the input plaintext bits, such that column-wise addition of the elements of $\mathcal{Q}$ results in a zero vector. Subsequently, a binary message vector m is added with a randomly chosen column of $\mathcal{Q}$. All the other elements of $\mathcal{Q}$ are chosen from D. Finally, L is multiplied with $\mathcal{Q}^i$ followed by R to obtain a ciphertext $\mathbf{C}^i$, which is i^{th} message vector sequence. In this manner, m-bit plaintext generates an $m \times n$ ciphertext. *BluXor* takes eight clock cycles to generate random numbers. Further, $\mathcal{Q}$ matrix generation and $L \times \mathcal{Q} \times R$ multiplication consume 16 and 48 clock cycles to produce ciphertexts.

It can be observed in Figure 5.11 that the shaded blocks are the matrix multiplication units implemented using LUTs and flip-flops. As mentioned above, encryption is the matrix multiplication of three matrices, L, Q, and R. Here, L and R are of specific formats shown in Figure 5.12, and Q is a 16×32 dense matrix. Q is stored in 2 KB BRAM (mem_compute) with 32-bit word size (16×32). The size of every element stored in it is 32-bit. As illustrated in Figure 5.12, two sets of secret keys, key^L and key^R of 16-bit and 32-bit, are employed to generate L (16×16) and R (32×32), respectively. Note that key^L and key^R are stored in a 16-bit register (R_1^L) and a 32-bit register (C_{32}^R).

As shown in Figure 5.11, there are two multiplication units. The first multiplier performs LQ, which produces P, and later the second multiplier evaluates PR to generate ciphertext $C = LQR$. The first row of LQ is evaluated by accumulating nonzero elements of the first row of L and the rows of Q by employing a 32-bit hardware adder until all the rows of Q are utilized in this process. The output of this adder is temporarily stored in a 32-bit register (R_{16}^{LQ}), and the final output of the iteration is stored in R_{16}^P during the computation of LQ. It can be observed that P is comprised of 16 numbers of 32-bit words denoted by $R_1^P, \ldots, R_{16}^P$. Here, R_{16}^{LQ} acts as a shared memory during the encryption operation. key^R stored in C_{32}^R is multiplied with P and its result is saved in R_{16}^Q. As stated in Figure 5.12, $\sum_{i=1}^{32} P_i R_i^R$ is computed by summing the columns of P corresponding to the nonzero elements of R. The result of this multiplication is saved in a 32-bit register C_{32}^{LQR} temporarily. It is stored later in the memory location allocated for C_{32}^P. After the encryption operation, each column of ciphertext is transmitted in the burst mode. Further, while writing into the memory of ARM SoC using FPGA interface, first R_{16}^M is transmitted, followed by R_1^M, R_2^M, $\ldots$, R_{15}^M. This memory interfacing technique ensures seamless transfer of encrypted messages from FPGA to ARM SoC for onward transmission.

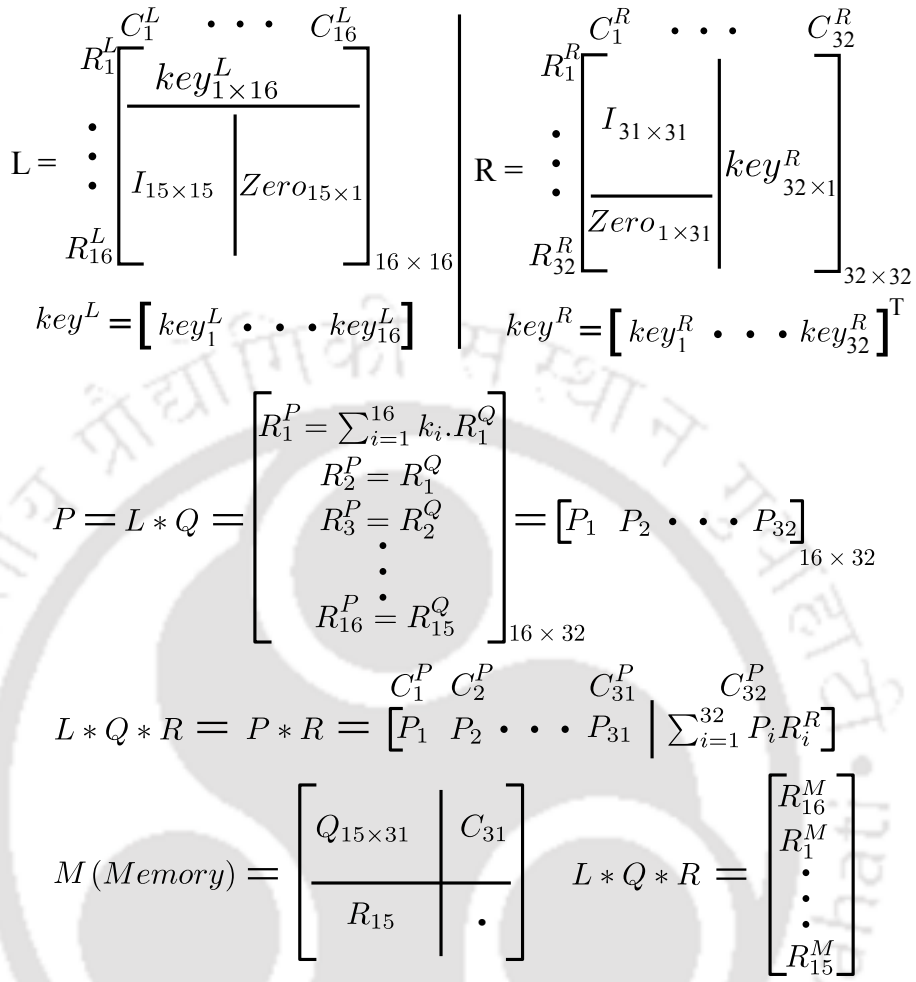


Figure 5.12: Matrix Representation and Encryption Steps

All the encryption steps mentioned above are presented in Figure 5.12.

The decryption module takes a ciphertext matrix **C** with pre-calculated L^{-1} and R^{-1} as inputs. As stated earlier, L^{-1} is multiplied with **C** and R^{-1} to evaluate plaintext m. Here, dimensions of L^{-1}, **C**, and R^{-1} are $m \times m$, $m \times n$, and $n \times n$, respectively. This results in an intermediate matrix of dimension $m \times n$. Further, the final m-bit plaintext is extracted after the column-wise addition of all the elements of this intermediate matrix. Note that the decryption steps are identical to encryption operations, where input L, R, and Q are replaced by L^{-1}, R^{-1}, and C. All the matrix operations required for the decryption are presented in Figure 5.13 for completeness.

5.4.3 Complexity Analysis

The complexity of the proposed encryption scheme depends on *BluXor* PRNG, generation of $\mathcal{Q}$, and the multiplication of $L \times \mathcal{Q} \times R$. The computational complexities of the steps mentioned above are

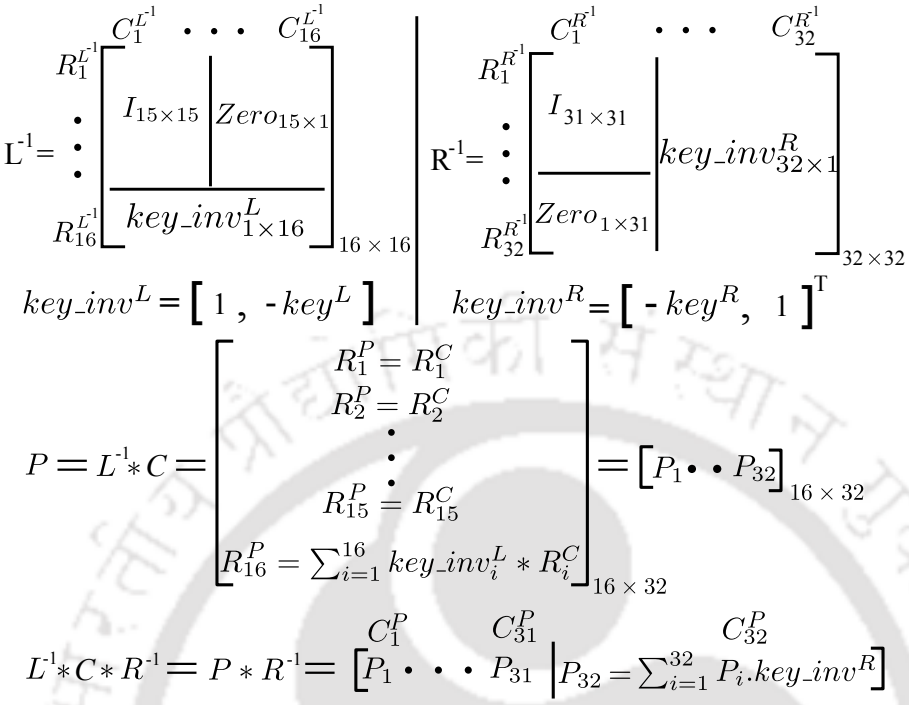


Figure 5.13: Matrix representation of L^{-1} and R^{-1}

$(O(2n-1)+O(log_2 n)+O(2))$, $O(1)$, and $O(2n+2)$, respectively. Therefore, the overall complexity of encryption operation is $(O(2n-1)+O(log_2 n)+O(2))+O(1)+O(2n+2) \approx O(n)$. The proposed decryption method comprises two matrix multiplication units, each having computational complexity $O(n+1)$. Thus, the overall complexity of the decryption operation is $O(2n+2) \approx O(n)$.

Later, the recryption of two ciphertexts is achieved by adding C_1^i and $C_2^{\pi(i)}$. A random permutation operation is performed on $C_2^{\pi(i)}$, which takes six clock cycles to complete. After that, an addition is performed using C_1^i and $C_2^{\pi(i)}$, which consumes five clock cycles only. Since the recryption module has one matrix addition block having computation complexity $O(1)$; therefore, the overall complexity of the recryption operation is $O(1)$. Thus, the computational complexity of the proposed lightweight PHE is $O(n)$. It is validated using Xilinx Zynq-7000 FPGA operating at 50 MHz, and timing and resource utilization of its encryption, decryption, and recryption units are presented in Table 5.2. The following section presents details of the integration of PHEs realized on the FPGA with an ARM-SoC.

5.5 Integration of FPGA accelerator & ARM-SoC

In the previous sections, algorithmic description and hardware implementation of the proposed PHE scheme is elaborated. The FPGA-based hardware implementation can be referred as hardware

Table 5.2: Timing, Resource Utilization and Complexity of the Lightweight High-speed Second PHE Scheme

Unit	Operation	Clock Cycles	Delay (ns)	Complexity	(Mbps)	Resource	Utilization
Encryption	Random Number Generation	8	160	$(O(2n-1)+O(log_2 n)+O(2))$	177.778	LUT (as logic)	5022
						LUT (as memory)	467
	Formation of Q	16	320	$O(1)+O(2n+2) \approx O(n)$		Register (as Flip Flops)	5281
	Matrix Multiplier	48	960			DSP (DSP48E1)	3
	Total (for 32 byte message)	72	1440			Block RAM	5
Decryption	Matrix Multiplier	48	960	$O(2n+2) \approx O(n)$	266.667	LUT (as logic)	4123
						LUT (as memory)	498
						Register (as Flip Flops)	5514
	Total (for 32 byte ciphertext)	48	960			DSP (DSP48E1)	0
						Block RAM	7
Recryption	Create Permutation Set	6	120	$O(1)$	1163.636	LUT (as logic)	375
	Adder Block	5	100			Register (as Flip Flops)	190
	Total (for 2 ciphertexts)	11	220			Block RAM	2

accelerator, if we employ it in any embedded system to enhance security. In this section, detailed description of the integration methodology of the proposed accelerator with an ARM based SoC is presented. As shown in Figure 5.14, the overall design can be broadly segmented into three parts, which are FPGA accelerator IP core using programmable logic (**PL**), interconnecting network between programmable system (**PS**) and **PL** IP core, and semi-custom embedded Linux. This embedded operating system (OS) handles user interface as well as cross device communication over universal serial bus (USB) or Ethernet protocol. It is to mention that the proposed design utilizes Ethernet protocols for communication and is an appropriate example of a software-hardware co-design. Details of the hardware accelerator and software interfaces are depicted below.

