Optimized hardware implementations of cryptography algorithms for resource-constraint IoT devices and high-speed applications

A thesis submitted to the
College of Graduate and Postdoctoral Studies (CGPS)
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Electrical &
Computer Engineering
University of Saskatchewan
Saskatoon, Canada

By

Karim Shahbazi

Permission to Use

In presenting this thesis/dissertation in partial fulfillment of the requirements for a Post-graduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis/dissertation in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis/dissertation work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis/dissertation or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis/dissertation.

Disclaimer

Reference in this thesis to any specific commercial products,process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the University of Saskatchewan. The views and opinions of the author expressed herein do not state or reflect those of the University of Saskatchewan, and shall not be used for advertising or product endorsement purposes

Request for permission to copy or to make any other use of material in this thesis in whole or in part should be addressed to:

Head of the Division of Biomedical Engineering
57 Campus Drive
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9

OR

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9 Canada

Acknowledgements

I would like to express my deepest gratitude toward my supervisor, Prof. Seok-Bum Ko, for his invaluable support, guidance, and giving me the freedom I needed through my research program at the University of Saskatchewan. It has indeed been an honor and rewarding experience to work under his supervision and at his Lab. I have learned important lessons on the skills and values of conducting research. Special thanks to all of my lab mates and friends. I had a fantastic experience in our lab 2C60.2 at the University of Saskatchewan. My special thanks to my committee members for their valuable feedback and comments. I would also like to extend my gratitude to my loving wife, my dearest parents, and my caring sisters who have paved this way for me to continue my studies with their endless support and encouragement. I would like to dedicate my Ph.D. thesis to my family members.

Abstract

The advent of technologies, including the Internet and smartphones, has made people's lives easier. Nowadays, people get used to digital applications for e-business, communicating with others, and sending or receiving sensitive messages. Sending secure data across the private network or the Internet is an open concern for every person. Cryptography plays an important role in privacy, security, and confidentiality against adversaries. Public-key cryptography (PKC) is one of the cryptography techniques that provides security over a large network, such as the Internet of Things (IoT).

The classical PKCs, such as Elliptic Curve Cryptography (ECC) and Rivest-Shamir-Adleman (RSA), are based on the hardness of certain number theoretic problems. According to Shor's algorithm, these algorithms can be solved very efficiently on a quantum computer, and cryptography algorithms will be insecure and weak as quantum computers increase in number. Based on NIST, Lattice-based cryptography (LBC) is one of the accepted quantum-resistant public-key cryptography. Different variants of LBC include Learning With Error (LWE), Ring Learning With Error (Ring-LWE), Binary Ring Learning with Error (Ring-Bin LWE), and etc. AES is also one of the secure cryptography algorithm that has been widely used in different applications and platforms. Also, AES-256 is secure against quantum attack.

It is very important to design a crypto-system based on the need and application. In general, each network has three different layers; cloud, edge, and end-node. The cloud and edge layer require to have a high-speed crypto-system, as it is used in high traffic application to encrypt and decrypt data. Unfortunately, most of the end-node devices are resource-constraint and do not have enough area for security guard. Providing end-to-end security is vital for every network. To mitigate this issue, designing and implementing a lightweight cryto-system for resource-constraint devices is necessary.

In this thesis, a high-throughput FPGA implementation of AES algorithm for high-traffic edge applications is introduced. To achieve this goal, some part of the algorithm

has been modified to balance the latency. Inner and outer pipelining techniques and loop-unrolling have been employed. The proposed high-speed implementation of AES achieves a throughput of 79.7Gbps, FPGA efficiency of 13.3 Mbps/slice, and frequency of 622.4MHz. Compared to the state-of-the-art work, the proposed design has improved data throughput by 8.02% and FPGA-Eff by 22.63%.

Moreover, a lightweight architecture of AES for resource-constraint devices is designed and implemented on FPGA and ASIC. Each module of the architecture is specified in which occupied less area; and some units are shared among different phases. To reduce the power consumption clock gating technique is applied. Application specific integrated circuit (ASIC) implementation results show a respective improvement in the area over the previous similar works from 35% to 2.4%. Based on the results and NIST report, the proposed design is a suitable crypto-system for tiny devices and can be supplied by low-power devices.

Furthermore, two lightweight crypto-systems based on Binary Ring-LWE are presented for IoT end-node devices. For one of them, a novel column-based multiplication is introduced. To execute the column-based multiplication only one register is employed to store the intermediate results. The multiplication unit for the other Binary Ring-LWE design is optimized in which the multiplication is executed in less clock cycles. Moreover, to increase the security for end-node devices, the fault resiliency architecture has been designed and applied to the architecture of Binary Ring-LWE. Based on the implementation results and NIST report, the proposed Binary Ring-LWE designs is a suitable crypto-system for resource-constraint devices.

Table of Contents

List of Abbreviations

ADP	Area-Delay-Product
AES	Advanced Encryption Standard
Af	Affine transformation
ARK	Add-RoundKey
ASIC	Application Specific Integrated Circuit
BRAM	Block random access memories
CBC	Cipher block chaining
CFB	Cipher feedback
CTR	Counter
ECB	Electronic codebook
ECC	Elliptic curve cryptography
ENS	Equivalent Number of Slices
FFT	Fast Fourier transform
FPGA	Field-programmable gate array
FPGA-Eff	FPGA efficiency
IDEA	International Data Encryption Algorithm
IoT	Internet of Things
LBC	Lattice-based cryptography

LOF	List of Figures
LOT	List of Tables
LUT	Look-up Table
LWE	Learning with Error
NIST	National Institute of Standards and Technology
NTRU	N-th degree Truncated polynomial Ring Units
NTT	Number Theoretic Transform
OFB	Output Feedback
PCBC	Propagating cipher block chaining
PKC	Public-key cryptography
Ring-BinLWE	Binary Ring Learning With Error
Ring-LWE	Ring Learning With Error
RSA	Rivest-Shamir-Adleman
S-box	Substitution box
SoC	system-on-a-chip
SoPs	Sum of products
VHDL	Very high-speed integrated circuit Hardware Description Language

List of Tables

List of Figures

Part I

Preface

1. Introduction

Cryptography is one of the ancient science that has been used for more than 2000 years ago [4]. Cryptography is the science of using mathematics to encrypt and decrypt data in order to provide a secure transmission across insecure networks so that the data cannot be read by anyone expect the intended recipient. As a result, cryptography plays an important role in each network. To provide the security, cryptography algorithms should be implemented on hardware or software platforms. Each of them has several advantages and disadvantages. For example, execution of cryptography algorithms on software is slower than its hardware implementation.

Hardware implementation includes implementation on Application Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA). This implementation is called crypto-system and used with other part of a design to encrypt and decrypt the data. By employing the hardware resources in FPGA and ASIC, the crypto-system can be designed based on the need and limitations. In general, hardware implementation provides more security against software implementation by accessing the physical layer. Also, by employing the entire hardware resources of FPGA and ASIC, high-speed implementation of crypto-system is achievable (compared to software implementation that is implemented on Graphics Processing Units (GPU) or Central Processing Unit (CPU)).

Most networks include three major layers: cloud; edge devices; and end-nodes devices. Figure 1.1 presents the overall architecture of current advanced IoT networks that are constructed from three major layers [1]. Cloud and some of the edge devices require high-performance and high-speed implementation, such as 64-bit processors and FPGAs.

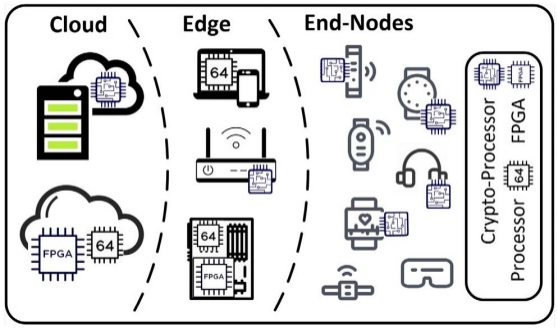


Figure 1.1: Architecture of IoT network and available hardware resources in each layer [1]

Moreover, the end-nodes devices have constrained resources. Most of the end-node devices do not have enough resources for security guard. For each network, providing end-to-end security is vital. The practical solution for resource-constrained devices is lightweight implementation of the crypo-system. The crypto-systems should be designed based on the layer of the network. As cloud and edge layers are high-traffic, they require high-speed crypto-systems. End-node devices are resource-constraint devices; thus, it is necessary to design lightweight crypto-systems for these devices. This motivates the research works to be presented in this thesis that are to design and implement the crypto-system architectures for resource-constraint devices and high-speed applications.

This chapter presents the need and advantage of using cryptography. The motivation of the research works is presented in Section 1.1. The contributions of the research works are summarized in Section 1.2. The overview of the thesis is presented in section 1.3.

1.1 Motivation of Research Works

Internet of Things (IoT) is a vast network in which myriad devices and applications are connected to transmit data. This information could be parts of a smart environment, such as e-health, public transportation, and smart city. These devices could be part of a high-tech network, from sensing technology, communication technology, and data processing to cloud computing and artificial intelligence. As the number of connected devices to IoT has been exponentially increased, providing security for all the transmitted information

in most scenarios is not easy. Moreover, most of the IoT end-node devices are resource-constrained and do not have encryption safeguards. As crypto-systems are used in different applications and platforms, designing an optimized architecture based on the application is very important. Some networks require lightweight crypto-system for their end-node devices to provide end-to-end security. In addition to the security of data transmission in many high-traffic applications, such as high-traffic servers that transfer a huge amount of data, fast encryption/decryption and transmission are also required.

Thus, cryptography plays a vital role in keeping the security in IoT and other transmitting networks. Cryptography algorithms are divided into two different categories, symmetric and asymmetric (public-key cryptography). Asymmetric cryptography algorithms use two keys for encryption and decryption data, such as elliptic curve cryptography (ECC) and Rivest–Shamir–Adleman (RSA). Compared with asymmetric cryptography algorithms, the symmetric cryptography algorithms use the same keys for encryption and decryption data, have less computation, and occupy fewer areas, making them very hardware friendly. The security of RSA relies on the hardness of factoring large integer numbers; ECC is based on an elliptic curve discrete logarithm problem. Shor's algorithm [5] can solve these problems very efficiently in a polynomial time using a quantum computer. By emerging quantum computers, most of the traditional crypto-systems will not be secure; thus, there is a tendency to design crypto-systems by quantum-resistance cryptography algorithms.

The advanced encryption standard (AES) is one of the most secure, fast, and most used symmetric algorithms that have a good performance in both software and hardware platforms. AES-256 is secure against quantum computers [6]. Compared with software platforms, hardware implementation of cryptography algorithms, such as application-specific integrated circuit (ASIC) and field-programmable gate arrays (FPGAs), provides higher security and throughput. FPGA implementation has some advantages over ASIC implementation, such as reconfigurability and low design cost. AES is used in broad applications, and different security networks, such as Wi-Fi protected access 2 system, secure sockets layer, automated teller machines, IoT, and digital video recorders.

According to NIST [7], lattice-based cryptography (LBC) is one of the accepted quantum-resistant public-key cryptography that LBC has received much attention in these years and has been selected as a great candidate for post-quantum cryptography system. A lattice $L \in Z^n$is the set of all integer linear combinations of n independent basis vectors b_i. Among the different varieties of LBC techniques, such as N-th degree Truncated polynomial Ring Units (NTRU) and learning with errors (LWE), Ring-LWE [8] is more practical and efficient in hardware and, compared to LWE, has a smaller key size [9] [10]. LBC algorithms, such as Ring-LWE, require a lot of multiplications to encrypt data; thus, modular multiplication is the main module of most of the LBC in terms of occupied area and latency. The Binary Ring-LWE (Ring-Bin LWE) is a variant of Ring-LWE and was introduced in 2016 by [11]. In Binary Ring-LWE, the errors are sampled with binary coefficients. The multiplication is done by shifting and adding operations that make Binary Ring-LWE a suitable crypto-system for resource-constrained devices. Based on the aforementioned, the motivations of the thesis are summarized as:

- Each network includes three layers. To provide end-to-end security, it is vital to design a suitable crypto-system based on each layer of the network. Edge and cloud layers have high-traffic, and require high-speed crypto-system to encrypt/decrypt and send the data. End-node devices are resource-constraint devices, and require lightweight crypto-systems.

- With the advent of quantum computers, most of the current crypto-systems are endangered. It is vital to design new architecture of crypto-systems that are resistant against quantum attacks. LBC is one of the promising method of post-quantum cryptography (PQC). AES is secure against quantum attacks that has been widely used in different applications platforms, and also is used as a random number generator (RNG) for other crypto-systems.

1.2 Contributions

Efficient crypto-systems for IoT applications will be proposed by considering all the requirements mentioned in Section 1.1 for a high-speed and lightweight design. This thesis contains a high-speed design for high-traffic applications, an area-efficient nano-AES for end-node resource-constrained devices, an area and power-efficient crypto-system based on Binary Ring-LWE for IoT resource-constrained devices, and implementation of an efficient and optimized multiplication for Binary Ring-LWE. Moreover in order to increase the security of Binary Ring-LWE, the fault resilient architecture is designed and applied to the Binary Ring-LWE. The main contributions of the proposed research are:

- A lightweight AES architecture for IoT resource-constrained devices is designed. To reduce the required logic, the Shift-Rows is embedded inside the State-Register. An optimized Sub-Bytes block is designed and shared with the key expansion and encryption phases to reduce the area. An optimized 8-bit block for Mix-Columns with 8-bit input and output is designed. To reduce the power, the clock gating technique is applied to the design.

- A high-throughput AES for high-traffic applications is designed. The AES algorithm is modified in which Sub-Bytes and Shift-Rows are exchanged for the first nine rounds; Add-Round-Key and Shift-Rows are merged into one stage. The Sub-Bytes is optimized in which inverse isomorphic and the affine transformation (Af) are combined together. The delay of Sub-Bytes is reduced by inserting registers in optimal places. Also, full loop unrolling, inner (inserting registers between rounds) and outer (inserting registers among functions) pipeline stages are used.

- A lightweight architecture for Binary Ring-LWE is designed. A novel multiplication technique is proposed for the design. The proposed multiplication is column-based, that all coefficients of multiplier and multiplicand are involved in the multiplication in each cycle. Moreover, in the proposed multiplication, the modular reduction and anti-circular rotation are executed one time in each multiplication cycle.

- Fault resiliency is evaluated for three phases of Binary Ring-LWE, key generation, encryption, and decryption. For Binary Ring-LWE, randomization fault does not have any impact over key generation and encryption phases. Skipping is avoiding to run certain operations in the algorithm, such as addition and multiplication. Zeroing contains setting some parts or the entire value of a coefficient to zero. Skipping and zeroing attacks have a high impact on Binary Ring-LWE. As a result, a fault resiliency architecture is designed and applied to the Binary Ring-LWE.

- The multiplication of Binary Ring-LWE is optimized. The location of each coefficient of polynomials in the multiplication in Binary Ring-LWE is evaluated, and an optimized column-based multiplication is introduced. The proposed multiplication method optimizes the column-based multiplication and rotation that requires $\frac{n}{2} \times n$ cycles to execute the multiplication. The values of two consecutive coefficients of matrix are fed to the design. The proposed architecture is designed efficiently based on the proposed method. The register bank of the design for storing the coefficients contains two sub-register banks for better controlling the multiplication and rotation.

1.3 Outline

In this thesis, several new hardware cryptography architectures are designed and implemented. For each of the proposed design, the hardware implementation merits, including execution time, occupied area, and etc. are analyzed in detail. The proposed designs are also compared with related designs available in the literature to show their advantages and improvements. The architecture of crypto-systems are beased on the requirement for lightweight and high-speed applications. The entire thesis is composed of four parts with eight chapters described as follow:

- **Part I** includes two chapters. Chapter 1 (*Introduction*) presents the importance of the works, motivation and contributions of the research works. Chapter 2 (*Background*) introduces the background of cryptography for the proposed research works and the related works.

- **Part II** includes two chapters. In this part, two hardware implementations of AES are presented. In chapter 3, a high-speed implementation of AES for high-traffic application is introduced. Chapter 4 presents a lightweight implementation of AES for resource-constraint devices. As AES algorithm also uses as the random number generator (RNG) for other crypto-systems, the lightweight design of AES is used as RNG for Binary Ring-LWE.

- **Part III** includes three chapters. This part presents the optimized architectures of Binary Ring-LWE. Binary Ring-LWE require a RNG unit. The lightweight AES design of chapter 4 is used as a RNG for Binary Ring-LWE. A lightweight architecture for Binary Ring-LWE is designed in chapter 5. In this chapter, a novel column-based multiplication is introduced. The execution of the design of chapter 5 requires more clock cycles. To reduce the number of clock cycles, the multiplication unit of Binary Ring-LWE of chapter 5 is re-designed. Chapter 7 presents the new architecture of Binary Ring-LWE. To increase the security against the fault injection attack, fault resiliency is evaluated for three phases of Binary Ring-LWE. Then in chapter 6, fault resilient is applied to the design of Binary Ring-LWE.

- **Part IV** contains chapter 8 that includes the summary and conclusion of the thesis and future work.

Below is the list of publications, arranged according to the order of appearance in this thesis:

- Chapter 3: *High Throughput and area-efficient AES implementation*

  - **K. Shahbazi** and S. -B. Ko, "High throughput and area-efficient FPGA implementation of AES for high-traffic applications," in IET Computers & Digital Techniques, vol. 14, no. 6, pp. 344-352, Nov. 2020, doi: 10.1049/iet-cdt.2019.0179.

- Chapter 4: *Area-Efficient Nano-AES Implementation*

- **K. Shahbazi** and S. -B. Ko, “Area-Efficient Nano-AES Implementation for Internet-of-Things Devices,” in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 1, pp. 136-148, Jan. 2021, doi: 10.1109/TVLSI.2020.3033928.

- Chapter 5: *Lightweight design of Binary Ring-LWE*

  - **K. Shahbazi**, and Seok-Bum Ko “Area and power efficient post-quantum cryptosystem for IoT resource-constrained devices.” Microprocessors and Microsystems. Vol. 84, PP 104280, July 2021, doi: 10.1016/j.micpro.2021.104280.

- Chapter 6: *Fault resilient implementation of Binary Ring-LWE*

  - **K. Shahbazi**, and Seok-Bum Ko “Lightweight and CCA2-Secure Hardware Implementation of Binary Ring-LWE.” 2022 IEEE International Symposium on Circuits and Systems (ISCAS).

- Chapter 7: *An Optimized Implementation of Modular Multiplication for Binary Ring-LWE*

  - **K. Shahbazi**, and Seok-Bum Ko “An Optimized Hardware Implementation of Modular Multiplication of Binary Ring LWE.” under review at IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

- Other publications that are not included in this thesis:

  - Y. Wang, **K. Shahbazi**, H. Zhang, K.-Il Oh, J.-J. Lee, S.-B. Ko, “Efficient spiking neural network training and inference with reduced precision memory and computing," *IET Computers & Digital Techniques*, vol. 13, 2019.

2. Background

AES is one of the secure algorithms that has been used in a variety of applications and platforms. The hardware implementation of AES contains a lightweight design for resource-constraint devices and high-throughput implementation. Section 2.1.1 presents the background and a brief explanation of AES, and the previous works related to AES implementation are in Section 2.2.1. On another side, more research has focused on LBC. Binary Ring-LWE does not have complicated and large functions; thus, it can be implemented efficiently on end-node devices. Ring-LWE and its operations are explained in Section 2.1.3. In Section 2.1.4, Binary Ring-LWE will be discussed. Section 2.2.2 presents some hardware implementation of Binary Ring-LWE.

2.1 Introduction to Algorithms

2.1.1 AES Algorithm

AES is a symmetric encryption/decryption algorithm established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [12]. The length of the input data and cipher keys can be 128, 192, or 256 bits. Both encryption and decryption procedures perform several rounds, indicating by N_r, due to the size of input/cipher key blocks that N_r can be 10, 12, and 14 for key sizes 128, 192, and 256, respectively. Each block of data is also called a 'state' that consists of four rows of bytes. Figure 2.1 shows the AES encrypting steps. AES algorithm has four main functions, Add Round Key, Shift Rows, Sub Bytes, and Mix Columns. All four functions are used in every round except the first and the last ones. The first round consists of only Add Round Key, and the last round does not include Mix

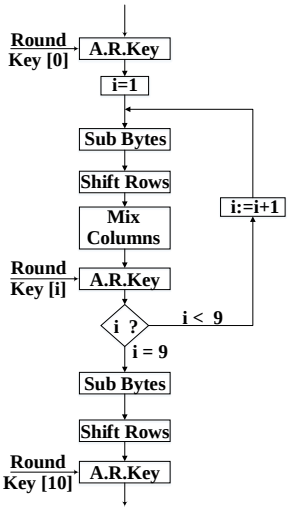


Figure 2.1: Block diagram of AES-128 encryption

Columns. The main functions are briefly described as:

Add Round Key

This function is the addition of input and keys by using bitwise XOR.

Shift Rows

This is a conversion that operates on the rows of a state. The bytes of the state are shifted cyclically to the left. The first row remains untouched. The second, third, and fourth rows are shifted by one, two, and three, respectively.

Substitute Bytes

This function is one of the most critical parts of the AES design regarding power, area, and latency. It performs a byte-by-byte substitution of the blocks to produce a new byte value. The most straightforward way of implementing is using the lookup table (LUT), such as [13] [14], or employing the Boolean simplification map (by using truth table in order to make a direct relation between the parameter of the Sub-Bytes [15] [16]), these two methods occupy more area and are not suitable for area-restricted devices. Decode–Switch–Encode

(DSE), which is employed by [17] [18], is another method for implementing Sub-Bytes that is a good option for low-power architecture; however, it occupies a larger area. The efficient way of implementing Sub-Bytes is to use composite field arithmetic, such as [19] [20] [21]. Sub-Bytes contains calculating the multiplicative inverse of $f(x)$ that is $g(x)$, in which $f(x) \cdot g(x) \bmod (x^8+x^4+x^3+x+1) = 1$ followed by affine transformation (AT). Calculating the multiplicative inverse in $GF(2^8)$ is complicated, in which employing the composite field arithmetic reduces the complexity.

Mix Columns

The Mix Columns transformation operates on the State, column by column with some fixed values. The Mix Columns multiplication is written in equation (2.1). As the Mix Columns has fixed values, one way is to pre-calculate values and store them as a LUT. This method, however, is not suitable because of high area consumption and high latency and is not efficient for area-limited applications. In the equation (2.1), the '.' is a multiplication modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$, and '+' is XOR.

$$\begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \rightarrow \begin{cases} a' = 2.a + 3.b + c + d \\ b' = a + 2.b + 3.c + d \\ c' = a + b + 2.c + 3.d \\ d' = 3.a + b + c + 2.d \end{cases} \tag{2.1}$$

2.1.2 Lattice-based Cryptography

In general, a lattice is a set of points in n-dimensional space with a periodic structure. This n-D space is generated by all combination of independent vectors $b_1, ..., b_n$, these vectors known as a *basis* of the lattice. Figure 2.2 shows a 2-D lattice that is generated by using b_1 and b_2 vectors. In terms of hardware implementation, the size of a computer memory is finite but the lattices is a collection of infinite large objects. Therefore, the term basis of lattice is used to solve the memory limitation issue by representing finite lattices in a concise way. A basis of lattice is a collection of small vectors that can be used to

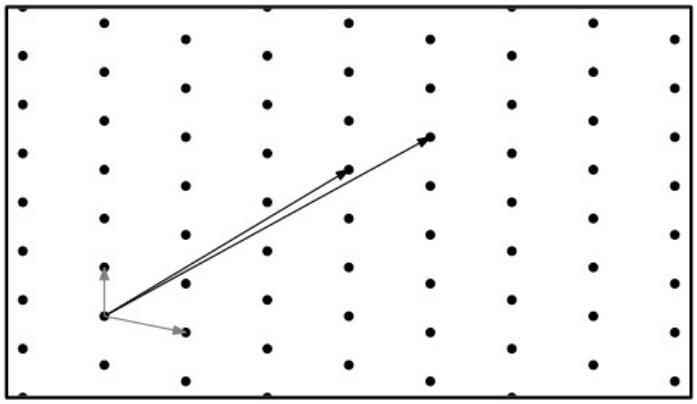


Figure 2.2: A two-dimensional lattice and two possible bases [2]

reconstruct a grid of points that forms a lattice. The number of vectors used in a basis is defined as the rank of lattices [2]. The hardness of Lattice-based cryptography is based on the shortest vector problem (SVP) and the closed vector problem (CVP). In SVP, the main goal is to find the shortest nonzero vector in lattice. The CVP tries to find the closest point to closest to the given nonlattice vector.

2.1.3 Ring Learning With Error

The Ring Learning With Error (Ring-LWE) based crypto-systems operate in a polynomial ring $R_q = Z_q[x]/f(x)$, where one typically chooses $f(x) = x^n + 1$ with n a power of two, and q a prime with $q \equiv 1 \bmod 2n$. The procedures for key generation, encryption, and decryption of this crypto-system are described as follows:

Key generation

This phase generates a public key (a and p) by $p \leftarrow r_1 - a \times r_2$ from a private key (r_2) and a; r_1 and r_2 are two polynomials that are sampled from Gaussian distribution χ_δ. The polynomial a is generated uniformly at random by a trusted source or the user.

Encryption

In this phase, the input message m is encrypted to two cipher-texts (C_1 and C_2). First, the input message should be encoded into a polynomial $\overline{m}$ in which $\overline{m}$ is calculated by $\overline{m} = \lfloor (q/2) \rfloor \, m \in R_q$. Then $C_1 = a \times e_1 + e_2$ and $C_2 = p \times e_1 + e_3 + \overline{m}$ are calculated based on the public key (a, p); e_1, e_2, and e_3 are sampled from the Gaussian distribution.

Decryption

This phase calculates $\overline{m}$ by using $\overline{m} = C_1 \times r_2 + C_2$. The original message m is gained by pre-decoding the polynomial $\overline{m}$ into $\{0, 1\}^n$ by using a decoder. The i-th coefficient of the message m is converted to 1 if and only if its corresponding value $\overline{m}$ satisfies the condition $\overline{m}[i] \in \lfloor q/4, 3q/4 \rfloor$; otherwise, it is converted to 0. As it is obvious, polynomial multiplication is the main module for Ring-LWE.