5.5.1 Hardware Accelerator

PL and **PS** shown in Figure 5.14 are realized using Verilog HDL on FPGA. The software of the proposed design is developed in embedded C. It is to mention that encryption, decryption and recryption units are implemented in **PL**. In the previous sections, formulation and implementation of these units along with their detailed hardware resource utilization and timing are summarized in Table 5.1 and 5.2, respectively. Further, a custom communication control logic is developed to establish

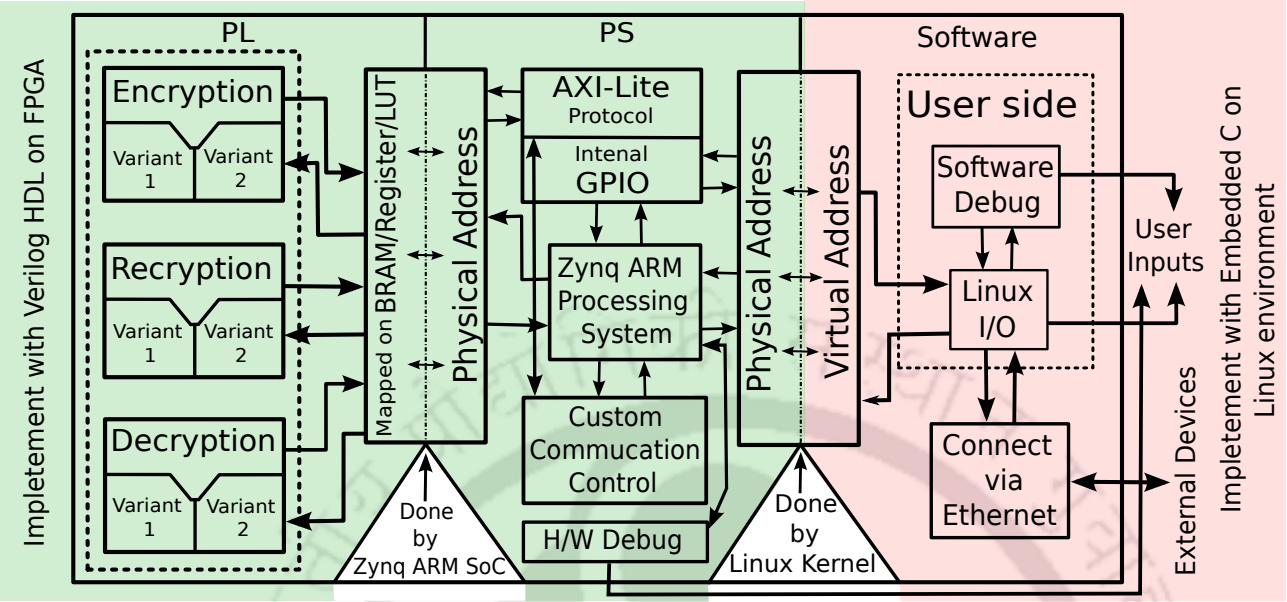


Figure 5.14: Overall Design of the Proposed Hardware Accelerator using ARM-SoC

duplex communication channel between FPGA accelerator and embedded OS, which is described below in detail.

5.5.1.1 Custom Communication Control Module

A communication control module is designed to interface **PL** with **PS** using default AXI interconnect bus. Its internal signals are shown in Figure 5.15 for completeness. Since the proposed PHE schemes are realized using FPGA along with Zynq ARM SoC acting as a base processing system, AXI bus is essential to communicate with other SoC modules, such as clock generator, interrupt controller etc. ARM SoC enabled Zynq processing system is the master device, which initiates all the transactions via connected slave devices to perform a task. The proposed communication module connect hardware accelerator with Zynq-ARM SoC via AXI interconnect to have greater control over data transmissions.

The proposed interconnect IP shown in Figure 5.15 has two slave IPs, *Custom Slave IP 1 and 2*, which consists of custom input output channels and AXI signals. The custom channels are responsible for the transfer of data from **PS** to **PL** accelerator IP core, whereas, AXI bus provides its default connectivity. There are two 32-bit input channels, i.e. $slv\_reg1\_PL\_wdata$ [31:0] and $slv\_reg3\_PL\_wdata$ [31:0], and two 32-bit output channels. i.e. $slv\_reg0\_PL\_wdata$ [31:0] and $slv\_reg2\_PL\_wdata$ [31:0] in the first slave IP along with AXI bus. The input channels carry sk, L and R from **PS** to **PL**, and

output channels provide operation completion status with vital information for hardware debugging.

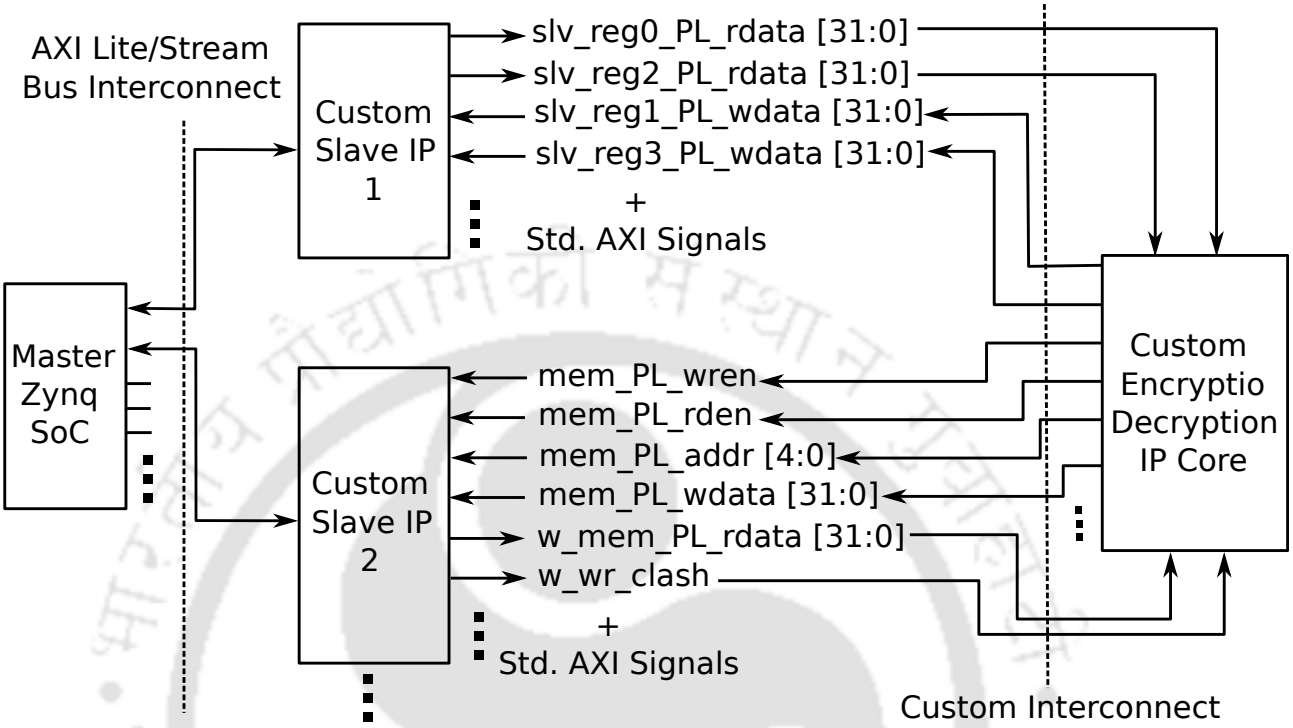


Figure 5.15: Internal Custom Communication Control Module

The second slave IP is composed of four input channels, i.e. *mem_PL_wren*, *mem_PL_rden*, *mem_PL_addr* [4:0] and *mem_PL_wdata* [31:0], and two channels dedicated to output signals, i.e. *w_mem_PL_rdata* [31:0] and *w_wr_clash*. Here, *mem_PL_wren* and *mem_PL_rden* indicate write or read status of dual port BRAMs. It is to mention that during encryption and decryption operations, the status of these signals should be 01 $\rightarrow$ 10 and 10 $\rightarrow$ 01, respectively. A 5-bit address pointer, *mem_PL_addr* [4:0], provides physical address of the input given to an encryption unit or the output produced by a decryption unit. It is to mention, for implementing logic on Zynq FPGA, four memory locations of one byte each are required to represent 32-bit data. Therefore, for encryption and decryption operations, we need minimum eight addresses to store intermediate data. Here, *mem_PL_addr* [4:0] can accommodate 32 addresses, but only eight addresses are used in the proposed design implementation. It is to be noted that the proposed design is based on memory mapped I/O. This allows each I/O to connect with Zynq processing system through a unique address, which makes our proposed implementation a scalable one. As we know, Zynq processing system allocates $64K$ addresses to be utilized, thus, for performance driven applications multiple accelerator IP cores can be employed in parallel easily using memory mapped design methodology. It is to mention that we show-

case sequential implementations of the proposed PHE schemes to validate their correctness, however, as mentioned above, these schemes can be parallelized for higher throughput with ease. The 32-bit input *mem_PL_wdata* [31:0] is the input message data to be encrypted or recrypted or decrypted. Similarly, *w_mem_PL_rdata* [31:0] is a 32-bit output after encryption or recryption or decryption operations. An output flag for error signal *w_wr_clash* is set, if there is any ambiguity while storing or reading data from/into *mem_PL_wdata* or *w_mem_PL_rdata*, respectively. If there is no clash, then *w_wr_clash* is set to "0". This flag is helpful during debugging operations.

All these 10 channels mentioned in Figure 5.15 are connected to control block of the hardware accelerator. Based on these signals, the seamless duplex data transfer between **PS** to **PL** can be ensured.

5.5.1.2 Virtual to Physical Memory Mapping

As mentioned earlier, the proposed architecture is based on memory mapped I/O, therefore the translation of virtual address from OS to physical address of an FPGA and storing in BRAM/register become its integral part. From the Figure 5.14, it can be observed that the conversion of address happens twice. The first conversion is performed between OS and SoC interface, whereas in the second conversion SoC internal variables are mapped to actual memory elements, which is accessed by our proposed hardware accelerator. Figure 5.16 depicts first conversion of embedded Linux generated virtual addresses to SoC compatible memory addresses. In order to store input or output data in a dynamic variable, Linux kernel generates virtual addresses, which later translate to physical addresses with help of memory page tables. These page tables are generated by translation lookaside buffers (TLBs), which are the part of memory-management unit (MMU) of Zynq ARM SoC. It is to mention that TLB is a temporary memory, which is used to reduce access time of user defined arbitrary memory locations. Using TLB and memory page table, a corresponding physical address is generated within the address range defined for each corresponding IP connected as a memory mapped device.

5.5.2 Software Integration

A complete system integration is achieved having semi-custom embedded Linux OS ported on the ARM host along with **PL** providing desired hardware acceleration. The essential components of OS and software-hardware debugging module are discussed in the following sections.

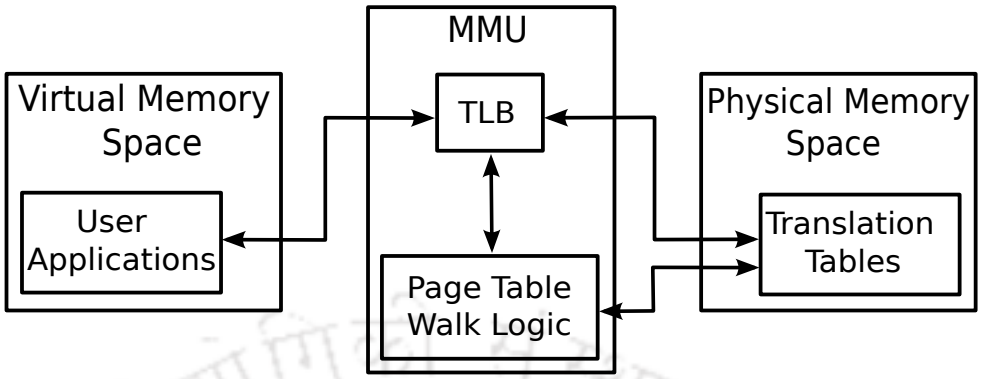


Figure 5.16: Virtual Address to Physical Address Conversion

5.5.2.1 Booting Process

In order to boot Linux on Zynq-ARM SoC, the following files need to be generated and stored in ROM, which are mentioned in Figure 5.17. It is to mention that Yocto project, an open-source Linux operating system [239] is utilized to repackage the semi-custom embedded Linux. The OS is stored in the ROM available on the FPGA board and, and Linux gets booted using this OS. In order to complete the boot process, bootloader is required, which is elaborated below.