Number Theoretic Transform (NTT)

Polynomial multiplication is the fundamental module for LBC.Polynomial multiplication is the operation that requires the most processing time and occupied area. NTT is an efficient method of polynomial multiplications among the different ways (School-book, Karatsuba-Ofman). Schoolbook algorithm has a computational complexity of $O(n^2)$; The Karatsuba and NTT algorithms have a computational complexity of $O(n^{log3})$ and $O(nlog(n))$, respectively [22]. The NTT is the FFT defined in a finite field in which the NTT does not use floating-point numbers and complex arithmetic. The NTT can be designed by implementing the butterfly diagram obtained from a radix-r algorithm based on the decimation-in-time (DIT), or decimation-in-frequency (DIF) approaches, where r is a power of two. By using NTT, each polynomial converts into the NTT domain in which polynomial multiplication is transmitted to a coefficient-wise multiplication. The $NTT_\omega(a)$ and $INTT_\omega^{-1}(A)$ are defined as:

$$A_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \; mod \; q$$
$$a_i = n^{-1} \sum_{j=0}^{n-1} A_j \omega^{-ij} \; mod \; q \tag{2.2}$$
$$Where \; i = 0, 1, ..., n-1$$

For most of the post-quantum cryptography algorithms, the arithmetic operations are performed in $R_q = Z_q[x]/f(x)$. To speed up polynomial multiplication, it is necessary to use the negative wrapped convolution for multiplication. Let ω be a primitive n-th root of unity in Z_q in which $\omega^n \equiv 1 \; mod \; q$ and $\psi^2 \equiv \omega \; mod \; q$; $a = (a_0, ..., a_{n-1})$ and $b = (b_0, ..., b_{n-1})$; $c = (c_0, ..., c_{n-1})$ be the negative wrapped convolution of a and b that each element is in Z_q. Also, $a^{'}$, $b^{'}$, and $c^{'}$ are defined as $a^{'} = (\psi^0 a_0, \psi^1 a_1, ..., \psi^{n-1} a_{n-1})$, $b^{'} = (\psi^0 b_0, \psi^1 b_1, ..., \psi^{n-1} b_{n-1})$, and $c^{'} = (\psi^0 c_0, \psi^1 c_1, ..., \psi^{n-1} c_{n-1})$, respectively. Then $c^{'}$ can be calculated by the bellow equation. As each coefficient is multiplied by ψ, the final result should be multiplied by powers of $\psi^{'}$.

$$\mathrm{c}^{'} = NTT_{\omega}^{-1}(NTT_{\omega}(a^{'}) \circ NTT_{\omega}(b^{'})) \tag{2.3}$$

2.1.4 Binary Ring Learning With Error

The Binary Ring-LWE (Ring-BinLWE) was introduced in 2016 by Buchmann [11]. There are two main differences between Ring-BinLWE and Ring-LWE. In Ring-BinLWE, the errors are sampled with binary coefficients, and thus Ring-BinLWE does not require Gaussian distribution and NTT; also, the key size is smaller. These differences make the execution of Ring-BinLWE more efficient on hardware platforms and suitable for IoT resource-constrained devices. An optimized variant of Ring-BinLWE was published in 2019 [1], where the range of coefficients has been changed to $(-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor - 1)$. The main advantage of changing the range is that the modular reduction is performed easily in hardware by overflow and underflow since the new range is matched to 2's-complement.

The key generation, encryption, and decryption of the Ring-BinLWE are introduced as [1]:

Key Generation

In this phase, the public-key is calculated by $p = r_1 - a.r_2 \in R_q$ (p has $n \times k$ bits and $k = log_2^q$), where $a \in R_q$ is a publicly known polynomial, r_2 is the private-key, and r_1 is just used for one time and after calculation can be discarded. Also, $r_1, r_2 \in \{0,1\}^n$ are binary vectors, which are chosen at random, and can be mapped to polynomial in R_q.

Encryption

This phase is used to encrypt the message $m \in \{0,1\}^n$. First, the message should be encoded to a unique polynomial $\overline{m}$ in R_q according to equation (2.4). After that two ciphertexts should be calculated by $C_1 = a.e_1 + e_2$ and $C_2 = p.e_1 + e_3 + \overline{m}$ (C_1 and C_2 are $n \times k$-bit). Also, e_1, e_2, and $e_3 \in \{0,1\}^n$ are errors that are randomly chosen.

$$\begin{aligned} ENCODE : \{0,1\}^n &\rightarrow R_q \\ (m_0, ..., m_{n-1}) &\rightarrow \overline{m} = \sum_{i=0}^{n-1} m_i(-\frac{q}{2})x^i \end{aligned} \tag{2.4}$$

Decryption

By using the private-key (r_2), C_1, and C_2, the message will be decrypted by $\overline{m} = C_1.r_2 + C_2 \in R_q$. To obtain the message, the $\overline{m}$ should be decoded as follow:

$$\begin{aligned} DECODE : R_q &\rightarrow \{0,1\}^n \\ \sum_{i=0}^{n-1} a_i x^i &\rightarrow (m_0, ..., m_{n-1}), \\ m_i &= \begin{cases} 0, & \left|a_i - i - \left\lfloor \frac{n-3}{2} \right\rceil\right| > \frac{q}{4} \\ 1, & else \end{cases} \end{aligned} \tag{2.5}$$

2.2 Related Works

2.2.1 AES Implementation

During the past years, many hardware implementations on ASIC and FPGAs were published in the literature for AES algorithm. Based on the utilized applications, there are different methods for implementing AES. For high-traffic applications that require high-speed, pipeline and loop-unrolling can be employed to achieve high throughput and frequency. It is worth mentioning that most of the high-throughput implementations occupy more area and resources. The other implementation of AES is the lightweight implementation for IoT edge devices that do not have enough resources for the cryptography part. Most of the lightweight designs contain series implementations, which have low throughput and occupy few areas.

[23] has a 32-bit data path with one shared Sub-Bytes. Their design included a big 20-to-1 8-bit MUX and four 32-bit registers for storing the plain-text and intermediate results, and each 32-bit register has four 8-bit outputs and three 32-bit and one 8-bit input, and one 128-bit register for storing keys. The main goal of [20] was to design a low-power AES architecture. They considered two Sub-Bytes and one specific block for Shift-Rows, LUT for storing RCON, and two more registers for storing the intermediate results; also, their Mix-Columns block was 32 bit, which contained eight modules to calculate $2 \cdot a$ and $3 \cdot a$.

The design of [19] transmitted Sub-Bytes and Mix-columns to their native functions. Also, their design included two native Sub-Bytes, which expanded the key simultaneously. This native design, followed by two Sub-Bytes, increased the area by adding more blocks to the design. [24] also considered a specific block for Shift-Rows that contained eight 8-bit registers. They designed two different architectures with one Sub-Bytes and two Sub-Bytes with different values of data-path. Their architecture included more registers for storing keys, data, and intermediate results. There were a big 32×8-bit RAM and one internal register to store the intermediate results in [25]. [21] did not explain more details about

their architecture. However, their architecture had an 8-bit data path that contained two Sub-Bytes, a Mix-Columns block with two 8-bit outputs, a parallel-to-serial converter, and a byte permutation unit.

[26] designed a nano-AES on $22nm$ CMOS technology. They used one Sub-Bytes among Shift-Rows and a Mix-Columns block, which included 8-bit input and 32-bit output that encrypted data in 336 clock cycles. Also, they employed three register banks (each containing sixteen 8-bit registers) to store keys, plain text, and intermediate results. The design of their RCON included an 8-bit shift register followed by four control signals, one 8-bit AND gate, and one 4-bit NOR gate. The design of [27] used two blocks for Sub-Bytes and 32-bit data-path for Mix-Columns that enabled the parallel operation of data encryption along with on-the-fly key generation, which led to encrypting data in 160 clock cycles.

2.2.2 Binary Ring-LWE implementation

As it was mentioned in section 2.1.4, Ring-BinLWE is a variant of Ring-LWE that does not require Gaussian sampler and NTT multiplication. Thus, it occupies less area and resources compared to Rin-LWE. [11] introduced the Ring-BinLWE scheme and implemented on an 8-bit Atmel AVR implementation. They discussed the hardness of Ring-BinLWE based on different parameter sets and evaluated the security of Ring-BinLWE on different platforms. [28] implemented a hardware architecture on FPGA for decryption part of Ring-BinLWE with $n = 256$ $q = 256$. They discussed the vulnerability of Ring-BinLWE against physical attacks. To increase the security against differential power analysis (DPA), they added a power side-channel countermeasure to their design. Their design had an in-place modular reduction and required $n \times n$ cycles to execute multiplication. The architecture of [28] had a maximum frequency of 135MHz, occupied 19 slices on Spartan-6, and $65k$ clock cycles for execution.

In [1], two hardware implementations, high-speed and lightweight architectures, of Ring-BinLWE on ASIC and FPGA were presented for resource-constrained end-node devices, containing three phases: encryption, decryption, and key generation. They selected

the parameter set of $n = 256$ $q = 256$ and $n = 512$ $q = 256$ for their implementations. The authors used a 2's-complement notation range for each coefficient that results in no need for modular reduction. The high-speed architecture of [1] contained the parallel adders of $ax + b$ and needed n clock cycles to complete the multiplication including adders, MUXex, and n 8-bit registers. This design had a frequency of 10MHz and occupied $46000\mu m^2$ on $65nm$ technology. Their lightweight design serially executed the multiplication and included two n-to1 8-bit MUXes and two big registers. The area and maximum frequency of the lightweight architecture of [1] are 33.3MHz and $6000\mu m^2$ on $65nm$ technology, respectively.

In [29], the authors implemented the architecture of [1] on AVR ATxmega128A1. To increase the security against fault attacks, they applied chosen cipher-text attacks (CCA2) [30] to their design. The generated errors and input message were encrypted by AES and sent with the cipher-text in the encryption part. In the decryption part, first, the message and errors were decrypted, and then the message was encrypted again based on decryption errors. Finally, the results were compared to each other. If the result was the same, there was no fault attack on the crypto-system. The execution time were $80.2ms$ and $120.3ms$ for encryption and decryption, respectively.

Part II

AES Implementations

3. High Throughput and area-efficient AES implementation

This chapter presents a high throughput FPGA implementation of AES-128. AES is a well-known symmetric key encryption algorithm with high security against different attacks that is widely used in different applications. The main goal of this chapter is to design a high throughput and FPGA efficiency (FPGA-Eff) crypto-system for high-traffic applications. In order to achieve high throughput, loop-unrolling, inner and outer pipelining techniques are employed. In AES, Sub-Bytes is one of the costly functions that occupies a large amount of resources and has a large delay. In order to reduce the area of Sub-Bytes, New-Affine-transformation, which is the combination of inverse isomorphic and affine transformation, is proposed and employed. Besides that, AES has been modified according to the proposed architecture. For the first nine rounds, Shift-Rows and Sub-Bytes have been exchanged, and Shift-Rows is merged with Add-Round-Key. In order to make an equal latency between stages, Mix-Columns is divided into two different stages. AES is implemented in CTR mode on Xilinx Virtex-5 using VHDL. Section 3.1 presents cryptography modes. AES functions' implementations and hardware implementation of the AES are in sections 4 and 5; the implementation results and comparison are explained in sections 6; and finally the conclusion will be in section 3.4.

The content of this chapter is originally published in: K. Shahbazi and S. -B. Ko, "High throughput and area-efficient FPGA implementation of AES for high-traffic applications," in IET Computers & Digital Techniques, vol. 14, no. 6, pp. 344-352, Nov. 2020, doi: 10.1049/iet-cdt.2019.0179. The manuscript has been reformatted for inclusion in this thesis.

3.1 Introduction

The advent of technologies including the internet and smartphones has made people's lives easier. Nowadays, people get used to digital applications for e-business, communicating with others, and sending or receiving sensitive messages. Sending secure data across the private network or the internet is an open concern for every person. Cryptography plays an important role in privacy, security, and confidentiality against adversaries. Cryptography algorithms are divided into two different categories, symmetric and asymmetric. Asymmetric cryptography algorithms, such as elliptic curve cryptography and Rivest–Shamir–Adleman, use two different keys for encryption and decryption data. In comparison with the asymmetric cryptography algorithms, the symmetric cryptography algorithms using the same keys for encryption and decryption data and have less computation and occupy a fewer area that makes them very hardware friendly. The advanced encryption standard (AES) is one of the most secure, fast, and most used symmetric algorithms that have a good performance in both software and hardware platforms. In comparison with software platforms, hardware implementation of cryptography algorithms, which are application-specific integrated circuit (ASIC) and field programmable gate arrays (FPGAs), provides higher security and throughput. FPGA implementation has some advantages over ASIC implementation, such as the re-configurability and low design cost. AES is used in broad applications and different security networks, such as Wi-Fi protected access 2 system, secure sockets layer, automated teller machines, internet of things, and digital video recorders. In addition to the security of data transmission in many high-traffic applications that transfer a huge amount of data, fast encryption/decryption and transmission are also required, such as high-traffic servers. In this work, a highspeed crypto-system for high-traffic application is presented.

When designing a high-speed crypto-system, there is a relationship between security of the design, speed that is determined by throughput, and cost in terms of area and fabrication, which is shown in Figure 3.1. It is worth to mention that for high-speed crypto-systems, the high priority is security and speed. In terms of security, the cryptographic algorithm and architecture play an important role. Three important factors that affect the

security of an algorithm are key length, number of rounds, and the degree of confusion and diffusion. More rounds and longer key lengths provide a safer algorithm. Confusion and diffusion make a complex relation between plaintext, keys, and cipher. By employing the diffusion and confusion, changing one bit of the plaintext will completely change the cipher-text, which the attacker cannot pre-calculate what the plaintext is. In AES, an excellent confusion and diffusion, Mix-Columns and substitution bytes (Sub-Bytes) are employed and repeated in several rounds that make AES secure against different attacks. The speed of the crypto-system is mainly determined by throughput. For a pipeline design, throughput is measured by frequency and number of proceed bits in every clock cycle, and for a serial architecture, the total number of clock cycles has a high impact on throughput. A pipeline architecture, compared to serial one, with a loop-unrolling technique, boosts speed. Finally, the cost of a crypto-system is directly related to the security and the speed of the design. For instance, AES-256 provides more security than AES-128 that needs more rounds for encryption plaintext that leads to an increase in the cost of the design. Also, the serial designs occupy fewer gates than parallel and pipeline ones. Needless to say, stronger and faster crypto-systems would require more costs.