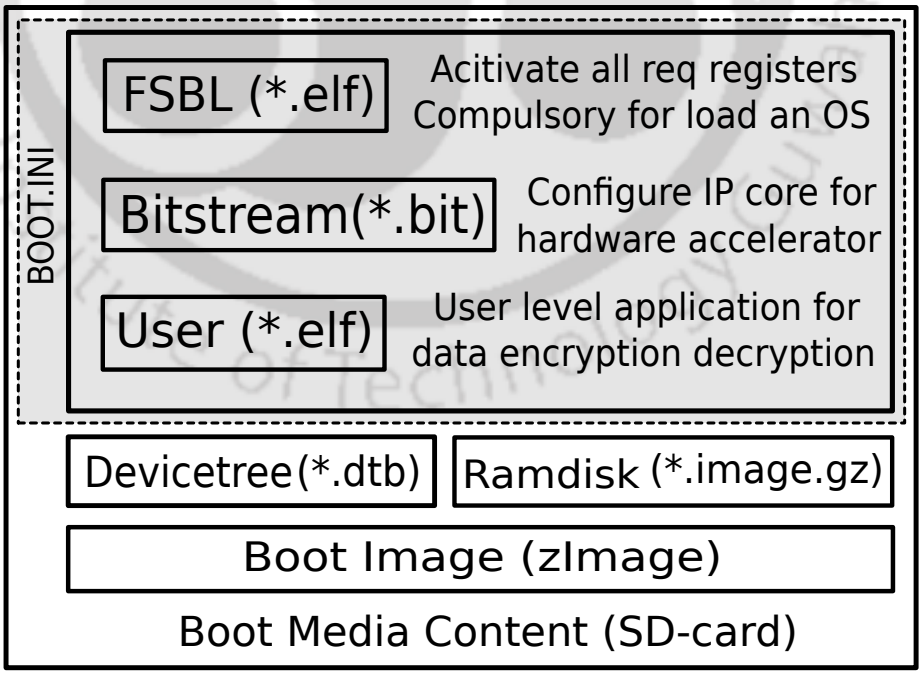


Figure 5.17: Description of Bootloader for software-hardware environment

Bootloaders: Bootloaders are the primary elements of boot media, which set up necessary environment to execute Linux kernel on device. First stage bootloader (FSBL) loads all the internal

registers and components, such as, AXI bus controllers, clocking wizard, MMU, interrupt controller, and other high speed registers etc. in **PS** of an FPGA. It is to mention that the custom FSBL in executable and linkable format (ELF) is generated by Vivado Software Development Kit (SDK) [191]. This FSBL contains Board Support Packages (BSP) provided by Xilinx. Thereafter, *BOOT.INI* can be complied using FSBL and other executable files, and is stored on the ROM before booting an OS. Bitstream generated using an FGPA for a particular hardware IP is also the part of *BOOT.INI*. Further, a second stage bootloader (SSBL) is included in *BOOT.INI*, which contains ARM compatible executable programs for establishing communication between user and FPGA based hardware accelerator. This also enables user to perform encryption, decryption or recryption. It is to be noted that assimilation of FSBL, bitstream and SSBL into *BOOT.INI* facilitates seamless integration of hardware and software on the FPGA board. In order to achieve this goal, *BOOT.INI* is loaded into DDR RAM of **PS** through Xilinx Microprocessor Debugger (XMD) provided by Vivado. Thereafter, it is executed to load Linux kernel on **PS** followed by device tree mapping and to build the file system as shown in Figure 5.17.

Device Tree and File system : Device tree blob file provides information about the physical hardware addresses and components to embedded Linux OS. The RAM disk image file is loaded into DDR memory and creates a temporary disk drive in the RAM available on the FPGA board to mount the root directory.

Connetivity and User Application: After successful boot, Zynq **PS** interfaces, such as Ethernet, USB etc. are established for high speed data connectivity among multiple devices. ARM compatible executable files can be executed using Linux terminal on Zynq SoC for user-defined applications after cross-compiled employing ARM-GCC.

Figure 5.18 showcases step-wise booting process of the custom Yocto based Linux OS on Zynq-ARM SoC. There are two modes for booting OS from ROM, which are power-on-reset mode (POR) and Non-POR mode. POR mode is applicable for either the first time boot on a new device or when the hard reset is required. Non-POR mode boots OS with previously configured registers. In non-POR mode, the debug control and MMU can be modified while booting the OS. After setting initial conditions in the first step, Linux OS image file, bootloaders and other files are transfered from ROM to DDR in the second step. This step ensures that basic kernel is operational, and ramdisk and device tree point to appropriate physical addresses allocated for SoC components. Thereafter, in the

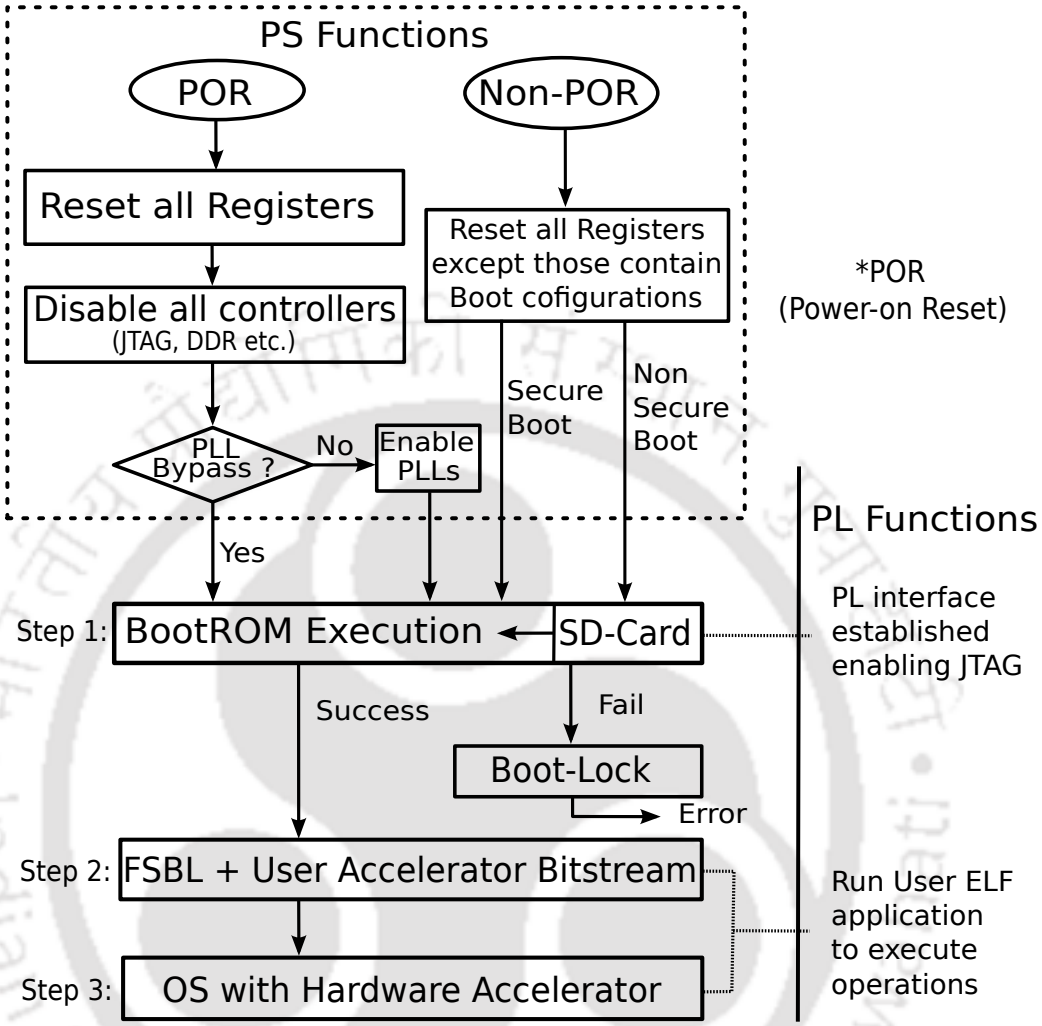


Figure 5.18: Operating System Booting Process on **PS-PL**

third step, FSBL and bitstream are executed to prepare the device to run user applications, such as, encryption or decryption or recryption on **PL**.

5.5.2.2 Debugging Process

We know that debugging is imperative while designing any complex system. Figure 5.19 exhibits the architecture of hardware and software debugging units employed in our proposed design. It is to be noted that debugging unit is required during designing and verification only. It can be observed from Figure 5.19 that there are three access points to debug an overall SoC, which are OS level debugging using UART2 connected at serial terminal, **PS** and **PL** debugging through UART1 using Xilinx Microprocessor Debugger (XMD) and register level debugging via JTAG1 channel. These interfaces allows us to easily rectify design and timing issues, and to check step by step execution of the overall

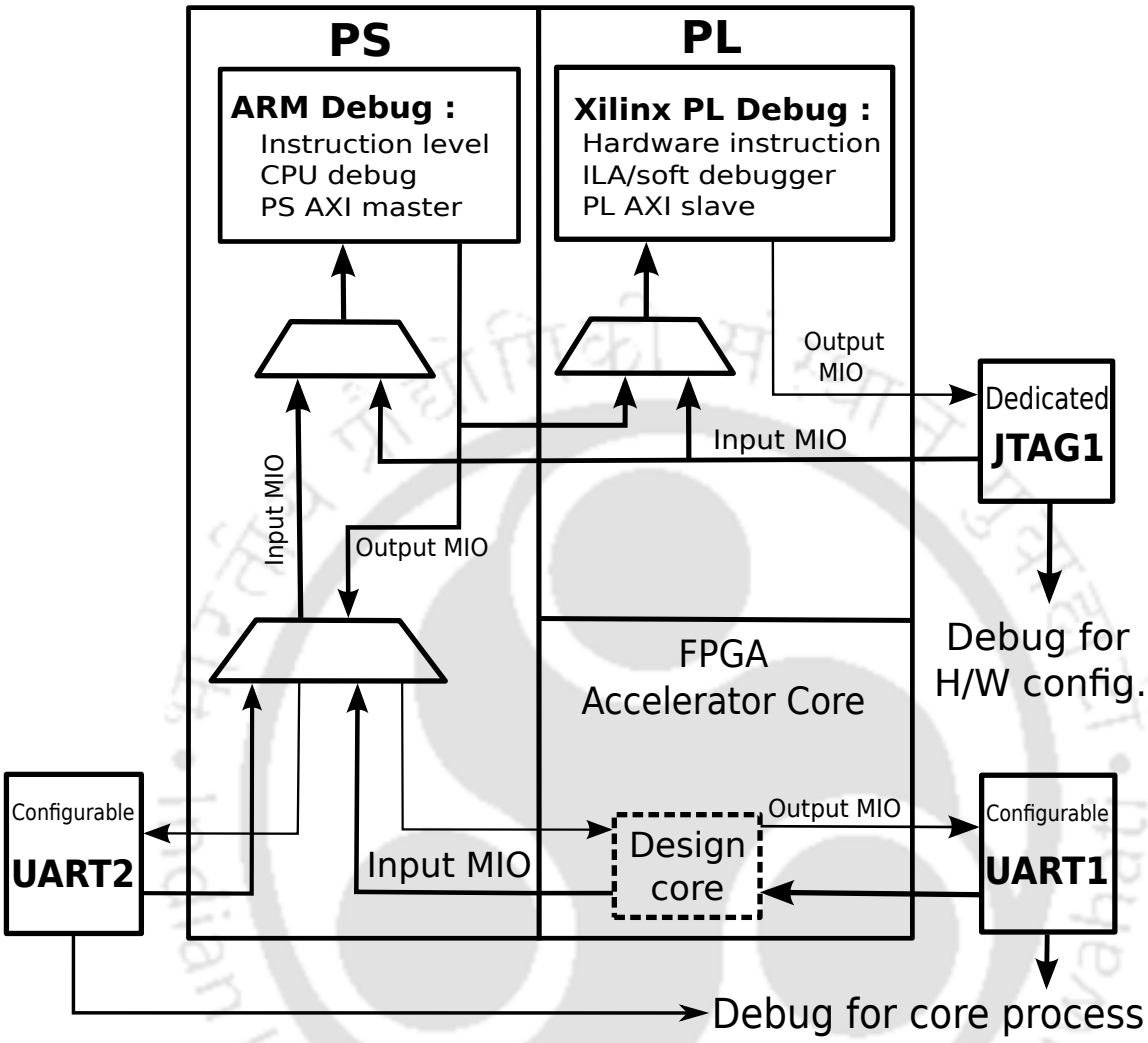


Figure 5.19: Architecture and Components of Debugging Unit

schemes on hardware as well as software.