Cryptography algorithms are implemented by different types of modes. There are two different cryptography modes, in which each cryptography algorithm is executed in different modes. In general, the cryptography modes are divided into two categories: independent and dependent. The main difference between them is a dependency of the previous cipher-text; for encrypting the block 'n' in the dependent mode, the cipher-text block '$n-1$' should be available. On the other hand, there is no relation for encrypting blocks in the independent mode, which includes counter (CTR) mode and electronic codebook (ECB). Thus, this mode is suitable for a high throughput pipeline design. Between these two independence modes, CTR and ECB, which are used for high-speed design, CTR is more secure than ECB and is recommended for employing in various applications.

There are various methods and ways to implement AES. In general, the works on the AES algorithm contain fast and high throughput implementation, optimisation of the

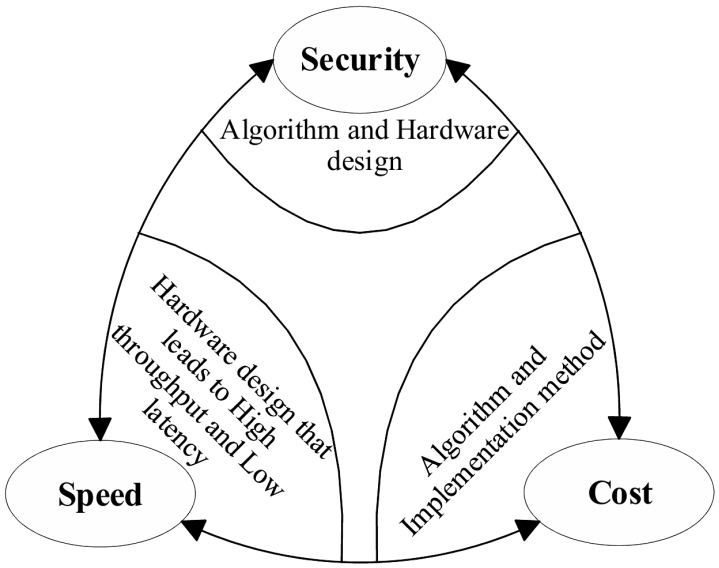


Figure 3.1: An illustration of the relationship between priorities when designing a high-speed crypto-system and the design factors that affect them

hardware design, low area, and low-power implementation. The main function of the AES algorithm in both area occupation and latency is the Sub-Bytes function that uses an 8-bit substitution box (S-box). There are different ways to implement the Sub-Bytes function. One of the straightforward ways for implementing Sub-Bytes is to use look-up table (LUT) [31] [32] [14]. Since LUT is an easy way, it has a high delay and requires an enormous amount of memory that is not suitable for high throughput design. The other way is to simplify the S-box table by using techniques such as sum of products (SoPs). In [16], the Sub-Bytes has 16 modules that are derived by using Boolean simplification based on the Karnaugh map with a 16 to 1 multiplexer. The first 4-bit input is the input of the modules, and another 4-bit is the selection input of the multiplexer. Since 16 to 1 multiplexer is big, the Sub-Bytes suffers a high and unbreakable delay. In [15], instead of using a 16 to 1 multiplexer, five 4 to 1 multiplexers were used. Another way for calculating Sub-Bytes is to use composite field arithmetic [33] [34] [35]. Besides that, there are various architecture designs for AES. AESs based on co-processor and

specific processor are implemented [13] [36] [37] [38]. The authors of [39] described pipelined implementations of AES and International Data Encryption Algorithm (IDEA), the other symmetric cryptography algorithm, with CTR mode and LUT for Sub-Bytes on FPGA. In [15], different pipelined AES implementations in ECB and CTR mode by using combinational logic circuits and the simplified of the S-box were presented. Some other pipelined designs are available in [40] [41] [42].

The main goal of this chapter is to design and implement a high throughput AES-128 algorithm for high-traffic applications. The following methods and techniques are considered to achieve the aforementioned goal:

- The AES algorithm is modified in which Sub-Bytes and ShiftRows are exchanged for the first nine rounds; Add-Round-Key and Shift-Rows are merged into one stage.
- The Sub-Bytes is optimised in which inverse isomorphic and affine transformation (Af) are combined together, and new affine-transformation, hereafter referred to as New-Aff, is proposed.
- The critical path of the composite field arithmetic for multiplicative inverters in $GF(2^8)$ is considered; and the delay of Sub-Bytes is reduced by inserting registers in optimal places.
- Mix-Columns are implemented based on logic gates and into two different stages. To make an equal delay between these two stages, the first stage is added to Sub-Bytes.
- Full loop unrolling, inner (inserting registers between rounds) and outer (inserting registers among functions) pipeline stages are used.

3.2 Cryptography Mode

To process the data and encrypt them, there are different cryptography modes. In general, the cryptography modes can be divided into two different categories: independent and dependent modes. The dependent mode contains cipher block chaining (CBC), propagating

CBC (PCBC), cipher feedback (CFB), and output feedback (OFB). In the dependent cryptography modes, the current block depends on the previous cipher-text. In some cases, the previous cipher-text is used as the input of the current block or one simple operation, most of the time XOR should be executed with the current plaintext and the previous cipher-text. Figure 3.2 shows the dependent cryptography modes. For increasing the security, CFB and OFB use an initial vector as input.

On the other hand, the independent cryptography modes contain CTR mode and ECB. Figure 3.3 shows the CTR and ECB configuration. These two cryptography modes are very suitable for pipelines and high throughput design [43]. ECB is the easiest way to use because every plaintext is encrypted without any need for previous data. However, ECB does not have enough security in comparison with other cryptography modes and is not suitable for sequential data blocks. The CTR mode, instead of using a constant initial vector, uses a sequence vector as input for each block that is increased by one for every block. Using a different initial value for each plaintext increases the security of the design. One advantage of the CTR mode is that the encryption and decryption can be performed by the same structure. The difference between using CTR and ECB modes for image encryption is shown in Figure 3.4. As is observed in Figure 3.4, the edge of the encrypted image in ECB can be easily detected; and by doing that the main features of the plain-image are distinguishable. In comparison with ECB, the encrypted image in CTR is uniform and indistinguishable.

3.3 AES Functions Implementation

This section describes efficient techniques for implementation of the Mix-Columns and Sub-Bytes, and express the employed method in the proposed design.

3.3.1 Mix-Columns

There are different ways and methods for calculating the Mix-Columns. As the Mix-Columns has fixed values, one way is to pre-calculate values and store them as a LUT. This

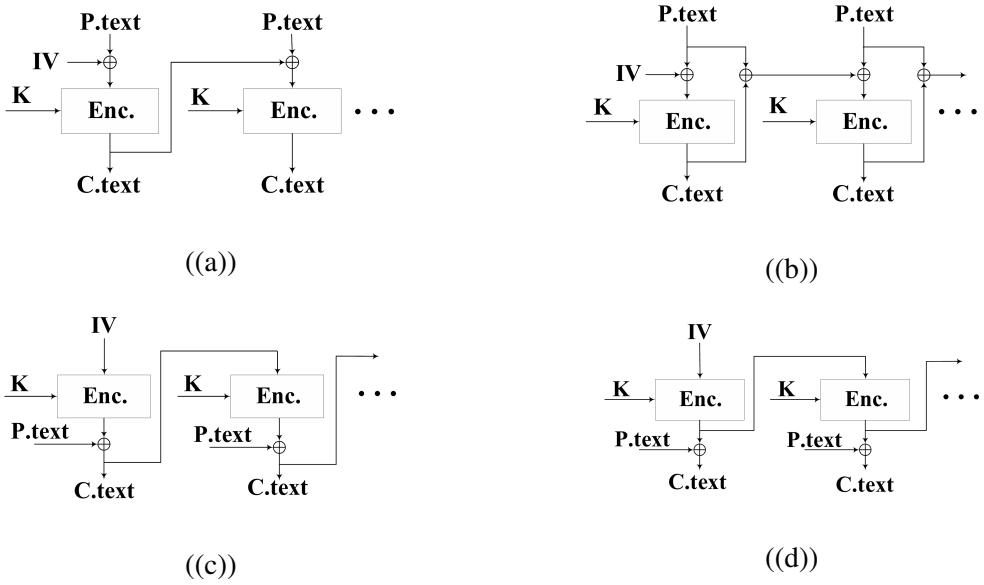


Figure 3.2: Different dependent cryptography modes (a) CBC, (b) PCBC, (c) CFB, (d) OFB (IV stands for initialisation vector)

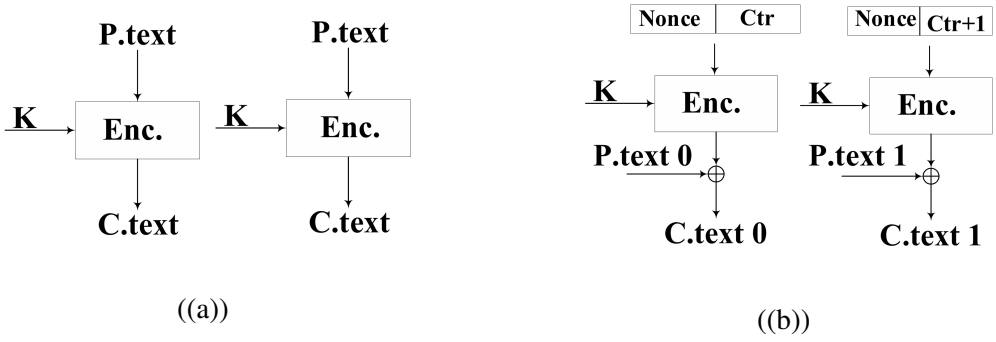
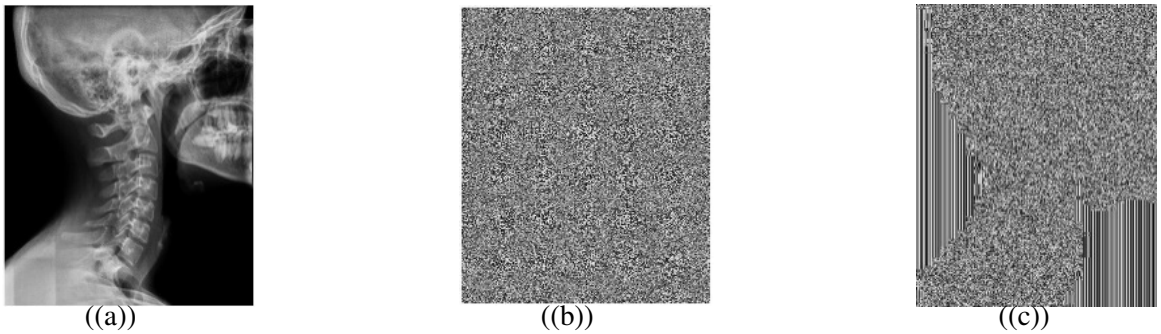


Figure 3.3: Independent cryptography modes (a) ECB, (b) CTR



((a)) ((b)) ((c))

Figure 3.4: The difference between encryption of an image using CTR and ECB modes (a) Main image, (b) Encrypted image in CTR, (c) Encrypted image in ECB

method, however, is not suitable because of high area consumption and high latency and is not efficient for area limited applications. As the Mix-Columns calculation is not very complicated, it is possible to calculate the values by incoming data according to equation (3.1).

$$\begin{pmatrix} a' \\ b' \\ c' \\ d' \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \rightarrow \begin{cases} a' = 2.a + 3.b + c + d \\ b' = a + 2.b + 3.c + d \\ c' = a + b + 2.c + 3.d \\ d' = 3.a + b + c + 2.d \end{cases}$$
$$\rightarrow \begin{cases} a' = 2.a + (2.b + b) + c + d \\ b' = a + 2.b + (2.c + c) + d \\ c' = a + b + 2.c + (2.d + d) \\ d' = (2.a + a) + b + c + 2.d \end{cases} \tag{3.1}$$

In equation (3.1), '.' is a multiplication modulo, the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$), and '+' is XOR. By simplifying $3a$ to $2.a + a$, the goal is to calculate the multiplication with 2. Here there are two different ways to calculate the $2.a$. The multiplication by 2 can be written by:

$$
\begin{aligned}
&(2.a) mod(x^8 + x^4 + x^3 + x + 1) \overset{a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0}{\longrightarrow} \\
&\left\{(a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0) \times (x^1)\right\} \\
&mod(x^8 + x^4 + x^3 + x + 1) \rightarrow \\
&\left\{a_7 x^8 + a_6 x^7 + a_5 x^6 + a_4 x^5 + a_3 x^4 + a_2 x^3 + a_1 x^2 + a_0 x^1\right\} \\
&mod(x^8 + x^4 + x^3 + x + 1)
\end{aligned} \tag{3.2}
$$

Now, the reduction over the module $m(x)$ should be calculated. As the element of the x^8 is a_7, $m(x)$ is multiplied by a_7 so:

$$
\begin{aligned}
&(a_7 x^8 + a_6 x^7 + a_5 x^6 + a_4 x^5 + a_3 x^4 + a_2 x^3 + a_1 x^2 + a_0 x^1) \\
&+ a_7(x^8 + x^4 + x^3 + x + 1) \rightarrow \\
&(a_7 + a_7)x^8 + a_6 x^7 + a_5 x^6 + a_4 x^5 + (a_3 + a_7)x^4 + (a_2 + a_7)x^3 \\
&+ a_1 x^2 + (a_0 + a_7)x^1 + a_7 \rightarrow \\
&a_6 x^7 + a_5 x^6 + a_4 x^5 + (a_3 + a_7)x^4 + (a_2 + a_7)x^3 + a_1 x^2 + (a_0 + a_7)x^1 + a_7
\end{aligned} \tag{3.3}
$$

Thus, $2.a$ can be calculated by:

$$
x = (2.a) mod(x^8 + x^4 + x^3 + x + 1) \rightarrow \begin{cases} x_7 = a_6 \\ x_6 = a_5 \\ x_5 = a_4 \\ x_4 = a_7 \oplus a_3 \\ x_3 = a_7 \oplus a_2 \\ x_2 = a_1 \\ x_1 = a_7 \oplus a_0 \\ x_0 = a_7 \end{cases} \tag{3.4}
$$

As $2.a$ is a shift bit to left, the other way to calculate $2.a$ using one-bit shift to left, one Mux 2×1, and a NOT logic gate. In equation (3.4), when $a_7 = 0$, $2.a$ is equal to the one-bit

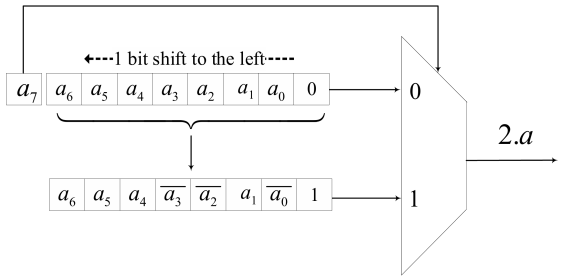


Figure 3.5: The proposed Mix-Columns-1 with Shift, NOT, and Mix 2×1

shift to the left of a, and when $a_7 = 1$, $2.a$ is calculated by one-bit shift to the left of a XOR by $(x^4 + x^3 + x + 1)$, which is equal to '$1b$' in hex. As '$1b$' is fixed, the result can be transferred to NOT gate. The diagram of calculating $2.a$ by Mux, Shift, and NOT is shown in Figure 3.5. In the proposed design, the Mix-Columns is executed in two different stages: in one stage, which is called 'Mix-Columns-1', $2.a$ is calculated; in another stage, which is called 'Mix-Columns-2', the final values of equation (3.1) is computed.