This concludes detailed overall description of the integration of hardware accelerator with Zynq-ARM SoC and, in the next section, comparison of the proposed PHE algorithms with other contemporary methods are presented.

5.6 Results

The proposed PHE variants are implemented on ZedBoard Zynq-XC7Z020 and Virtex-VC707 FPGA boards. ARM-SoC integration with FPGA is performed on ZedBoard to prove correctness of the proposed schemes. Table 5.3 presents comparative analysis of other contemporary PHE hardware accelerator implementations. From Table 5.3, it is observed that the proposed RLWE PHE variant

Table 5.3: Comparison With Existing Hardware Accelerator Implementations

Work	Scheme	Platform	Op[a]	LUT/REG/DSP/BRAM	Comparative Resource Utilization	Clock (MHz)	Clock Cycle	Delay	Speed (Mbps)[b]
Variant 1	RLWE PHE	Zedboard	E	7622/7058/3/13.5	**1.43 X**	50	344	6.88μs	5.814
			D	6940/7865/3/12.5			809	16.18μs	2.472
			R	938/365/-/-			142	2.84μs	14.084
		Virtex-7	E	7821/7159/3/14	**1.47 X**	200	330	1.65μs	24.242
			D	7114/8085/3/15			876	4.38μs	9.132
			R	980/396/-/-			144	0.72μs	55.556
Variant 2	Lightweight LWE PHE	Zedboard	E	5489/5281/3/5	**X**	50	72	1.44μs	177.778
			D	4621/5514/-/7			48	0.96μs	266.667
			R	375/190/-/2			11	0.22μs	1163.636
		Virtex-7	E	5519/5298/3/6	**~X**	200	70	0.35μs	731.429
			D	4762/5510/-/8			45	0.225μs	1137.778
			R	379/192/-/3			10	0.055μs	4654.545
[214]	BGV RLWE Non-pipelined	Virtex-7	E/D	~	~	150	~	14.54μs	237.69
			R					0.85μs	4065.88
	BGV RLWE pipelined	Ultra-Scale	E/D	527493/133813/165/23.5	**52.71 X**		~	6.84μs	505.26
			R	381068/89849/120/16				0.85μs	4065.88
[224]	FV RLWE	Virtex-6	R	72613/63086/250/84	**6.33 X**	100	~	50ms	804.78
[215]	GH FHE	TSMC 90-nm	E	~	~	666	12M	18.1ms	~
			D				10.7M	16.1ms	
			R				2000M	3.1s	
[216]	Iterative FV	Virtex-7	E	77K/~/952/325.5	**3.64 X**	200	~	1.4μs	2.72
			D					1.24μs	3.08
			R					0.96μs	3.97
	Four-Step FV	Virtex-7	E	67K/~/599/129	**3.15 X**	200	~	1.8μs	2.12
			D					1.8μs	2.12
			R					1.4μs	2.73
[217]	GH-FHE	Virtex-7	E/D	153771/68467/672/~	**10.37 X**	~	~	11ms	~

[a]: Operation; E: Encryption; D: Decryption; R: Recryption; ~: Data not specified; −: Resource not utilized; [b]:(Degree of Polynomial × No. of bits) / Speed.

consumes {7622, 7821} LUTs, {7058, 7159} registers, {3, 3} DSP blocks and {13.5, 14} BRAM for the realization of encryption scheme on ZedBoard and Virtex-7 FPGAs, respectively. Decryption and recryption block of the first variant consumes {6940, 7111} and {938, 980} LUTs, {7865, 8085} and {365, 396} registers, {3, 3} and {0, 0} DSP blocks, {12.5, 15} and {0, 0} BRAM on ZedBoard and Virtex-7 FPGA board, respectively. The throughput of encryption, decryption and recryption in the first PHE variant are {5.814, 2.472 and 14.084} Mbps and {24.242, 9.132 and 55.556} Mbps on ZedBoard and Virtex-7 FPGAs, respectively.

Similarly, the proposed lightweight LWE PHE variant consumes {5489, 5519} LUTs, {5281, 5298} registers, {3, 3} DSP blocks and {5, 6} BRAMs for the encryption scheme on ZedBoard and Virtex-7 FPGA, respectively. The decryption and recryption schemes of this variant consume {4621, 4762} and {375, 379} LUTs, {5514, 5510} and {190, 192} registers, {7, 8} and {2, 3} BRAMs on ZedBoard and Virtex-7 FPGA, respectively. The throughput of encryption, decryption and recryption schemes in the proposed lightweight LWE PHE variant are {177.778, 266.667 and 1163.636} Mbps and {731.429, 1137.778 and 4654.545} Mbps on ZedBoard and Virtex-7 FPGA, respectively. It can be observed that the proposed lightweight LWE PHE variant is 43% more area efficient on Virtex-7 FPGA as compared to the proposed first variant. This leads to improve average delay of the proposed lightweight LWE PHE variant by 10.71× than the average delay of proposed first variant.

It is to mention that the proposed PHE schemes are implemented sequentially and their operations can be parallelized due to their scalable memory mapped architectures. The resource utilization of our lightweight LWE PHE scheme are 52.71× than BGV RLWE based implementation presented in [214]. Further, the overall throughput of our proposed lightweight scheme is 1.43× and 1.29× than the throughputs of non-pipelined and pipelined BGV RLWE scheme, respectively, on Virtex-7 FPGA board. The overall delay of proposed lightweight LWE PHE scheme is 47.51× and 23.06× lesser than the overall delay of non-pipelined and pipelined BGV RLWE scheme, respectively, on Virtex-7 FPGA board.

It can be observed in Table 5.3 that the proposed PHE schemes are better than the other contemporary schemes. Resource consumption of our proposed PHE is lowest among all other contemporary schemes. ASIC implementation of the basis GF PHE method reported in [215] using TSMC 90 nm CMOS technology achieves higher clock frequency, but it takes very long time to perform PHE operations and is slower as compared to our proposed methods. The timing for all the three PHE operations

in [216] are comparable to our proposed method, but throughput and recourse consumption of our proposed method are better than [216]. Finally, the work presented in [217] consumes higher hardware resources and delay. Therefore, it can be summarized that our proposed PHE schemes are resource optimal, delay efficient and presents better throughput than other state-of-the-art methods. The proposed lightweight PHE scheme exhibits best performance among all the methods and it may be used to secure edge enabled low power IoT or embedded applications.

5.7 Conclusion

This chapter presented lattice based fully homomorphic encryption schemes based on RLWE methodology to provide malicious security attacks. These schemes are designed to get integrated with IoT enabled devices and exhibit low power and resource optimal behavior, when implemented in a software-hardware co-design environment using a FPGA accelerator and an ARM-SoC. The best implementation of the proposed PHE scheme is found to be 52.71× more resource efficient than another contemporary method based on BGV RLWE. The cumulative throughput of our proposed scheme is 6523.752 Mbps exhibiting 43% and 29% improvement over non-pipelined and pipelined BGV RLWE schemes, respectively, on Virtex-7 FPGA board. Similarly, the cumulative delay of our proposed PHE scheme developed for IoT enabled hardware applications is 0.63μs, which is 23× lower than BGV RLWE based PHE schemes. Since the proposed RLWE PHE methods are immune even to quantum computer based adversary attacks and have lowest area and delay footprints, these schemes are the most suitable candidates not only to provide desired security to edge enabled IoT devices but also to be employed in the wide range of applications.

6

Conclusion and Future Work

Contents

Conclusion

Strong and robust cryptosystems are the backbone for retaining privacy, secure transactions and maintaining overall digital information infrastructure. Due to the leap-frog improvements in the conventional computing capabilities and rise of practical quantum computers, cloud data and network security protected by classical cryptosystems are facing tremendous challenges against various adversaries. Quantum Shor's algorithm can solve large prime number factorization, discrete logarithm problem which causes the reduction in security of the conventional asymmetric key cryptography schemes, such as, RSA, ECC etc. Similarly, Grover's quantum search algorithm can break the standard classical symmetric cryptosystems. Although, by implementing fully quantum cryptosystem, these security vulnerabilities can be addressed, but complete transformation of classical communication medium to quantum medium is practically infeasible. Therefore, in this research work, we proposed post quantum cryptography (PQC) schemes, which are immune against both classical and quantum computer generated crypto attacks, and realized these PQC algorithms on the hardware platform.

The research objective was to propose cryptosystems and implement them on a hardware platform that can resist quantum computer-generated attacks. Therefore, two PQC algorithms belonged two different classes are developed and realized on hardware. Among these two PQC algorithms, while designing a lattice-based encryption scheme, we had to develop custom hardware IP for random number generator and multivariate polynomial modulo multiplication units. Low power, high throughput PRNGs (BluXor and MPCG) are designed and realized on hardware to implement the lattice-based cryptography algorithms efficiently. Instead of employing existing ones, custom PRNGs are created mainly to gain desired throughput resource efficiency and security and acquire new IP cores for PRNGs. Subsequently, the hardware multivariate polynomial multiplication module is the maiden proposed by us, which reduces time and resource complexities many folds. Moreover, custom hardware is very much needed to realize lattice cryptography polynomial modulo operation efficiently.

Based on these two building blocks and a new innovative lattice LWE-based homomorphic algorithm is developed in the research. Two variants of the lattice homomorphic encryption schemes are derived and realized on hardware whose mathematical foundation and security proofs are well established in the literature. For this, the main challenge was to develop a custom efficient hardware accelerator module for lattice LWE PQC, which can be compatible with various software-hardware

SoC platforms. We demonstrated the software-hardware integration for ARM architecture hosting Yocto-based Linux embedded OS on ZedBoard. We are working to develop software and hardware board support packages for other architecture also (RISCV on Polarfire SoC). Finally, outside conventional PQCs, we developed an entirely new cryptosystem based on a chaotic nonlinear map, and this class of cryptosystems is generally named the Baptista-type encryption decryption technique. After examining various chaotic map generators, we chose a multi-pendulum dynamic system (a triple pendulum arrangement is selected) as one of the perfect candidates due to its higher degree of unpredictability, ergodicity, high sensitivity to initial conditions, and parameters. Unlike most chaotic systems, the triple pendulum-based chaotic system has multiple chaotic attractors, making this perfect for key generation and encryption-decryption operations.

Finally, as per the motivation of this research work, we developed two PQCs on the hardware platform with a few essential building blocks. The following sections present chapter-wise the summary of the results presented in the thesis and the future directions and scope of our research.

6.1 Summary of this work

Among various PQC schemes, lattice and chaos based encryption algorithms are developed as the part of current research objective. Two hardware compatible and secure PQCs are proposed in this work, viz., chaos based PQC and lattice LWE based PHE on both software and hardware platforms. For realizing these algorithms on the hardware platforms efficiently, two custom PRNGs (BluXor and MPCG) and a polynomial modulo multiplier are developed. BluXor and MPCG are implemented on ZedBoard Zynq$^{TM}-7000$ FPGA platform. The quality of the randomness of the proposed PRNGs are validated using Diehard battery suite and NIST SP $800-22$ suite. These PRNGs are also compared with AES, KISS, MT19937, LCG, Modular exponentiation, BBS and Xorshift PRNGs. It is observed that MPCG can be used in low-power and area-constrained applications, such as light-weighted security applications for portable stand-alone devices, whereas, BluXor can be utilized in general purpose cryptographic applications.

A tensor based multivariate polynomial modulo multiplication is proposed as another building block. The proposed multiplier is embarrassingly parallel and scalable, and has linear and quadratic time complexities for single variable and multivariate polynomial multiplications, respectively. The chip area, power and delay estimated of the proposed multiplier are $50113.9\mu m^2$, $1.485\ W$ and $8.812\ ns$,

respectively for 7-degree and 2-variable polynomial expressions. This multiplier is successfully validated using LWE based lattice multivariate homomorphic encryption scheme to measure its efficacy.