3.3.2 Sub-Bytes

Sub-Bytes is the most costly transformation in AES, in both time and area aspects. The first step for finding the values of this function is to calculate the multiplicative inverse of inputs over $GF(2^8)$. The multiplicative inverse of $f(x)$ is $g(x)$ that $f(x).g(x)mod(x^8 + x^4 + x^3 + x + 1) = 1$. The second step is computation over an *Af*. The important effect of applying an invertible *Af* before and after the Sub-Bytes is that the resistance of S-boxes against most attacks remains unchanged [44]. Equation (3.5) is the *Af*, which $g(x) = (a'_7a'_6a'_5a'_4a'_3a'_2a'_1a'_0)$ is the multiplicative inverse of $f(x)$, and Y is the Sub-Bytes of the $f(x)$.

$$Y = Af(g(x)) + \Phi = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} a'_7 \\ a'_6 \\ a'_5 \\ a'_4 \\ a'_3 \\ a'_2 \\ a'_1 \\ a'_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{3.5}$$

There are different hardware implementations of Sub-Bytes function. Direct calculation of the multiplicative inverse is not easy. As all of the values are available, one of the straightforward implementations of the S-box is pre-calculating the values and storing them in a LUT read only memory. The most significant part of the input determines the row of the table and the least significant part determines the column of the table. The read and write from the LUT to registers need high latency and more areas. Thus, LUT is not an efficient method for high-speed application and pipeline design.

The other way is the Boolean simplification map for the pre-calculation of Sub-Bytes using truth table based on the Karnaugh map to make a direct relation between the parameter of the Sub-Bytes function. The simplification of the whole S-box table is not an easy way. The authors of [16] divided the S-box table into 16 tables and for each sub-table the SoP is calculated as a module logic function. The 16 to 1 multiplexer was used to achieve the output. The most significant 4-bit are the inputs of the modules and the least significant 4-bit are used as the selector of the multiplexer. Although this method reduces the latency of the S-box by LUT, it needs a large area too.

Instead of working with the S-box, the multiplicative inverse is calculated by composite field arithmetic and followed by the *Af*. The composite field arithmetic is based on $GF(2^1)$, $GF(2^2)$, and $GF((2^2)^2)$. As a result, the element that is on $GF(2^8)$ should be mapped to its composite field representation. The isomorphic function, 'δ', and its inverse, 'δ^{-1}' are

used to transfer between $GF(2^8)$ and its composite field and vice versa (equations (3.6) and (3.7)). After mapping to the composite field, the elements in $GF(2^8)$ are decomposed to the lower order fields of $GF((2^2)^2)$, $GF(2^2)$, and $GF(2^1)$ by the irreducible polynomials equation (3.8).

$$\delta \times x = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \tag{3.6}$$

$$\delta^{-1} \times x = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \tag{3.7}$$

$$\begin{cases} GF(2^2) \rightarrow GF(2) \\ GF((2^2)^2) \rightarrow GF(2^2) \\ GF(((2^2)^2)^2) \rightarrow GF((2)^2) \end{cases} \rightarrow \begin{cases} P_0(x) : x^2 + x + 1 \\ P_1(x) : x^2 + x + \phi, \quad where \qquad \phi = \{10\}_2 \\ P_2(x) : x^2 + x + \lambda, \quad where \qquad \lambda = \{1100\}_2 \end{cases} \tag{3.8}$$

One of the advantages of using the composite field arithmetic is to reduce the area of Sub-Bytes as well as it is an efficient method for pipeline design, while the hardware

complexity is its disadvantage. As it is mentioned, the Sub-Bytes consists of two steps: in the first step, the multiplicative inverse in $GF(2^8)$ is calculated; the second step is to apply the *Af*. Let 'Mul' and 'S' define as multiplicative inverse and Sub-Bytes. Thus, Sub-Bytes can be written as follows:

$$\begin{aligned} S(f(x)) &= Af(g(x)) + \Phi \\ S(f(x)) &= Af(\delta^{-1}(Mul(\delta.f(x)))) + \Phi \\ S(f(x)) &= Af.\delta^{-1}(Mul(\delta.f(x))) + \Phi \\ S(f(x)) &= \gamma(Mul(\delta.f(x))) + \Phi \end{aligned} \tag{3.9}$$

The 'γ' is calculated by: '$\gamma = Af.\delta^{-1}$'; thus, instead of using two different matrices in two different stages, one matrix is used in the design. γ is called New-Aff. New-Aff is written as:

$$\gamma = Af.\delta^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \tag{3.10}$$

Figure 3.6 shows the employed multiplicative inverse $GF(2^8)$ in the proposed architecture. In Figure 3.6, 'δ' is the isomorphic mapping to composite fields, 'x^2' is the squarer in $GF(2^4)$, 'λ' is the multiplication with constant 'λ' in the $GF(2^4)$, 'x^{-1}' is the multiplicative inversion in $GF(2^4)$, and 'X' is the multiplication operation in $GF(2^4)$.

In Figure 3.6, λx^2 is squarer in $GF(2^4)$ followed by multiplication with constant λ in $GF(2^4)$. Every binary number in the Galois field 'q' can be represented by $bx + c$, where

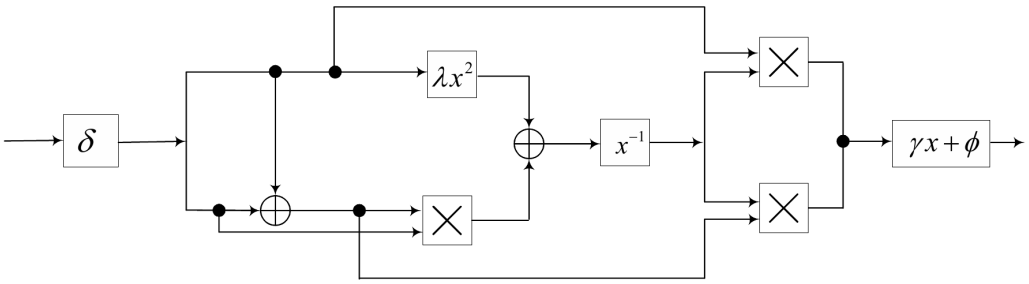


Figure 3.6: Implementation of Sub-Bytes using composite field with the combination of Af and δ^{-1}

'b' is the upper half-term ('q_H') and 'c' is the lower half-term ('q_L'). By considering this idea and equation (3.8), parameters in Figure 3.6 can be derived. For example, to compute the squaring in $GF(2^4)$, $A = B^2$, which 'A' and 'B' are elements in $GF(2^4)$, is equal to the $(A_H x + A_L) = (B_H x + B_L)^2$, and by computation equation (3.11) is achieved.

$$\begin{aligned} A_3 &= B_3 \\ A_2 &= B_3 \oplus B_2 \\ A_1 &= B_2 \oplus B_1 \\ A_0 &= B_3 \oplus B_1 \oplus B_0 \end{aligned} \tag{3.11}$$

Multiplication with constant λ can be performed by the same way as squarer. By using the achieved equations for multiplication and squarer, the parameter λx^2 is simplified and calculated by

$$y = \lambda x^2 \rightarrow \begin{cases} y_3 = x_0 + x_1 + x_2 \\ y_2 = x_0 + x_3 \\ y_1 = x_3 \\ y_0 = x_2 + x_3 \end{cases} \tag{3.12}$$

Similar to the multiplication inverse in $GF(2^8)$, there are different ways of implementing

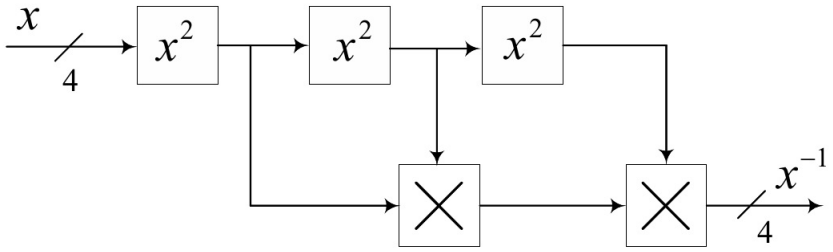


Figure 3.7: Implementations of inversion in $GF(2^4)$ by square and multiplication approach

Table 3.1: Multiplicative inverse over $GF(2^4)$

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x^{-1}	0	1	3	2	f	c	9	b	a	6	8	7	5	e	d	4

the multiplication inverse for $GF(2^4)$. In a Galois field, each non-zero element 'x' in $GF(q^m)$ can be represented by $x = g^n$, where 'g' is a primitive element and 'n' is a unique number with $0 \leq n \leq q^{m-2}$. Furthermore, $g^{q^{m-1}}$ is equal to $\{01\}$. For the two elements $a = g^\alpha$ and $b = g^\beta$, $a.b$ is equal to $g^{\alpha+\beta}$. The multiplicative inversion in $GF(2^4)$ for 'x' is x^{-1} that $x.x^{-1} = x^{15} \longrightarrow x^{-1} = x^{14}$. Also, x^{14} can be written by $((x^2)^2)^2.(x^2)^2.x^2$. Thus, the inversion in $GF(2^4)$ is achieved by using the square of 'x' and multiplication. Figure 3.7 shows the circuit for multiplicative inverse in $GF(2^4)$.

The other way to compute x^{-1} over $GF(2^4)$ is by decomposing the $GF(2^4)$ to the lower order fields, such as $GF(2^8)$, of $GF(2^2)$ and $GF(2^1)$ by the irreducible polynomials. The third way is to pre-calculate the elements and store them. In comparison with $GF(2^8)$, $GF(2^4)$ has only 16 elements and does not occupy a lot of resources, which is used for the design. The pre-calculation results of $GF(2^4)$ are shown in Table 3.1.

3.4 Proposed Architecture and Security Analysis

In the previous section, the methods of the hardware implementation of the Sub-Bytes and Mix-Columns are explained. In this part, the modified AES algorithm, the employed ar-

chitecture, and security analysis are discussed. For the proposed architecture, the CTR mode is employed, which increases security and is the best option for the pipeline implementation of sequential data blocks.

3.4.1 Modified AES algorithm

According to Figure 2.1, the AES algorithm has: Add-Round-key, followed by iteratively Sub-Bytes, Shift-Rows, Mix-Columns, and Add-Round-Key in sequence and execute for nine rounds, and finally, Sub-Bytes, Shift-Rows, and Add-Round-key are executed (Figure 3.8a). Since Shift-Rows and Sub-Bytes are executed on each byte of the input, these two functions can be exchanged without any effect on final results (Figure 3.8b). By moving Shift-Rows to the previous round, the new design of the AES algorithm is obtained. In the modified version, which is shown in Figure 3.8c, Add-RoundKey and Shift-Rows are merged to one block, which is called ARK/Shift, for the first nine rounds; and the last round does not have Shift-Rows function. The advantages of performing this modification are as follows:

- Considering the Shift-Rows and Add-Round-Keys in a separate pipeline stage occupies more area, by merging these two functions into one block, the number of pipeline stages and occupied area have been reduced.
- In comparison with Mix-Columns and Sub-Bytes, Add-roundKey and Shift-Rows are low latency functions of the algorithm, and also merging these two functions does not have any effect on the critical path of the design.
- One part of Mix-Columns, Mix-Columns-1, can be transferred to Sub-Bytes. By doing that the latency of the Mix-Columns is reduced.

3.4.2 Proposed hardware structure

The main contribution of this design is achieving high throughput and efficient hardware implementation. To achieve this goal, loop unrolling, inner pipelining, and outer pipelining

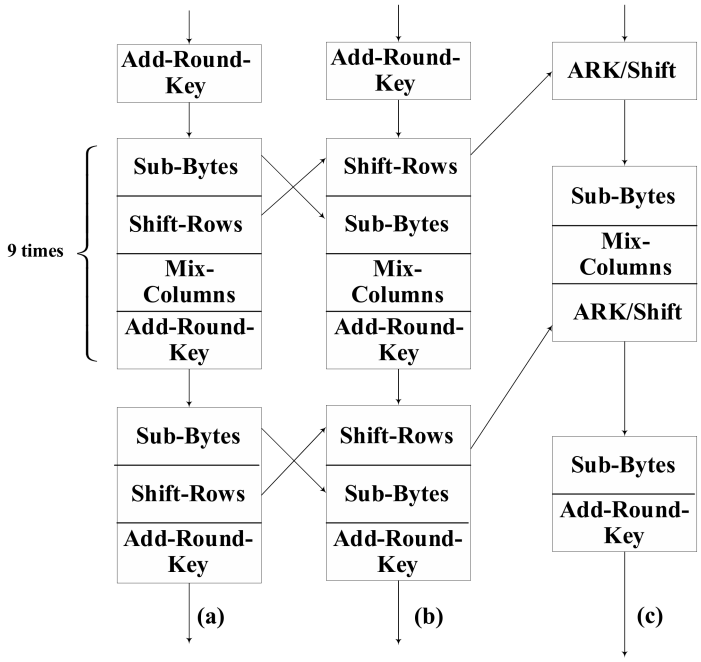


Figure 3.8: The process of modifying the AES algorithm based on the proposed design

are employed. The main idea of loop unrolling is to iterate the design in a number of times. For the loop unrolling, the loop unrolling factor, 'r', is the number of loops that are unrolled to implement the design, and 'T' is the total number of iterations (i.e. for AES-128, the number of rounds is 10). Thus, the number of rounds that need to be iterated to encrypt one block of data is '$R = \frac{T}{r}$'. If '$R = 1$', the design is a full loop unrolling; if '$R > 1$', it is a partial loop unrolling [45].

The proposed architecture is a full loop unrolling design ('$r = T = 10$'). Two advantages of using loop unrolling technique are: first, it mitigates and modifies the critical path by removing all rounds and loops in the design; second, the implementation can easily be extended to outer-round pipelining. The loop unrolling technique converts the design to the pipeline one by inserting registers between the unrolled rounds. The pipeline design is a high performance hardware implementation and a wise way to implement high throughput and fast architecture. In comparison with the non-pipeline design, such as serial, that the plaintext should feed to the design after getting the previous plaintext, the pipeline design

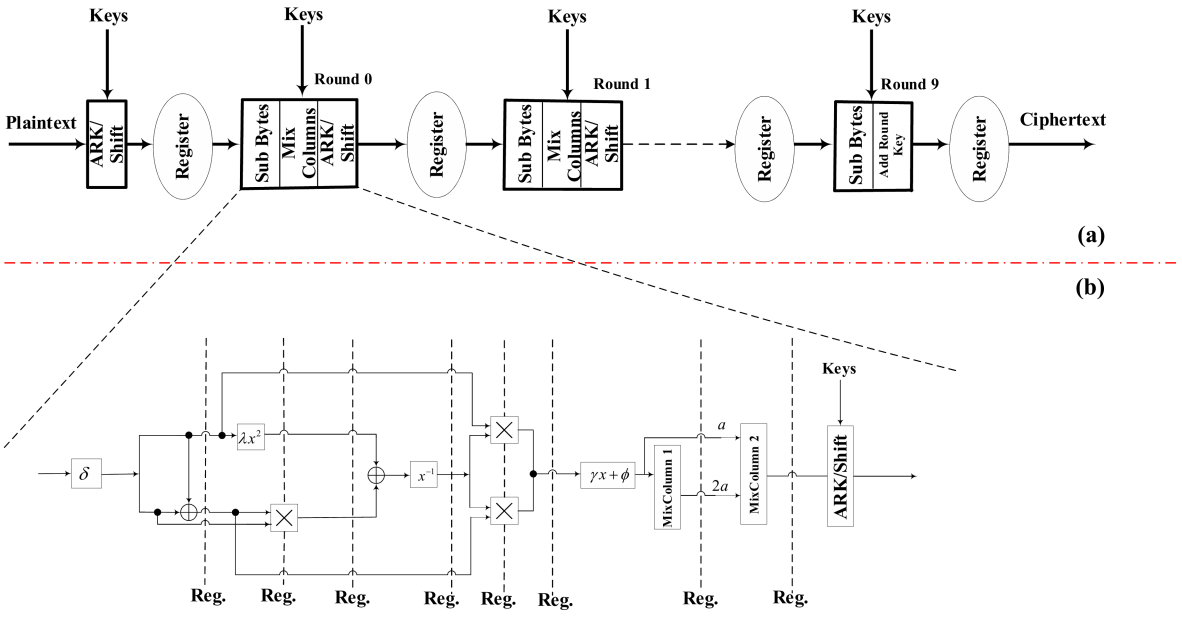


Figure 3.9: The proposed block diagram of AES with pipeline and loop unrolled technique (a) Loop unrolled and outer pipelined stages, (b) Inner pipelined stages

feeds by plaintext in each clock cycle. Thus, the rate of encryption will be increased. Besides that, the pipeline design increases the security against classical cryptanalysis such as differential power analysis [46]. The block diagram of AES with loop unrolled, outer pipelined, and inner pipelined is shown in Figure 3.9.

Besides the outer pipeline, the inner pipeline is used for each round of the algorithm, which decreases the latency of the design. The diagram for one round of the inner pipelined AES implementation is shown in Figure 3.9(b). The inner pipeline is used in inside Sub-Bytes and Mix-Columns. As is mentioned, Mix-Columns and Sub-Bytes have high latency for the design. To achieve the low latency and high frequency of the design, these two functions are divided into some sub-stages with approximately equal delay.

Ever since using the composite field for Sub-Bytes, the Sub-Bytes function has been a high critical path; thus, this unit may need to be divided into more sub-pipeline stages to achieve low latency. The critical path for Sub-Bytes is multiplication operation in $GF(2^4)$.

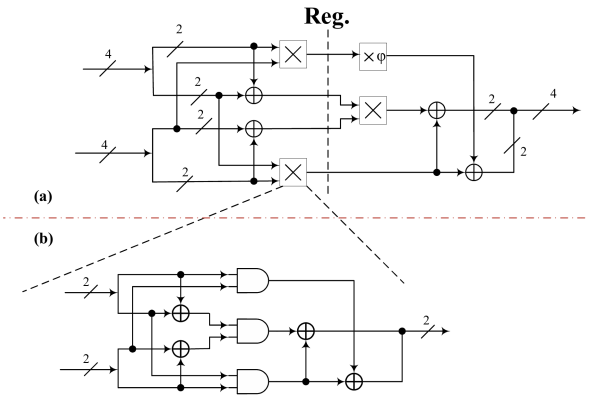


Figure 3.10: Pipeline stage through the multiplication operation (a) Multiplication operation in $GF(2^4)$, (b) Multiplication operation in $GF(2^2)$

The multiplication operation in $GF(2^4)$ is shown in Figure 3.10(a) ('$\times$' in Figure 3.10(a) is the multiplication operation in $GF(2^2)$). The multiplication operation in $GF(2^2)$ is shown in Figure 3.10(b).

Execution of the Mix-Columns in one clock has a high latency. As is explained in the previous section, the Mix-Columns contains two parts, in one part 2.a is calculated and then the inputs are XORed to each other according to the Mix-Columns matrix. To reduce the delay, Mix-Columns, which is shown in Figure 3.11, is executed into two different stages: in one stage 2.a of each byte of the state is calculated by equation (3.4), in the other stage, the output is computed according to equation (3.1). Mix-Columns-1 is transferred to the last stage of Sub-Bytes.

In AES-128, the keys are 128-bit (such as input state, keys have 16 bytes for each round), which is expanded for the next rounds. AES-128 is executed ten times that need $4 \times 10 = 44$ columns of keys. In the proposed design, keys are expanded before running the algorithm and are stored into registers.

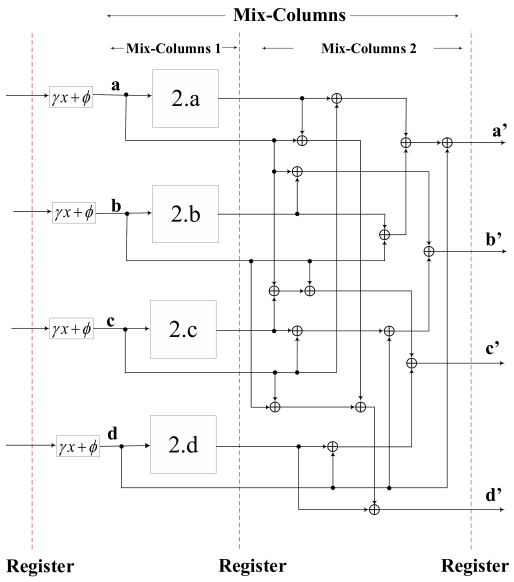


Figure 3.11: Execution of Mix-Columns in two stages

3.4.3 Security analysis of the proposed crypto-system

The security of a crypto-system contains two parts: the algorithm and the implementation. In the proposed architecture, AES is implemented without any alteration on the algorithm's functions. Since the content of AES has not been changed, it is expected the design keeps the same level of confidentiality and security.

In the implementation side, there are some attacks and threats that depend on the implementation design. A side-channel attack is implementation-dependent. The target of side-channel attacks is exploiting the relationship between physical attributes, such as power consumption and timing behaviour, and the manipulated data. Loop unrolling is a recommended technique for increasing the security against side-channel attacks [47]. Loop unrolling, inner-round pipelining, and outer-round pipelining are used for the design, which guarantees reasonable protection from side-channel attacks. Also, the proposed architecture is secure against timing attacks because (*i*) there is no conditional branches and dependency between the inputs and cipher-text; (*ii*) for encrypting each plaintext, the proposed architecture executes in a constant number of clock cycles; and (*iii*) the critical

path delay of the proposed architecture is constant during the execution [48].

3.5 Implementation results, simulation, and comparison

The results of the implementation are reported in this section. The target FPGA is Xilinix Vertix-5 (XC5VLX85-FF676-3) by using ISE 14.7 and ISim 14.7 for synthesis, simulation, and post-placement timing results with VHDL. In the pipelined AES implementations, plaintext blocks are accepted in each clock cycle. As the design is pipelined, the throughput is calculated by the number of processed bits per cycle, which is 128-bit for the design, multiplied by the frequency. Thanks to the pipelined implementation, reducing the latency of the design by inserting the pipeline stage in optimised placement, and exchanging and merging some part of the AES, the achieved clock cycle is $1.607ns$ and the frequency is 622.4 MHz. The throughput of the design is 79.7 Gbps. Moreover, the FPGA efficiency (FPGA-Eff) is a parameter to consider the hardware resources of a design and it is calculated by the design's throughput divided by the number of slices (in Mb/s per slice). However, different articles calculated FPGA-Eff by a different number of slices that contains number of slices LUT and number of occupied slices (the detail of implementation results are written in Table 3.2), in which the total number in Vertix-5 (XC5VLX85) is 51840 and 12960, respectively. Table 3.2 shows the result of the proposed work and some other comparable FPGA implementations. The first five implementations, including the proposed work, are on the same platform.

The experiments performed in [15] [39] [49] are in CTR mode. In [15], four 128-bit AES cores are executed in parallel; as the authors of [15] have mentioned that the throughput of a single core is a quarter of their four designs, the throughput for one core is equal to 44.7 Gbps. The authors of [15] did not mention which number of slices they had used for calculation FPGA-Eff. Since the reported slice number in [15] is closed to the total number of Slices-LUT in FPGA, it is assumed that the reported number would be the number of Slices-LUT. The encryption part of the AES algorithm was designed by Farashahi *et al.* [51] that included a feedback loop and iterative architecture, which occupied fewer areas and

Table 3.2: Implementation results and comparison

References	Device	Frequency, (MHz)	Register	LUT	Occupied Slices	BRAM	Throughput (Gbps)	FPGA-Eff (Mbps/slice)	
								LUT	Occupied
This work	Virtex-5 XC5VLX85	622.4	19,123	14966	5974	0	79.7	5.3	13.3
[15][1]	Virtex-5 XC5VLX85	348.8	30806			0	178.62	5.8	
[49]	Virtex-5 XC5VLX85	576.0	-	22994	-	0	73.7	3.2	-
[50]	Virtex-5 xc5vfx70t	460.0	-	-	9756	0	60.0	-	6.1[2]
[51]	Virtex-5 XC5VLX85	52804	-	3557	-	-	67.6	19.0	-
[39]	XC2V6000-6	194.7	-	-	3720	28	24.9	-	6.7
[52]	Virtex-5 XC5VFX70T	91.6	-	2030	-	28	0.9	0.5	-
[53]	Virtex-5 XC5VLX85	425.0	922	564	303	10	1.3	2.3[2]	4.4
[54]	Virtex-5 XC5VLX85	242.2	-	5256	1745	0	3.1	0.6[2]	1.8[2]
[55]	Virtex-5 XC5VLX85	339.1	-	1338	399	0	4.3	3.2[2]	10.8[2]
[56]	Virtex-5 XC5VLX110T	250.0	-	-	-	0	31.2	-	-

[1] It has four cores; to make a fair comparison and according to [15], the throughput should be divided by four. The throughput for a single core is $\frac{178.62}{4} = 44.7$Gbps.

[2] Manually calculated

required more clock cycles. To make a fair comparison with [51], AT=$area \times time$ has been calculated, which is a common and wide measurement in literature, including [1]. As the proposed crypto-system has been tested by 688×576 pixels images, the encryption time in [51] has been manually calculated, and it is equal to $507\mu s$ (for the proposed design it is equal to $40\mu s$). Then, AT is 1.8 and $0.6s\times$ $slices$ for [51] and the proposed design, respectively, which has 66% improvement through encryption over the same task. As a result, although the design in [51] occupied a few LUTs, it is not suitable for encrypting sequential and dependent large data, such as images. Although the method proposed in [39] was implemented in different FPGAs, it is pipelined with CTR mode, which is very similar to the proposed design. As is observed from Table 3.2, the proposed design has the highest throughput, 79.7 Gbps. The throughput of the proposed design is improved by 8.02%, compared with the best previous work [49], for implementing on the mentioned FPGA and same cryptography mode. By considering the number of occupied slices, the proposed design's FPGA-Eff is 13.3 Mbps/slice by 22.63% improvement over [55].