The first proposed PQC algorithm is a novel symmetric key cryptography scheme based on a nonlinear chaotic map generated utilizing triple pendulum dynamic system. The detailed analysis and standard tests of the proposed method proves its effectiveness while compared with various chaos based schemes over standard security parameters. The design methodology and mathematical analysis of the chaos generator are presented in this thesis, which include bifurcation diagrams and Lyapunov exponent test, especially to showcase its chaotic property. The generated chaotic map is ratified as a cryptographically secure through various standard benchmark tests, viz., sensitivity to parametric and to initial values, ergodicity, collision test, NIST randomness test etc. The proposed chaotic generator is utilized to develop a symmetric key encryption scheme and is realized on FPGA platform, whose overall power consumption, delay and throughput are 0.186 W, 1.592 ns and 2396.164 MBps, respectively. The proposed chaos based PQC core occupies 0.20374 mm^2 die area and consumes 61.88 mW of power when implemented using SCL 180nm Bulk CMOS technology at 250MHz. It is found that the power consumption and resource utilization of the encryption scheme are less and has better throughput as compared to the other best known contemporary methods.

The two variants of lattice based partially homomorphic (PHE) encryption schemes derived from ring learning with errors (RLWE) hidden subspace membership (HSP) methodology are presented. Lattice schemes are designed to get integrated with IoT enabled devices and exhibit low power and resource optimal behavior, when implemented in a software-hardware co-design environment using an FPGA accelerator and an ARM-SoC. The best implementation of the proposed PHE scheme is found to be $\sim 53\times$ more resource efficient than a contemporary method based on BGV RLWE. The cumulative throughput of the proposed scheme is 6523.752 Mbps exhibiting 43% and 29% improvement over non-pipelined and pipelined BGV RLWE schemes, respectively, on FPGA. Similarly, the cumulative delay of our proposed PHE scheme developed for IoT enabled hardware applications is lower than other PHE schemes. Since the proposed RLWE PHE schemes have lowest area and delay footprints, these schemes are the most suitable candidates for providing desired security to edge enabled IoT devices and can be employed in various data security applications.

6.2 Directions for future work

From the summary of the work presented above, it can be observed that the thesis is focused on proposing alternate security solutions in post quantum cryptography (PQC) domain and their efficient hardware implementations on FPGA board and as an ASIC using SCL 180nm Bulk CMOS technology. Some of the potential directions to which the contribution of the thesis can be extended are discussed below.

- Physically uncloneable functions (PUFs) based seed generators can be used during seed generation step of our proposed PRNGs to further enhance their performance. PUFs are proven to be very fast in terms of preliminary random sequence generation, which are efficient and can be realized optimally on the hardware platforms.

- In the current research work, chaos triple pendulum based symmetric encryption system is proposed. The chaotic maps can further be utilized to develop asymmetric public-private key cryptosystem. Besides that, the chaotic triple pendulum methodology can be extended to chaotic n-pendulum system to introduce higher degree of randomness while generating the chaotic map.

- Software support package is an important step of IP integration. In the current work, only ARM integration is addressed, which may be upgraded to generic IP. Therefore, full board support package (BSP) for generic purpose can be developed for various cross platform seamless software-hardware integration.

- Data driven IoT based smart technologies are revolutionizing the healthcare domain. Therefore, to ensure patients' biomedical data, such as, ECG, EEG etc., and medical records within a connected health system can be securely transmitted and stored by employing the proposed chaos PQC encryption techniques. For biomedical data especially multi-channel ECG data can be encrypted by a key bound chaotic map to secure sensitive personal data.

- Community power distribution and trading network utilizes classical symmetric or asymmetric encryption protocols for secure data transfer. Here, the proposed lattice LWE based homomorphic PQC algorithms can be adapted as a reliable alternative of tradition cryptosystems to ensure robust security against the both classical and quantum computer generated attacks.

List of Publications

Journal Publications

1. **Bikram Paul**, A. Nath, S. Krishnaswami, J. Pidanič, Z. Němec, and G. Trivedi, "*Tensor Based Multivariate Polynomial Modulo Multiplier for Cryptographic Applications*", IEEE Transaction on Computers. (DOI:10.1109/TC.2022.3215638)

2. **Bikram Paul**, T. K. Yadav, B. Singh, S. Krishnaswami, and G. Trivedi, "*A Resource Efficient Software-Hardware Co-Design of Lattice-Based Homomorphic Encryption Scheme on the FPGA*", in IEEE Transaction on Computers. (DOI:10.1109/TC.2022.3198628)

3. **Bikram Paul**, S. Pal, A. Agarwal, and G. Trivedi, "*Triple Pendulum Based Nonlinear Chaos Generator and its Applications in Cryptography*", in IEEE Access. (DOI:10.1109/ACCESS.2022.3226515)

4. **Bikram Paul**, G. Awasthi, A. S. Gupta, V. Alapati, H. S. Shekhawat, J. Pidanič, Z. Němec, and G. Trivedi, "*FPGA-based Real-time Drone Detection using Power and Resource Efficient Machine Learning Algorithms*", in IEEE Access. (Under Review)

5. **Bikram Paul**, M. Janveja, and G. Trivedi, "*Chaos Based Secure and Energy Efficient ECG Data Transmission for ECG Application Targeted to Smart Healthcare Devices*", in IEEE Transactions on Circuits and Systems II: Express Briefs. (Under Review)

6. P. Tiwari, **Bikram Paul**, P. Mahanta, and G. Trivedi, "*A Novel Low-power Encryption Scheme based on Chaotic Dynamic Triple-pendulum System in Residential Network for Smart-grid*", in IEEE SmartGrids.(Under Review)

Conference Publications

1. Sunil Dutt, **Bikram Paul**, Anshu Chawhan, Sukumar Nandi and Gaurav Trivedi, "*ApproxHash: Delay, Power and Area Optimized Approximate Hash Functions for Cryptography Applications*", 10th International Conference on Security Of Information And Networks (SIN 2017), Jaipur, INDIA.

2. **Bikram Paul**, Apratim Khobragade, Javvaji Sai Soumith, Sushree Sila P. Goswami, Sunil Dutt, Gaurav Trivedi, "*Design and Implementation of Low-power High-throughput PRNGs for*

Security Applications", 32th International Conference on VLSI Design (VLSID 2019). New Delhi, INDIA.

3. **Bikram Paul**, Gaurav Trivedi, Pidanič Jan and Zdeněk Němec, "*Efficient PRNG Design and Implementation for Various High Throughput Cryptographic and Low Power Security Applications*", 29th International Conference Radioelektronika (MAREW 2019), Pardubice, Czech-Republic.

4. S. S. P. Goswami, **Bikram Paul**, S. Dutt and G. Trivedi, "*Comparative Review of Approximate Multipliers*", 2020 30th International Conference Radioelektronika (RADIOELEKTRONIKA), Bratislava, Slovakia, 2020.

5. Meenali Janveja, **Bikram Paul**, Gonella Vijayakanthi, Astha Agrawal, Gaurav Trivedi, Pidanic Jan and Zdenek Nemec, "*Design of Efficient AES Architecture for Secure ECG Signal Transmission for Low-power IoT Applications*", 2020 30th International Conference Radioelektronika (RADIOELEKTRONIKA), Bratislava, Slovakia, 2020.

6. Praveen Tiwari, Maj Sunil Kumar Panwar, **Bikram Paul**, Pinakeswar Mahanta and Gaurav Trivedi, "*GUI based Secure Architecture Design for Distributed Community Micro-Grid*", Springer-ESIC-2020, March 2020.

7. **Bikram Paul**, Nupur Choudhury, Eeshankur Saikia, and Gaurav Trivedi, "*Digital Boolean Logic Equivalent Reversible Quantum Gates Design*", CIS-2022, September 2022.

8. **Bikram Paul**, and Gaurav Trivedi, "*Post Quantum Cryptography Algorithms: A Review and Applications*", ICIT-2022, December 2022.

9. **Bikram Paul**, Meenali Janveja, Lakshita Singhal, K. Pavan Kumar and Gaurav Trivedi, "*Hardware Accelerator for Secure ECG-data Transmission for IoT Healthcare Applications*", 36th VLSI Design (VLSID), Hyderabad, India 2023.

Patents

1. **Bikram Paul**, Gaurav Trivedi, Arpan Vyas, Rishav Karki, Rahul Khamkar, Ben Thomas, "*MIPS architecture based Multicore Honeycomb structure inspired AI processor design for various application*", Indian patent filed on March 2019, application no. 2018831018596, Published on 22/11/2019 Journal number: 47/2019

Bibliography

[1] D. Bernstein, J. Buchmann, and E. Dahmen, *Post-Quantum Cryptography*. Springer Berlin Heidelberg, 2009. [Online]. Available: https://books.google.co.in/books?id=VB598lO47NAC

[2] R. J. Hughes, "Cryptography, quantum computation and trapped ions," *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 356, no. 1743, pp. 1853–1868, 1998.

[3] M. R. Albrecht, P. Farshim, J.-C. Faugere, and L. Perret, "Polly cracker, revisited," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2011, pp. 179–196.

[4] K. Roy, "A survey on post-quantum cryptography for constrained devices," *International Journal of Applied Engineering Research*, vol. 14, pp. 2608–2615, 05 2019.

[5] A. Saraswat, C. Khatri, Sudhakar, P. Thakral, and P. Biswas, "An extended hybridization of vigenere and caesar cipher techniques for secure communication," *Procedia Computer Science*, vol. 92, pp. 355–360, 2016, 2nd International Conference on Intelligent Computing, Communication and Convergence, ICCC 2016, 24-25 January 2016, Bhubaneswar, Odisha, India. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050916316465

[6] N. Agrawal, M. Kumar, and M. A. Rizvi, "Transposition cryptography algorithm using tree data structure," in *International Conference on Information Communication and Embedded Systems (ICICES2014)*, 2014, pp. 1–6.

[7] A. Al-Sabaawi, "Cryptanalysis of vigenère cipher: Method implementation," in *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, 2020, pp. 1–4.

[8] J. C. Das and D. De, "Atbash cipher design for secure nanocommunication using qca," *Nanomaterials and Energy*, vol. 6, no. 1, pp. 36–47, 2017.

[9] P. Meng, L. Hang, W. Yang, and Z. Chen, "Attacks on translation based steganography," in *2009 IEEE Youth Conference on Information, Computing and Telecommunication*, 2009, pp. 227–230.

[10] R. Morris, "The hagelin cipher machine (m-209) reconstruction of the internal settings," *Cryptologia*, vol. 2, no. 3, pp. 267–289, 1978.

[11] M. Campbell, "Uncrackable codes: The second world war's last enigma," *New Scientist*, vol. 210, no. 2813, p. 44, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262407911612025

[12] D. Kahn, "The international conference on ultra," *The Journal of Military History*, vol. 43, no. 2, p. 97, 1979.

[13] D. W. Davies, "The lorenz cipher machine sz42," *Cryptologia*, vol. 19, no. 1, pp. 39–61, 1995.

[14] H. Williams, "A modification of the rsa public-key encryption procedure (corresp.)," *IEEE Transactions on Information Theory*, vol. 26, no. 6, pp. 726–729, 1980.

[15] J. Botes and W. Penzhorn, "Public-key cryptosystems based on elliptic curves," in *1993 IEEE South African Symposium on Communications and Signal Processing*, 1993, pp. 1–5.

[16] R. Davis, "The data encryption standard in perspective," *IEEE Communications Society Magazine*, vol. 16, no. 6, pp. 5–9, 1978.

[17] X. Zhang and K. Parhi, "High-speed vlsi architectures for the aes algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 9, pp. 957–967, 2004.

[18] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[19] F. Petitcolas, "La cryptographie militaire," *J. des Sci. Militaires*, vol. 9, pp. 161–191, 1883.

[20] A. I. M. P. Ltd. (2019) Mosca's inequality and its effect on quantum cryptography. [Online]. Available: https://analyticsindiamag.com/moscas-inequality-and-its-effect-on-quantum-cryptography/

[21] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467–488, 1982. [Online]. Available: http://dx.doi.org/10.1007/BF02650179

[22] M. Hirvensalo, *Quantum Computing*. Dordrecht: Springer Netherlands, 2013, pp. 1922–1926. [Online]. Available: http://dx.doi.org/10.1007/978-1-4020-8265-8_1230

[23] W. J. Paul, N. Pippenger, E. Szemeredi, and W. T. Trotter, "On determinism versus non-determinism and related problems," in *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 1983, pp. 429–438.

[24] L. Gurvits, "Classical deterministic complexity of edmonds' problem and quantum entanglement," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, 2003, pp. 10–19.

[25] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, no. 1, p. 147, 1996.

[26] Y. Takahashi, "Quantum arithmetic circuits: a survey," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 92, no. 5, pp. 1276–1283, 2009.

[27] S. Beauregard, G. Brassard, and J. M. Fernandez, "Quantum arithmetic on galois fields," *arXiv preprint quant-ph/0301163*, 2003.

[28] D. P. DiVincenzo, "The physical implementation of quantum computation," *Fortschritte der Physik: Progress of Physics*, vol. 48, no. 9-11, pp. 771–783, 2000.

[29] F. Schmidt-Kaler, H. Häffner, M. Riebe, S. Gulde, G. P. Lancaster, T. Deuschle, C. Becher, C. F. Roos, J. Eschner, and R. Blatt, "Realization of the cirac–zoller controlled-not quantum gate," *Nature*, vol. 422, no. 6930, pp. 408–411, 2003.

[30] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio, "Optimal local implementation of nonlocal quantum gates," *Physical Review A*, vol. 62, no. 5, p. 052317, 2000.

[31] O. Goldreich, *Foundations of Cryptography, Basic Applications*. Cambridge University Press, 2004, vol. 2.

[32] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, p. 14841509, 1997.

[33] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, p. 120126, 1978.

[34] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[35] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.

[36] V. Namias, "The fractional order fourier transform and its application to quantum mechanics," *IMA Journal of Applied Mathematics*, vol. 25, no. 3, p. 241, 1980.

[37] H. Cohen, *A course in computational algebraic number theory*. New York, NY, USA: Springer, 1993.

[38] L. A, K, L. J. H, W, M. M, S, and J. M. Pollard, "The number field sieve," 1980, pp. 564–572.

[39] L. A, K and L. H, W, "The development of the number field sieve," *Lecture Notes in Mathematics*, vol. 1544, 1993.

[40] M. Mosca and A. Ekert, *The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 174–188.

[41] E. Mark, H. Peter, and K. Emanuel, "The quantum query complexity of the hidden subgroup problem is polynomial," *Information Processing Letters*, vol. 91, no. 1, pp. 43–48, 2004.

[42] S. Hallgren, "Polynomial-time quantum algorithms for pells equation and the principal ideal problem," *Journal of the ACM*, vol. 54, no. 1, pp. 1–19, 2007.

[43] S. Hallgren, R. Alexander, and T.-S. Amnon, "Normal subgroup reconstruction and quantum computation using group representations," *SIAM Journal on Computing*, vol. 32, no. 4, pp. 919–934, 2003.

[44] K. Greg, "A subexponential-time quantum algorithm for the dihedral hidden subgroup problem," *SIAM Journal on Computing*, vol. 35, no. 1, pp. 170–188, 2005.

[45] Y. Inui and F. L. Gall, "Efficient quantum algorithms for the hidden subgroup problem over semi-direct product groups," *Quantum Information and Computation*, vol. 7, no. 5&6, pp. 559–570, jul 2007. [Online]. Available: https://doi.org/10.26421%2Fqic7.5-6-9

[46] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 212–219.

[47] E. Borbely, "Grover search algorithm," *CoRR*, vol. abs/0705.4171, 2007.

[48] C. Pomerance, "The number field sieve," in *Proceedings of Symposia in Applied Mathematics*, vol. 48, 1994, pp. 465–480.

[49] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. LNCS 196, pp. 379423, 623656, 1948.

[50] E. T. Jaynes, "Information theory and statistical mechanics," *Phys. Rev.*, vol. 106, pp. 620–630, 1957.

[51] H. Lipmaa, "On optimal hash tree traversal for interval time-stamping," vol. LNCS 2433. Springer, 2002, pp. 357–371.

[52] C. Dods, N. Smart, and M. Stam, "Hash based digital signature schemes. in cryptography and coding," *Phys. Rev.*, vol. LNCS 3796, pp. 96–115, 2005.

[53] R. Merkle, "A certified digital signature. advances in cryptology," vol. LNCS 435. Springer, 1989, pp. 218–238.

[54] J. Buchmann, C. Coronado, E. Dahmen, M. Döring, and E. Klintsevich, "Cmss - an improved merkle signature scheme," *Progress in Cryptology - IN-DOCRYPT 2006*, vol. LNCS 4329, pp. 349–363, 2006.

[55] [Online]. Available: https://www.sav.sk/journals/uploads/0728120003-7-StSt.ps

[56] R. McEliece, "A public key cryptosystem based on algebraic coding theory," *DSN progress report*, vol. 42, no. 44, pp. 114–116, 1978.

[57] N. Sendrier, "On the security of the mceliece public-key cryptosystem," *Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday*, pp. 141–163, 2002.

[58] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Probl. Control and Inform. Theory*, vol. 15, pp. 19–34, 1986.

[59] N. Courtois, M. Finiasz, and N.Sendrier, "How to achieve a mceliece-based digital signature scheme," *In Advances in Cryptology - ASIACRYPT 2001*, vol. 2248, pp. 157–174, 2001.

[60] P. Vron, "Improved identification schemes based on error-correcting codes," *Appl. Algebra Eng. Commun. Comput.*, vol. 8, no. 1, pp. 57–69, 1996.

[61] J. Fischer and J. Stern, "An eficient pseudo-random generator provably as secure as syndrome decoding," *Advances in Cryptology- EUROCRYPT 96*, vol. LNCS 1070, pp. 245–255, 1996.

[62] P. Gaborit, C. Laudaroux, , and N. Sendrier, "Synd: a very fast code-based cipher stream with a security reduction," *IEEE Conference, ISIT07*, pp. 186–190, 2007.

[63] J. Stern, "A new identification scheme based on syndrome decoding," *In Advances in Cryptology - CRYPTO93*, vol. LNCS 773, 1994.

[64] E. Gabidulin, A. Paramonov, and O. Tretjakov, "Ideals over a non-commutative ring and their applications to cryptography," *Proc. Eurocrypt 91*, vol. LNCS 547, 1991.

[65] P. Gaborit, "Shorter keys for code based cryptography," 2005, pp. 81–90.

[66] V. Sidelnikov, "A public-key cryptosystem based on binary reed-muller codes," vol. 4, no. 3, 1994.

[67] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Probl. Control and Inform. Theory*, vol. 15, pp. 19–34, 1986.

[68] H. Janwa and O. Moreno, "Mceliece public key cryptosystems using algebraic-geometric codes," *Designes, Codes and Cryptography*, vol. 8, pp. 293–307, 1996.

[69] R. McEliece, "A public key cryptosystem based on algebraic coding theory," *DSN progress report*, pp. 42–44,114–116, 1978.

[70] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 197–206. [Online]. Available: http://doi.acm.org/10.1145/1374376.1374407

[71] D. G., C. A., and P. H., "Eap-swift: An efficient authentication and key generation mechanism for resource constrained wsns," *International Journal of Distributed Sensor Networks*, vol. ID 460914, p. 12 pages, 2015. [Online]. Available: http://dx.doi.org/10.1155/2015/460914

[72] O. Goldreich, S. Goldwasser, and S. Halevi, "Public-key cryptosystems from lattice reduction problems," *Advances in cryptology*, vol. LNCS 1294, pp. 112–131, 1997.

[73] P. Nguyen and J. Stern, "Cryptanalysis of the ajtai-dwork cryptosystem," *Advances in cryptology (CRYPTO)*, vol. 1462, pp. 223–242, 1998.

[74] D. Micciancio, "Improving lattice based cryptosystems using the hermite normal form," *Cryptography and Lattices Conference- CaLC 2001*, vol. 2146, pp. 126–145, 2001.

[75] J. Hoffstein, J. Pipher, and J. Silverman, "Ntru: a ring based public key cryptosystem," *Proceedings of ANTS-III*, vol. LNCS 1423, pp. 267–288, 1998.

[76] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Proc. 37th ACM Symp. on Theory of Computing (STOC)*, pp. 84–93, 2005.

[77] A. Kawachi, K. Tanaka, and K. Xagawa, "Multi-bit cryptosystems based on lattice problems," *Public Key Cryptography PKC 2007*, vol. LNCS 4450, pp. 315–329, 2007.

[78] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[79] N. Masuda and K. Aihara, "Cryptosystems with discretized chaotic maps," *Ieee transactions on circuits and systems i: fundamental theory and applications*, vol. 49, no. 1, pp. 28–40, 2002.

[80] H. Zhou and X.-T. Ling, "Problems with the chaotic inverse system encryption approach," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 44, no. 3, pp. 268–271, 1997.

[81] F. Huang and Z.-H. Guan, "A modified method of a class of recently presented cryptosystems," *Chaos, Solitons & Fractals*, vol. 23, no. 5, pp. 1893–1899, 2005.

[82] N. K. Pareek, V. Patidar, and K. Sud, "Discrete chaotic cryptography using external key," *Physics Letters A*, vol. 309, no. 1-2, pp. 75–82, 2003.

[83] L. Kocarev and G. Jakimoski, "Logistic map as a block encryption algorithm," *Physics Letters A*, vol. 289, no. 4-5, pp. 199–206, 2001.

[84] S. Vaidyanathan, A. T. Azar, K. Rajagopal, A. Sambas, S. Kacar, and U. Cavusoglu, "A new hyperchaotic temperature fluctuations model, its circuit simulation, fpga implementation and an application to image encryption," *International Journal of Simulation and Process Modelling*, vol. 13, no. 3, pp. 281–296, 2018.

[85] S. S. Askar, A. A. Karawia, and A. Alshamrani, "Image encryption algorithm based on chaotic economic model," *Mathematical Problems in Engineering*, vol. 2015, p. 341729, Jan 2015. [Online]. Available: https://doi.org/10.1155/2015/341729

[86] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 212–219.

[87] T. Habutsu, Y. Nishio, I. Sasase, and S. Mori, "A secret key cryptosystem by iterating a chaotic map," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991, pp. 127–140.

[88] Z. Kotulski and J. Szczepański, "Discrete chaotic cryptography," *Annalen der Physik*, vol. 509, no. 5, pp. 381–394, 1997.

[89] Z. Kotulski, J. Szczepański, K. Górski, A. Paszkiewicz, and A. Zugaj, "Application of discrete chaotic dynamical systems in cryptographydcc method," *International Journal of Bifurcation and Chaos*, vol. 9, no. 06, pp. 1121–1135, 1999.

[90] J. Fridrich, "Symmetric ciphers based on two-dimensional chaotic maps," *International Journal of Bifurcation and chaos*, vol. 8, no. 06, pp. 1259–1284, 1998.

[91] L. M. Ward and R. L. West, "Modeling human chaotic behavior: Nonlinear forecasting analysis of logistic iteration," *Nonlinear Dynamics, Psychology, and Life Sciences*, vol. 2, no. 4, pp. 261–282, 1998.

[92] Y.-H. Chu and S. Chang, "Dynamical cryptography based on synchronized chaotic systems," *Electronics Letters*, vol. 35, no. 12, pp. 974–975, 1999.

[93] E. Alvarez, A. Fernandez, P. Garcıa, J. Jiménez, and A. Marcano, "New approach to chaotic encryption," *Physics Letters A*, vol. 263, no. 4-6, pp. 373–375, 1999.

[94] A. Sambas, S. Vaidyanathan, E. Tlelo-Cuautle, B. Abd-El-Atty, A. A. A. El-Latif, O. Guilln-Fernndez, Sukono, Y. Hidayat, and G. Gundara, "A 3-d multi-stable system with a peanut-shaped equilibrium curve: Circuit design, fpga realization, and an application to image encryption," *IEEE Access*, vol. 8, pp. 137 116–137 132, 2020.

[95] A. Sambas, S. Vaidyanathan, E. Tlelo-Cuautle, S. Zhang, O. Guillen-Fernandez, Sukono, Y. Hidayat, and G. Gundara, "A novel chaotic system with two circles of equilibrium points: Multistability, electronic circuit and fpga realization," *Electronics*, vol. 8, no. 11, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/11/1211

[96] J. Ding and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," *Conference on Applied Cryptography and Network Security- ACNS 2005*, vol. LNCS 3531, pp. 164–175, 2005.

[97] J. Ding, "A new variant of the matsumoto-imai cryptosystem through perturbation," in *Public Key Cryptography – PKC 2004*, F. Bao, R. Deng, and J. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 305–318.