The proposed crypto-system can encrypt and decrypt different plaintexts and plaint-images. Owing to two reasons, the proposed design is tested on some medical images: first, images have sequential and dependent data blocks; second, the transmission of medical images over the network has been increased rapidly. To show the independency of inputs, however, coloured images with different sizes have been tested by the proposed crypto-system. Block random access memories (BRAM) are used to store image data when testing encryption or decryption using the proposed design. Afterwards, the data is sent to the crypto-system, and the en/decrypted results are then sent back to the host PC. Table 3.2 lists the results of the proposed AES crypto-system and offers a direct comparison to related works.

Histogram of an image is a statistical feature that represents the total distribution of pixel values in a digital image. A secure and effective crypto-system should produce a uniform cipher histogram distribution, which guarantees the resistance against statistical attacks. Figure 3.12 depicts the main tested images, the encryption (in noiseless status) of

images, and the histogram of the images and their corresponding ciphers.

The histogram of a coloured image is produced by taking the histograms over each of its three channels (red, green, and blue). As can be seen in Figure 3.12, the histograms of the encrypted images by the proposed crypto-system are uniform for every encrypted image and different from the input images. The proposed crypto-system encrypts a grey-scale image with 688×576 pixels in $40\mu s$. Every coloured image has three channels (red, green, and blue); and to encrypt (or decrypt) a coloured image, every channel should be encrypted (or decrypted) independently and finally merged together. A coloured image with $688 \times 576 \times 3$ pixels is encrypted in $120\mu s$.

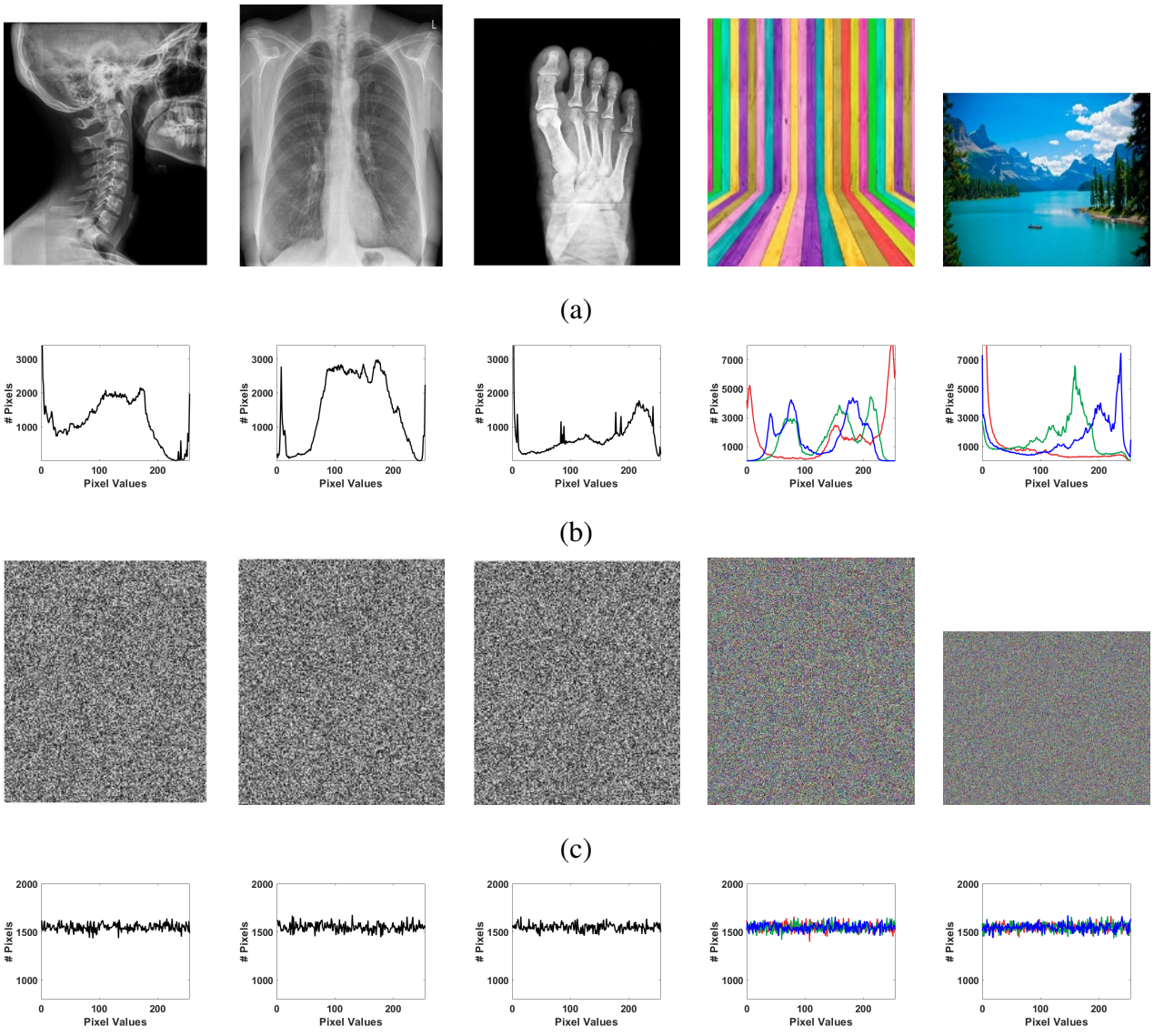


Figure 3.12: Encryption, decryption, and histogram of coloured images with different size by using the proposed crypto-system (a) Main images, (b) Histograms of main images, (c) Encrypted images, (d) Histograms of encrypted images

4. Area-Efficient Nano-AES Implementations

Due to the fast-growing number of connected tiny devices to the Internet of Things (IoT), providing end-to-end security is vital. Therefore, it is essential to design the crypto-system based on the requirement of resource-constrained IoT devices. This chapter presents a lightweight AES, a high-secure symmetric cryptography algorithm, implementation on FPGA and 65*nm* technology for resource-constrained IoT devices. The proposed architecture includes 8-bit data-path and five main blocks. We design two specified register banks, Key-Register and State-Register, for storing the plain-text, keys, and intermediate data. To reduce the area, Shift-Rows is embedded inside the State-Register. To adapt the Mix-Column to 8-bit data-path, an optimized 8-bit block for Mix-Columns with four internal registers is designed, which accept 8-bit and send back 8-bit. Also, a shared optimized Sub-Bytes is employed for the key expansion phase and encryption phase. To optimize Sub-Bytes, we merge and simplify some parts of the Sub-Bytes. To reduce power consumption, we apply the clock gating technique to the design. Section 4.1 presents the motivation and introduction of the design; the proposed 8-bit data-path AES architecture and its blocks are presented in Section 4.2; section 4.3 presents the implementation results, analysis, and comparison with other similar works;

The content of this chapter is originally published in: K. Shahbazi and S. -B. Ko, "Area-Efficient Nano-AES Implementation for Internet-of-Things Devices," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 1, pp. 136-148, Jan. 2021, doi: 10.1109/TVLSI.2020.3033928. The manuscript has been reformatted for inclusion in this thesis.

4.1 Introduction

Internet of things (IoT) is a wide network that a myriad of tiny devices are connected to each other and transmitted data. This information could be part of a smart environment, such as e-health, public transportation, and so on; these tiny devices could be part of a high-tech network from sensing technology, communication technology, data processing, to cloud computing, and artificial intelligence. As the number of connected devices has been exponentially increased, providing security for all the transmitted information in most scenarios is not an easy task. The main reason is that most of the end-node tiny devices do not have enough resources for the cryptography part. Due to a fast-growing IoT environment, designing low-cost cryptography architecture is an important research area. Lightweight cryptography architectures are particularly interesting for battery-powered devices, which are widely used in IoT end node devices.

Advanced Encryption Standard (AES) is one of the secure symmetric cryptography algorithms that is widely used in different networks and is the main security part in various applications, platforms, and networks, such as IoT, LoraWan [57], and in other Internet standards. According to the length of the key, AES provides different security levels. Based on [6], the AES algorithm with a 256-bit key is secure enough in the quantum era, which means this algorithm can provide the security requirement of different levels of IoT applications and protocols. Some of the disadvantages of the software implementation of AES are the high latency for processing the data and transmission and consumption of more power [17]. As a result, the tendency of hardware implementation has been increased for high-performance applications and resource-constrained devices. The AES implementations for resource-constrained devices contain a serial architecture with 8, 16, or 32-bit data-path, which leads to low throughput.

In this work, we propose a 8-bit data-path architecture for resource-constrained devices or mobile SoCs. In comparison to 32-bit data-path, 8-bit data-path occupies fewer internal wires. We try to reduce the number of blocks, employ the low area design, and try to merge functions to reduce the size of the design. Among the other blocks, the proposed

design includes two specified register banks, State-Register and Key-Register, that are based on shift-register memory and used for storing keys, plain-text, and intermediate results. These two registers have a major role during the key expansion and encryption phase. We implement the proposed architecture on FPGA and ASIC 65*nm* technology. Based on the NIST lightweight cryptography [58], the ASIC implementation of the proposed architecture is proper for crypto-systems in resource-constrained IoT devices. The hardware implementation results appear to be better than previous works. Our proposed design improves the area and Area-Delay-Product (ADP) for chip design by 2.4% and 71.7%, respectively; and improves the core area with power rings by 22.1% on 65*nm* technology compared with state-of-the-art works.

The main contribution of this chapter is to design a lightweight AES architecture for IoT resource-constrained devices. To achieve this goal, some of the best implementation techniques and design specified blocks according to the mentioned goals are employed as follows:

- In order to reduce the required logic, the Shift-Rows is embedded inside the State-Register.
- We optimize Sub-Bytes block and share it with key expansion phase and encryption phase, which results in 15.5% area reduction.
- Although Mix-Columns for executing requires 32-bit at the same time, we design an optimized 8-bit block for Mix-Columns with 8-bit input and output that is based on the structure of 8-bit data-path, which is followed by Add-Round-Key. Thus, the results send to Add-Round-Key byte-by-byte. In comparison to 32-bit Mix-columns, it is not necessary to store the results in the registers or increase the data-path for Key-Register to 32-bit.
- To reduce the power consumption of the design, the clock gating technique is applied in different parts of the design, which leads to reduce the power consumption by 18.9% on 65*nm* technology.

4.2 8-Bit nano-AES data path accelerator

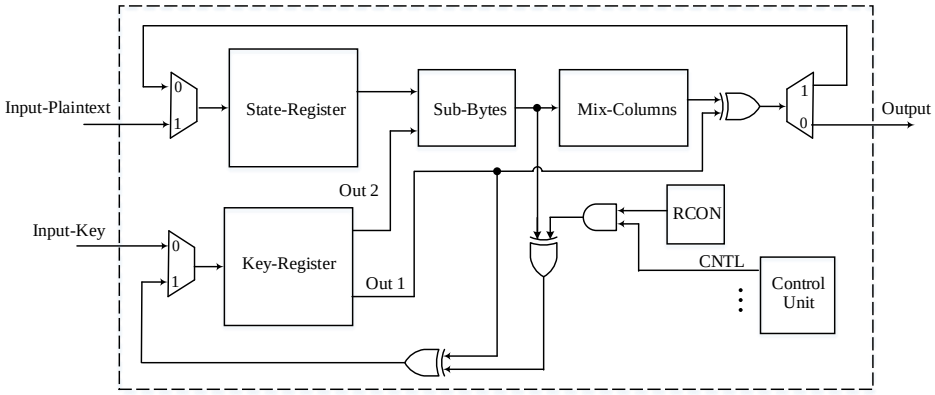


Figure 4.1: The architecture of the proposed nano-AES design

In this section, we explain the architecture of the proposed 8-bit data-path nano-AES. Our proposed architecture is presented in Figure 4.1. The design includes one Sub-Bytes for both key expansion and encryption, one 8-bit Mix-columns, two register banks for storing keys (called Key-Register) and plain-text (called State-Register), which also act as temporary registers for storing the intermediate results, RCON block, and control unit. Furthermore, a bypass circuit is employed in the Mix-Columns and Sub-Bytes to avoid unnecessary operations. In order to store intermediate results into register banks, the design has two feedback paths: one in the key expansion and return the key into the key bank, and the other one in the encryption path.