[98] N. Courtois, L. Goubin, and J. Patarin, "Quartz: Primitive specification (second revised version)," p. 18 pages, 2001. [Online]. Available: https://www.cosic.esat.kuleuven.be/nessie

[99] J. Patarin, "Hidden field equations (hfe) and isomorphisms of polynomials (ip): two new families of asymmetric algorithms," *Advances in Cryptology- EUROCRYPT 1996*, vol. LNCS 1070, pp. 33–48, 1996. [Online]. Available: http://www.minrank.org/hfe.pdf

[100] D. Lazard, *Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 146–156. [Online]. Available: http://dx.doi.org/10.1007/3-540-12868-9-99

[101] S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti, "On the security of supersingular isogeny cryptosystems," in *Advances in Cryptology – ASIACRYPT 2016*, J. H. Cheon and T. Takagi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 63–91.

[102] [Online]. Available: https://www.candc.or.jp/en/2014/group_a.html

[103] J. Tian, J. Lin, and Z. Wang, "Fast modular multipliers for supersingular isogeny-based post-quantum cryptography," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 2, pp. 359–371, 2021.

[104] W. Liu, J. Ni, Z. Liu, C. Liu, and M. ONeill, "Optimized modular multiplication for supersingular isogeny diffie-hellman," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1249–1255, 2019.

[105] Y. Huang, F. Zhang, Z. Liu, and H. Zhang, "An efficient signature scheme from supersingular elliptic curve isogenies," *IEEE Access*, vol. 7, pp. 129 834–129 847, 2019.

[106] M. Ajtai, "The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract)," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: ACM, 1998, pp. 10–19. [Online]. Available: http://doi.acm.org/10.1145/276698.276705

[107] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 333–342. [Online]. Available: http://doi.acm.org/10.1145/1536414.1536461

[108] D. Micciancio, "The hardness of the closest vector problem with preprocessing," *IEEE Transactions on Information Theory*, vol. 47, no. 3, pp. 1212–1215, 2001.

[109] D. Micciancio and S. Goldwasser, *Closest Vector Problem*. Boston, MA: Springer US, 2002, pp. 45–68. [Online]. Available: http://dx.doi.org/10.1007/978-1-4615-0897-7-3

[110] M. C. Kasapbai, "A new chaotic image steganography technique based on huffman compression of turkish texts and fractal encryption with post-quantum security," *IEEE Access*, vol. 7, pp. 148 495–148 510, 2019.

[111] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, p. 120126, 1978.

[112] E. E. May, M. A. Vouk, D. L. Bitzer, and D. I. Rosnick, "An error-correcting code framework for genetic sequence analysis," *Journal of the Franklin Institute*, vol. 341, no. 1-2, pp. 89–109, 2004.

[113] R. Schmitz, "Use of chaotic dynamical systems in cryptography," *Journal of the Franklin Institute*, vol. 338, no. 4, pp. 429–441, 2001.

[114] T. Yang, "A survey of chaotic secure communication systems," *International journal of computational cognition*, vol. 2, no. 2, pp. 81–130, 2004.

[115] L. Kocarev, "Chaos-based cryptography: a brief overview," *IEEE Circuits and Systems Magazine*, vol. 1, no. 3, pp. 6–21, 2001.

[116] T. Habutsu, Y. Nishio, I. Sasase, and S. Mori, "A secret key cryptosystem by iterating a chaotic map," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991, pp. 127–140.

[117] F. James, "A review of pseudo-random number generators," *Computer physics communication*, p. 60, 1990.

[118] G. Marsaglia, "A current view of random number generators," in *Sixteenth Symposium on the Interface, ed. L. Balliard*. Amsterdam: Elsevier, 1985, pp. 3 – 10.

[119] C. Li, Q. Wang, J. Jiang, and N. Guan, "A metastability-based true random number generator on fpga," in *2017 IEEE 12th International Conference on ASIC (ASICON)*, Oct 2017, pp. 738–741.

[120] S. Bhuin, J. Sweeney, S. Pagliarini, A. K. Biswas, and L. Pileggi, "A self-calibrating sense amplifier for a true random number generator using hybrid finfet-straintronic mtj," in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2017, pp. 147–152.

[121] Y. Cao, C. H. Chang, Y. Zheng, and X. Zhao, "An energy-efficient true random number generator based on current starved ring oscillators," in *2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, Oct 2017, pp. 37–42.

[122] I. Reidler, Y. Aviad, M. Rosenbluh, and I. Kanter, "Ultrahigh-speed random number generation based on a chaotic semiconductor laser," *Phys. Rev. Lett.*, vol. 103, p. 024102, Jul 2009. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.103.024102

[123] T. Tuncer and E. Avarolu, "Random number generation with lfsr based stream cipher algorithms," in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2017, pp. 171–175.

[124] J. Eichenauer and J. Lehn, "A non-linear congruential pseudo random number generator," *Statistische Hefte*, vol. 27, no. 1, pp. 315–326, Dec 1986. [Online]. Available: https://doi.org/10.1007/BF02932576

[125] J. Boyar, "Inferring sequences produced by a linear congruential generator missing low-order bits," *Journal of Cryptology*, vol. 1, no. 3, pp. 177–184, Oct 1989.

[126] P. L'Ecuyer, F. Blouin, and R. Couture, "A search for good multiple recursive random number generators," *ACM Trans. Model. Comput. Simul.*, vol. 3, no. 2, pp. 87–98, apr 1993. [Online]. Available: http://doi.acm.org/10.1145/169702.169698

[127] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[128] T. Stojanovski and L. Kocarev, "Chaos-based random number generators-part i: analysis [cryptography]," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 48, no. 3, pp. 281–288, Mar 2001.

[129] "The marsaglia random number cdrom including the diehard battery of tests of randomness," 1995. [Online]. Available: http://stat.fsu.edu/pub/diehard/

[130] "A statistical test suite for random and pseudorandom number generators for cryptographic applications," April 2010. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-22r1a

[131] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM Journal on Computing*, vol. 15, no. 2, pp. 364–383, 1986. [Online]. Available: https://doi.org/10.1137/0215025

[132] G. Marsaglia, "Xorshift rngs," *Journal of Statistical Software, Articles*, vol. 8, no. 14, pp. 1–6, 2003. [Online]. Available: https://www.jstatsoft.org/v008/i14

[133] R. S. Katti, R. G. Kavasseri, and V. Sai, "Pseudorandom bit generation using coupled congruential generators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 3, pp. 203–207, March 2010.

[134] XilinxInc. (2012, August) Zedboard zynq^{TM}-7000. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-elhabt.html;https://reference.digilentinc.com/_media/zedboard:zedboard_ug.pdf

[135] AES_CFBMode. (2017, December) Aes cfb mode cipher feedback. [Online]. Available: https://www.cryptopp.com/wiki/CFB_Mode

[136] M. George and A. Zaman, "The kiss generator," *Technical report, Department of Statistics*, 1993. [Online]. Available: https://ipfs.io/ipfs/wiki/KISS_(algorithm).html

[137] Al-Khaffaf, H. S. M., A. Z. Talib, and R. Abdul Salam, "A study on the effects of noise level, cleaning method, and vectorization software on the quality of vector data." Springer, 2008, pp. 299–309.

[138] M. B. Jacques, F. Xiaole, G. Christophe, and L. Laurent, "Fpga design for pseudorandom number generator based on chaotic iteration used in information hiding application," August 2016, pp. 1–24.

[139] H. Fan and M. A. Hasan, "A new approach to subquadratic space complexity parallel multipliers for extended binary fields," *IEEE Transactions on Computers*, vol. 56, no. 2, pp. 224–233, 2007.

[140] S. Kumar, T. Wollinger, and C. Paar, "Optimum digit serial $gf(2^m)$ multipliers for curve-based cryptography," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1306–1311, 2006.

[141] W. . Huang, C. H. Chang, C. W. Chiou, and S. . Tan, "Non-xor approach for low-cost bit-parallel polynomial basis multiplier over $gf(2^m)$," *IET Information Security*, vol. 5, no. 3, pp. 152–162, 2011.

[142] C. Y. Lee and C. W. Chiou, "Efficient design of low-complexity bit-parallel systolic hankel multipliers to implement multiplication in normal and dual bases of $gf(2^m)$," *IEICE Trans. Fund. Electron. Commun. Comput. Sci.*, vol. E88-A, no. 11, pp. 3169–3179, 2005.

[143] J. . Wang, H. W. Chang, C. W. Chiou, and W. . Liang, "Low-complexity design of bit-parallel dual-basis multiplier over $gf(2^m)$," *IET Information Security*, vol. 6, no. 4, pp. 324–328, 2012.

[144] A. Reyhani-Masoleh and M. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over gf(2^m)," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 945–959, 2004.

[145] Chiou-Yng Lee, Erl-Huei Lu, and Jau-Yien Lee, "Bit-parallel systolic multipliers for $gf(2^m)$ fields defined by all-one and equally spaced polynomials," *IEEE Transactions on Computers*, vol. 50, no. 5, pp. 385–393, 2001.

[146] S. Talapatra, H. Rahaman, and J. Mathew, "Low complexity digit serial systolic montgomery multipliers for special class of $gf(2^m)$," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 5, pp. 847–852, 2010.

[147] L. H. Chen, P. L. Chang, and C. Y. Lee, "Scalable and systolic dual basis multiplier over $gf(2^m)$," *Int. J. Innovat. Comput. Inf. Control*, vol. 7, no. 3, pp. 1193–1208, 2011.

[148] Y. Hua, J.-M. Lin, C. Chiou, C.-Y. Lee, and Y. Liu, "A novel digit-serial dual basis systolic karatsuba multiplier over $gf(2^m)$," *Chinese Journal of Computers*, vol. 23, pp. 80–94, 07 2012.

[149] J. Xie, P. K. Meher, M. Sun, Y. Li, B. Zeng, and Z. H. Mao, "Efficient fpga implementation of low-complexity systolic karatsuba multiplier over $f(2^m)$ based on nist polynomials," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 7, pp. 1815–1825, July 2017.

[150] P. L. Chang, Y. K. Yang, L. H. Chen, F. H. Hsieh, and C. Y. Lee, "Low-complexity dual basis digit serial $gf(2^m)$ multiplier," *Icic Express Letters*, vol. 3, no. 4, pp. 1113–1118, 2009.

[151] Po-Lun Chang, F. Hsieh, L. Chen, and C. Lee, "Efficient digit serial dual basis $gf(2^m)$ multiplier," in *2010 5th IEEE Conference on Industrial Electronics and Applications*, 2010, pp. 166–170.

[152] E. Ozcan and A. Aysu, "High-level synthesis of number-theoretic transform: A case study for future cryptosystems," *IEEE Embedded Systems Letters*, vol. 12, no. 4, pp. 133–136, 2020.

[153] F. Yarman, A. C. Mert, E. ztrk, and E. Sava, "A hardware accelerator for polynomial multiplication operation of crystals-kyber pqc scheme," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1020–1025.

[154] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone, "An implementation for a fast public-key cryptosystem," *J. Cryptology*, vol. 3, pp. 63–79, 1991.

[155] Y.-H. Ying, L. Jim-Min, C. Che-Wun, L. Chiou-Yng, and L. Yong-Huan, "Low space-complexity digit-serial dual basis systolic multiplier over galois field $gf(2^m)$ using hankel matrix and karatsuba algorithm," *IET Information Security*, vol. 7, pp. 75–86(11), June 2013. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/iet-ifs.2012.0227

[156] T. Alshawi, A. Bentrcia, and S. Alshebeili, "Design and low-complexity implementation of matrix-vector multiplier for iterative methods in communication systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 12, pp. 3099–3103, Dec 2015.

[157] M. A. Hasan, A. H. Namin, and C. Negre, "Toeplitz matrix approach for binary field multiplication using quadrinomials," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 3, pp. 449–458, March 2012.

[158] D. Zoni, A. Galimberti, and W. Fornaciari, "Flexible and scalable fpga-oriented design of multipliers for large binary polynomials," *IEEE Access*, vol. 8, pp. 75 809–75 821, 2020.

[159] C. W. Chiou, C. Lee, J. Lin, Y. Yeh, and J. Pan, "Low-latency digit-serial dual basis multiplier for lightweight cryptosystems," *IET Information Security*, vol. 11, no. 6, pp. 301–311, 2017.

[160] A. C. Mert, E. Karabulut, E. ztrk, E. Sava, M. Becchi, and A. Aysu, "A flexible and scalable ntt hardware : Applications from homomorphically encrypted deep learning to post-quantum cryptography," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 346–351.

[161] T. Pöppelmann and T. Güneysu, "Towards practical lattice-based public-key encryption on reconfigurable hardware," in *Selected Areas in Cryptography – SAC 2013*, T. Lange, K. Lauter, and P. Lisoněk, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 68–85.

[162] T. Fritzmann, G. Sigl, and J. Seplveda, "Risq-v: Tightly coupled risc-v accelerators for post-quantum cryptography," Cryptology ePrint Archive, Report 2020/446, 2020, https://ia.cr/2020/446.

[163] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-lwe cryptoprocessor," in *Cryptographic Hardware and Embedded Systems – CHES 2014*, L. Batina and M. Robshaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 371–391.

[164] G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng, "Vpqc: A domain-specific vector processor for post-quantum cryptography based on risc-v architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2672–2684, 2020.

[165] T. Zhang and K. Parhi, "Systematic design of original and modified mastrovito multipliers for general irreducible polynomials," *IEEE Transactions on Computers*, vol. 50, no. 7, pp. 734–749, 2001.

[166] Y. H. Chang, J. S. Wang, and R. C. T. Lee, "Generating all maximal independent sets on trees in lexicographic order," in *Proceedings ICCI '92: Fourth International Conference on Computing and Information*, May 1992, pp. 34–37.

[167] S. Zaks, "Lexicographic generation of ordered trees," *Theoretical Computer Science*, vol. 10, no. 1, pp. 63–82, 1980. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397580900730

[168] V. Holota, I. Kohyt, and T. Golota, "The modification of the lexicographic data-string sorting algorithm," in *Experience of Designing and Applications of CAD Systems in Microelectronics. Proceedings of the VI-th International Conference. CADSM 2001 (IEEE Cat. No.01 EX473)*, 2001, pp. 218–219.

[169] J. Wiedermann, "The complexity of lexicographic sorting and searching," in *Mathematical Foundations of Computer Science 1979*, J. Bečvář, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979, pp. 517–522.

[170] L. Carlitz, "The arithmetic of polynomials in a galois field," *American Journal of Mathematics*, vol. 54, no. 1, pp. 39–50, 1932.

[171] P. Scott, S. Tavares, and L. Peppard, "A fast vlsi multiplier for gf(2^m)," *IEEE Journal on selected Areas in Communications*, vol. 4, no. 1, pp. 62–66, 1986.

[172] B. Singh and M. U. Siddiqi, "Multivariate polynomial products over modular rings using residue arithmetic," *IEEE transactions on signal processing*, vol. 43, no. 5, pp. 1310–1312, 1995.

[173] U. Dowerah and S. Krishnaswamy, "A new multiplication technique for lwe based fully homomorphic encryption," *IEEE Letters of the Computer Society*, vol. 3, no. 2, pp. 62–65, 2020.

[174] S. Ivasiev, M. Kasianchuk, I. Yakymenko, R. Shevchuk, M. Karpinski, and O. Gomotiuk, "Effective algorithms for finding the remainder of multi-digit numbers," in *2019 9th International Conference on Advanced Computer Information Technologies (ACIT)*, 2019, pp. 175–178.

[175] P. Chen, S. N. Basha, M. Mozaffari-Kermani, R. Azarderakhsh, and J. Xie, "Fpga realization of low register systolic all-one-polynomial multipliers over $gf(2^m)$ and their applications in trinomial multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp. 725–734, 2017.

[176] M. Cenk, M. A. Hasan, and C. Negre, "Efficient subquadratic space complexity binary polynomial multipliers based on block recombination," *IEEE Transactions on Computers*, vol. 63, no. 9, pp. 2273–2287, 2014.

[177] J.-H. Guo and C.-L. Wang, "Systolic array implementation of euclid's algorithm for inversion and division in gf(2^m)," *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1161–1167, 1998.

[178] H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in gf(2^m)," *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 1010–1015, 1993.

[179] M.-S. Kang and K.-H. Lee, "Hardware implementation of fast division algorithm for gf(2^m)," in *2006 8th International Conference Advanced Communication Technology*, vol. 1, 2006, pp. 117–119.

[180] V. Xilinx, "Design suite," 2018. [Online]. Available: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

[181] W. Mao and G. J. Milne, "An automated proof technique for finite-state machine equivalence," in *Computer Aided Verification*, K. G. Larsen and A. Skou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 233–243.

[182] E. Karl, Z. Guo, Y. Ng, J. Keane, U. Bhattacharya, and K. Zhang, "The impact of assist-circuit design for 22nm sram and beyond," in *2012 International Electron Devices Meeting*, 2012, pp. 25.1.1–24.1.4.

[183] F. Dinechin and B. Pasca, "Large multipliers with less dsp blocks," 08 2009.

[184] U. Dowerah and S. Krishnaswamy, "A new symmetric key homomorphic encryption scheme," in *23rd International Symposium on Mathematical Theory of Networks and Systems*, 2018, pp. 885–892.

[185] J. Awrejcewicz, G. Kudra, and G. Wasilewski, "Experimental and numerical investigation of chaotic regions in the triple physical pendulum," *Nonlinear Dynamics*, vol. 50, no. 4, pp. 755–766, Dec 2007. [Online]. Available: https://doi.org/10.1007/s11071-007-9235-0

[186] "Matlab scientific programming tool," 2019. [Online]. Available: https://in.mathworks.com/products/matlab.html

[187] S. L. de Souza and I. L. Caldas, "Calculation of lyapunov exponents in systems with impacts," *Chaos, Solitons & Fractals*, vol. 19, no. 3, pp. 569–579, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960077903001309

[188] L. Li, "Bifurcation and chaos in a discrete physiological control system," *Applied Mathematics and Computation*, vol. 252, pp. 397–404, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300314016403

[189] K. Ramasubramanian and M. Sriram, "A comparative study of computation of lyapunov spectra with different algorithms," *Physica D: Nonlinear Phenomena*, vol. 139, no. 1-2, pp. 72–86, 2000.

[190] D. Bernstein, J. Buchmann, and E. Dahmen, "Post-quantum cryptography springer-verlag," *Berlin Heidelberg*, 2009.

[191] "Xilinx vivado eda tool," 2012. [Online]. Available: https://bit.ly/2MpTxeP

[192] A. Pérez-Resa, M. Garcia-Bosque, C. Sánchez-Azqueta, and S. Celma, "Chaotic encryption applied to optical ethernet in industrial control systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 12, pp. 4876–4886, 2019.

[193] ——, "Physical layer encryption for industrial ethernet in gigabit optical links," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 4, pp. 3287–3295, 2019.

[194] R. Agbeyibor, J. Butts, M. Grimaila, and R. Mills, "Evaluation of format-preserving encryption algorithms for critical infrastructure protection," in *International Conference on Critical Infrastructure Protection*. Springer, 2014, pp. 245–261.

[195] Z. Hua, B. Zhou, and Y. Zhou, "Sine chaotification model for enhancing chaos and its hardware implementation," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 2, pp. 1273–1284, 2019.

[196] S. Kalanadhabhatta, D. Kumar, K. K. Anumandla, S. A. Reddy, and A. Acharyya, "Puf-based secure chaotic random number generator design methodology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 7, pp. 1740–1744, 2020.

[197] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Advances in Cryptology – CRYPTO 2011*, P. Rogaway, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 505–524.

[198] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun, "Batch fully homomorphic encryption over the integers," in *Advances in Cryptology – EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 315–335.

[199] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Advances in Cryptology – CRYPTO 2011*, P. Rogaway, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 487–504.

[200] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Advances in Cryptology – EUROCRYPT 2011*, K. G. Paterson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 129–148.

[201] C. Gentry, S. Halevi, and N. P. Smart, "Better bootstrapping in fully homomorphic encryption," in *Public Key Cryptography – PKC 2012*, M. Fischlin, J. Buchmann, and M. Manulis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–16.

[202] ——, "Fully homomorphic encryption with polylog overhead," in *Advances in Cryptology – EUROCRYPT 2012*, D. Pointcheval and T. Johansson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 465–482.

[203] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography – PKC 2010*, P. Q. Nguyen and D. Pointcheval, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 420–443.

[204] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology – EUROCRYPT 2010*, H. Gilbert, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–43.

[205] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 868–886.

[206] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, July 2014. [Online]. Available: https://doi.org/10.1145/2633600

[207] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[208] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 75–92.

[209] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 8493.

[210] ——, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, Sep 2009.

[211] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Annual Cryptology Conference*. Springer, 2012, pp. 868–886.

[212] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachne, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *ASIACRYPT (1)*. Springer, 2016, pp. 3–33.

[213] L. Ducas and D. Micciancio, "Fhew: Bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology – EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 617–640.

[214] Y. Su, B. Yang, C. Yang, and L. Tian, "Fpga-based hardware accelerator for leveled ring-lwe fully homomorphic encryption," *IEEE Access*, vol. 8, pp. 168 008–168 025, 2020.

[215] Y. Dorz, E. ztrk, and B. Sunar, "Accelerating fully homomorphic encryption in hardware," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1509–1521, 2015.

[216] A. C. Mert, E. ztrk, and E. Sava, "Design and implementation of encryption/decryption architectures for bfv homomorphic encryption scheme," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 353–362, 2020.

[217] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Optimised multiplication architectures for accelerating fully homomorphic encryption," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2794–2806, 2016.

[218] T. Morshed, M. M. A. Aziz, and N. Mohammed, "Cpu and gpu accelerated fully homomorphic encryption," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 142–153.

[219] E. ztrk, Y. Dorz, E. Sava, and B. Sunar, "A custom accelerator for homomorphic encryption applications," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 3–16, 2017.

[220] D. B. Cousins, J. Golusky, K. Rohloff, and D. Sumorok, "An fpga co-processor implementation of homomorphic encryption," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, 2014, pp. 1–6.

[221] C. Jayet-Griffon, M.-A. Cornelie, P. Maistri, P. Elbaz-Vincent, and R. Leveugle, "Polynomial multipliers for fully homomorphic encryption on fpga," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2015, pp. 1–6.

[222] J.-H. Ye and M.-D. Shieh, "Low-complexity vlsi design of large integer multipliers for fully homomorphic encryption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1727–1736, 2018.

[223] C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "Accelerating integer-based fully homomorphic encryption using comba multiplication," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, 2014, pp. 1–6.

[224] S. Sinha Roy, K. Jrvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, "Hepcloud: An fpga-based multicore processor for fv somewhat homomorphic function evaluation," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1637–1650, 2018.

[225] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat, "Hardware/software co-design of an accelerator for fv homomorphic encryption scheme using karatsuba algorithm," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 335–347, 2018.

[226] O. Regev, "Lattice-based cryptography," *Advances in cryptology (CRYPTO)*, pp. 131–141, 2006.

[227] ——, "On lattices, learning with errors, random linear codes, and cryptography," *Proc. 37th ACM Symp. on Theory of Computing (STOC)*, pp. 84–93, 2005.

[228] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 333–342.

[229] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehl'e, "Classical hardness of learning with errors," *In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 2013.

[230] C. Schnorr, "A hierarchy of polynomial time lattice basis reduction algorithms," *Theoretical Computer Science*, p. 53, 1987.

[231] D. Micciancio and P. Voulgaris, "A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations," *SIAM Journal on Computing 42(3)*, 2013.

[232] U. Dowerah and S. Krishnaswamy, "A new symmetric key homomorphic encryption scheme," in *23rd International Symposium on Mathematical Theory of Networks and Systems*, 2018, pp. 885–892.

[233] ——, "Homomorphic encryption based on hidden subspace membership," *CoRR*, vol. abs/1910.06270, 2019. [Online]. Available: http://arxiv.org/abs/1910.06270

[234] W. Stein *et al.*, *Sage Mathematics Software (Version 9.2)*, The Sage Development Team, 2020, `http://www.sagemath.org`.

[235] W.-C. Lin, J.-H. Ye, and M.-D. Shieh, "Scalable montgomery modular multiplication architecture with low-latency and low-memory bandwidth requirement," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 475–483, 2014.

[236] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster interleaved modular multiplication based on barrett and montgomery reduction methods," *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1715–1721, 2010.

[237] J. D. Bruguera, "Low latency floating-point division and square root unit," *IEEE Transactions on Computers*, vol. 69, no. 2, pp. 274–287, 2020.

[238] B. Paul, G. Trivedi, P. Jan, and Z. Nmec, "Efficient prng design and implementation for various high throughput cryptographic and low power security applications," in *2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA)*, 2019, pp. 1–6.

[239] *The Yocto Project (Version 9.2)*, The Linux Foundation, 2020, `https://www.yoctoproject.org/`.