Figure 4.2 shows the structure of the proposed State-Register. The State-Register consists of sixteen 8-bit registers, each register contains 8 flip-flops. State-Register is based on a shift-register memory topology, which in each cycle one 8-bit is fed to the design and one 8-bit is stored in the State-Register if needed. To reduce the area, we do not design a specific block for Shift-Rows. Since Shift-Rows and Sub-Bytes are executed on each byte of the input, these two functions can be exchanged without any effect on final results. We calculated the final results after Shift-Rows function, and applied it to the State-Register. One of the State-Register's duty is to execute Shift-Rows. Each register of State-Register contains one or two inputs that requires a multiplexer to select from two inputs. One input

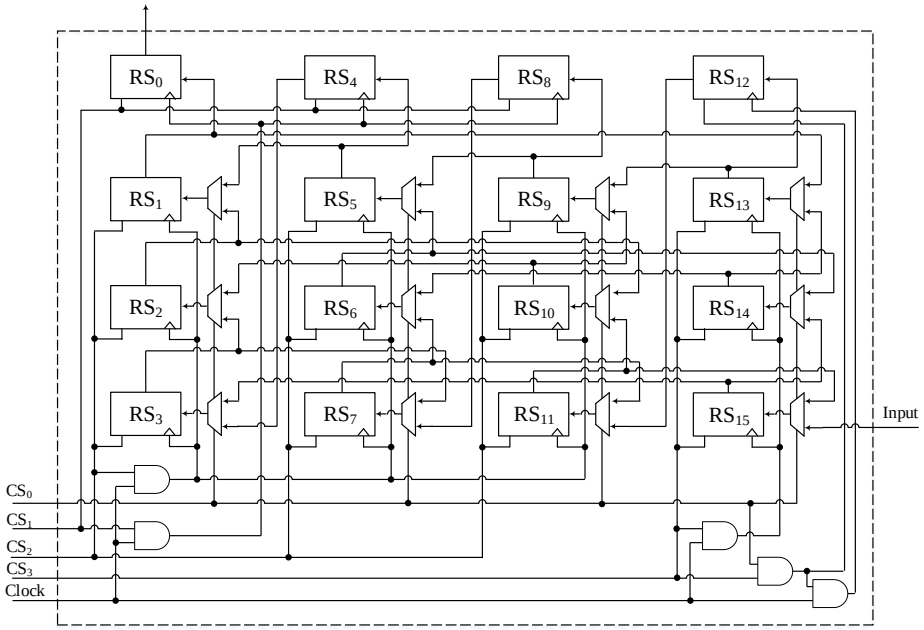


Figure 4.2: The structure of the proposed State-Register with Shift-Rows, control circuitry, and clock gating technique

receives the data from the previous register to execute the encryption phase and pass the data between internal registers (The vertical interconnections among registers in each column in Figure 4.2). The other one includes some interconnections between different units of the State-Register to execute Shift-Rows (horizontal interconnections among registers in each row in Figure 4.2). Thus, Shift-Rows is done by wiring; and this completely removes the logic for Shift-Rows step, based on [59].

[60] designed a specific block for executing Shift-Rows that required twelve registers. A specific block for executing Shift-Rows occupies more area and is not suitable for lightweight design. Compared to the proposed embedded Shift-Rows inside the State-Register, if we apply the Shift-Rows of [60] into the proposed State-Register, the Shift-Rows is executed in more clock cycles (150 more clock cycles for execution the crypto-system) and has a complicated Control-Unit for activating signals. [19] borrowed the design of Shift-Rows from [60]. The Shift-Register of [19] contained three 2-1 MUXs and one 4-1 MUX; the proposed design contains twelve 2-1 MUXs. In [19], each MUX and some of the

registers has its own control signal that makes the control unit complicated. As a result, it causes increasing the switching factor followed by increasing power consumption (the power consumption is available in Table 4.3).

Shift-Rows and Mix-Columns are permutation operation on the rows and on the columns, respectively. Mix-Columns needs the data of one column for execution. As there is a specific 8-bit block for executing the Mix-Columns, the way of storing and feeding data to the design from the State-Register is column-based; moreover, after four clock cycles, one column of the State-Register is fed to the Mix-Columns block. Then the results are sent back to the store register in four clock cycles. To access the registers in each operation, the State-Register has four control signals (CS_3, CS_2, CS_1, and CS_0). CS_0 is MUXs selection; CS_1 is an activation signal for registers in the first row; CS_2 is an activation signal for RS_1, RS_2, RS_3, RS_5, RS_6, RS_7, RS_9, RS_{10}, and RS_{11}; CS_3 is an activation signal for RS_{13}, RS_{14}, and RS_{15}; the AND result of CS_0 and CS_3 is an activation signal for RS_{12}. The State-Register has five main operations:

1. Storing the main plain-text. To store the main plain-text into the State-Register, all of the internal registers should be activated; thus, all of the control signals should be set to '1'. In each clock cycle, one 8-bit data is fed to the design and stored in RS_{15} (in Figure 4.2). Based on the shift-register memory and interconnection between units, when a new 8-bit data comes, the stored data is transmitted from RS_{15} to RS_0.

2. Executing Shift-Rows. As it is mentioned, there are internal intersections between registers; and Shift-Rows is done by wiring over the internal registers and interconnection. To execute Shift-Rows, the registers in the second, third, and fourth columns of the State-Register should be activated. Thus, CS_0 and CS_1 should be '0'; and CS_2 and CS_3 should be '1'.

3. Feeding the design by one 8-bit and storing the data at the same time based on shift-register memory for the first Add-Round-Key and the last round, which do not include Mix-Columns. In this operation, all of the internal registers should be activated.

4. Feeding the design by the saved data in the first column of the State-Register (from RS_0 to RS_3) by one 8-bit, for four clock cycles, and shift the register's data in order to execute Mix-columns (all of the control signals should be set to '1'). After four clock cycles, the data is shifted in which the fourth column is ready to store the result of Mix-Columns.

5. Storing the data that comes from Mix-Columns only in the last column. As it is explained in section 4.2.2, storing the calculated results of Mix-Columns into State-Register requires four clock cycles; and data should be stored in the last column of the State-Register. As a result, only data in the last column is shifted to store the coming data from Mix-Columns block; and data in the other columns is not transmitted. During this operation, the connection of the fourth and third columns of State-Register will be cut off by deactivating the internal registers in the first, second, and third columns ($CS_3CS_2CS_1CS_0$ will be set to '1001').

The data movement of State-Register for the Add-Round-Key and the first round is available in Table 4.1; the value of registers will be repeated for the other rounds.

4.2.1 Sub-Bytes Optimization

Sub-Bytes is one of the most critical parts of the AES design in terms of power, area, and latency. Our design contains one Sub-Bytes, which reduces the required silicon resources. The single Sub-Bytes is used for both the encryption phase and for key expansion. There are different methods of implementation of this block. Although the most straightforward way of implementing is using look-up-table (LUT), such as [13] [14], or employing the Boolean simplification map (by using truth table in order to make a direct relation between the parameter of the Sub-Bytes) [15] [16], these two methods occupy more area and are not suitable for area-restricted devices. Decode–Switch–Encode (DSE), which is employed by [17] [18], is another method for implementation Sub-Bytes that is a good option for low power architecture; however, it occupies a larger area. The efficient way of implementing Sub-Bytes is to use composite field arithmetic, such as [19] [20] [21]. Sub-Bytes contains calculating the

Table 4.1: The content of State-Register during different operations for the first round

t	RS_{15}	RS_{14}	RS_{13}	RS_{12}	RS_{11}	RS_{10}	RS_9	RS_8	RS_7	RS_6	RS_5	RS_4	RS_3	RS_2	RS_1	RS_0	Operation
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Initial value
0	a_0'	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	$RS_{15} \leftarrow \text{ARK}(a_0)$
1	a_1'	a_0'	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	$RS_{15} \leftarrow \text{ARK}(a_1)$
									$\vdots$								
15	a_{15}'	a_{14}'	a_{13}'	a_{12}'	a_{11}'	a_{10}'	a_9'	a_8'	a_7'	a_6'	a_5'	a_4'	a_3'	a_2'	a_1'	a_0'	$RS_{15} \leftarrow \text{ARK}(a_{15})$
16	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	a_{15}'	a_{10}'	a_5'	a_0'	Shift-Rows
17	a_{11}'	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	a_{15}'	a_{10}'	a_5'	$RM_0 \leftarrow \text{Sub-Bytes}(a_0')$
18	a_{11}'	a_{11}'	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	a_{15}'	a_{10}'	$RM_0 \leftarrow \text{Sub-Bytes}(a_5')$
19	a_{11}'	a_{11}'	a_{11}'	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	a_{15}'	$RM_0 \leftarrow \text{Sub-Bytes}(a_{10}')$
20	a_{11}'	a_{11}'	a_{11}'	a_{11}'	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	$RM_0 \leftarrow \text{Sub-Bytes}(a_{15}')$
21	b_0	a_{11}'	a_{11}'	a_{11}'	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	$b_0 \leftarrow \text{ARK}(RM_3)$
22	b_1	b_0	a_{11}'	a_{11}'	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	$b_1 \leftarrow \text{ARK}(RM_3)$
23	b_2	b_1	b_0	a_{11}'	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	$b_2 \leftarrow \text{ARK}(RM_3)$
24	b_3	b_2	b_1	b_0	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	a_4'	$b_3 \leftarrow \text{ARK}(RM_3)$
25	b_3	b_3	b_2	b_1	b_0	a_{11}'	a_6'	a_1'	a_{12}'	a_7'	a_2'	a_{13}'	a_8'	a_3'	a_{14}'	a_9'	$RM_0 \leftarrow \text{Sub-Bytes}(a_4')$
									$\vdots$								
48	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	The last value

RSs are internal registers of State-Register.

RMs are internal registers of Mix-Columns. The movement of internal registers of Mix-Columns is shown in Figure 4.6.

ARK stands for Add-Round-Key operation, which is XOR with keys.

multiplicative inverse of $f(x)$ that is $g(x)$ in which $f(x) \cdot g(x) mod(x^8 + x^4 + x^3 + x + 1) = 1$ followed by Affine Transformation (AT). Calculating the multiplicative inverse in $GF(2^8)$ is complicated in which employing the composite field arithmetic reduces the complexity. By using composite field, the multiplicative inverse is calculated by decomposing $GF(2^8)$ to the lower fields.

In the proposed design, composite field arithmetic is optimized and used followed by the Affine Transformation of [3], which is an efficient method for area-restricted devices. Based on the previous work [3], we select the irreducible polynomials based on equation (4.1) that decomposed $GF(2^8)$ to the lower fields $GF(2^1)$, $GF(2^2)$, and $GF((2^2)^2)$. As the Sub-Byte design of [3] was one of the efficient designs, we borrow their parameters. In equation (4.1), $\lambda = \{1100\}_2$ in the $GF(2^4)$ and $\varphi = \{10\}_2$ in the $GF(2^2)$. By mapping $GF(2^8)$ to its lower field, each element 'A' can be represented by $A_h x + A_l$, where A_h is the most significant part, and A_l is the least significant part based on the irreducible polynomial $x^2 + x + \lambda$. By selecting the irreducible polynomial (equation (4.1)), the multiplicative inverse is calculated by equation (4.2) [3].

$$\begin{aligned} &GF(2^2) \rightarrow GF(2) && : x^2 + x + 1 \\ &GF((2^2)^2) \rightarrow GF(2^2) && : x^2 + x + \varphi \\ &GF(((2^2)^2)^2) \rightarrow GF((2^2)^2) && : x^2 + x + \lambda \end{aligned} \tag{4.1}$$

$$\begin{aligned} (A_h x + A_l)^{-1} = A_h(A_h^2\lambda + A_l(A_h + A_l))^{-1}x + \\ (A_h + A_l)(A_h^2\lambda + A_l(A_h + A_l))^{-1} \end{aligned} \tag{4.2}$$

A straightforward method for generating a transformation matrix to map elements of $GF(2^8)$ to $GF((2^4)^2)$ is explained in details in [61]. The isomorphic function, 'δ' is used to transfer between $GF(2^8)$ to its composite field; and the inverse isomorphic, 'δ^{-1}', is used to transfer from composite field to its value in $GF(2^8)$. 'δ' is a 8×8 binary matrix that is calculated by the field polynomials of $GF(2^8)$ and its composite fields. Similar to [3], [62], and [63], by selecting $P(x) = x^8 + x^4 + x^3 + x + 1$ and using the sub-field of equation (4.1), the isomorphic function can be calculated as equation (4.3). The inverse isomorphic function

'δ^{-1}', which is written as equation (4.4), is constructed by inverting the 8×8 matrix 'δ'.

$$\delta(x) = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \tag{4.3}$$

$$\delta^{-1}(x) = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \tag{4.4}$$

After calculating the multiplication inverse followed by inverse isomorphic function (δ^{-1}), the Affine Transformation (AT) is applied to achieve the final result. In equation (4.5), AT is Affine Transformation, $f(x)$ is the result from the multiplication inverse followed by inverse isomorphic function, ϕ is a constant number (based on [64] ϕ is equal to $\{63\}_8$), and $g(x)$ is the final result of Sub-Bytes block.

$$g(x) = AT(f(x)) + \phi = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{4.5}$$

The Sub-Bytes of $f(x)$ is $g(x)$ and is calculated by equation (4.6). In this equation, $IMUL$ is the multiplicative inverse. We defined $(AT \times \delta^{-1}) \oplus \phi$ as γ and rewrote the equation.

$$\begin{aligned} g(x) &= Sub(f(x)) \\ &= (IMUL(\delta \times f(x)) \times \delta^{-1}) \times AT \oplus \phi \\ &= (IMUL(\delta \times f(x))) \times (\delta^{-1} \times AT \oplus \phi) \\ &= (IMUL(\delta \times f(x))) \times \gamma \end{aligned} \tag{4.6}$$

Thus, the combination of inverse isomorphic (equation (4.4)) and Affine Transformation (equation (4.5)) is used in the proposed architecture of Sub-Bytes. After doing the multiplication of δ^{-1} by AT followed by ϕ, γ obtained, which is written in equation (4.7).

The optimized architecture of γ is drawn in Figure 4.3. The proposed architecture of γ is implemented by twelve XOR gates, three XNOR gates, and one NOT gate. This module of the proposed Sub-Bytes reduced the area by 6.1% and 19% compared to [20] and [3], respectively (based on gate equivalent estimation on 65nm [65]).

$$
\begin{aligned}
\gamma_7 &= x_7 \oplus x_3 \oplus x_2 \\
\gamma_6 &= \overline{x_7 \oplus x_6 \oplus x_5 \oplus x_4} \\
\gamma_5 &= \overline{x_7 \oplus x_2} \\
\gamma_4 &= x_7 \oplus x_4 \oplus x_1 \oplus x_0 \\
\gamma_3 &= x_2 \oplus x_1 \oplus x_0 \\
\gamma_2 &= x_6 \oplus x_5 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_0 \\
\gamma_1 &= \overline{x_7 \oplus x_0} \\
\gamma_0 &= \overline{x_7 \oplus x_6 \oplus x_2 \oplus x_1 \oplus x_0}
\end{aligned}
\tag{4.7}
$$

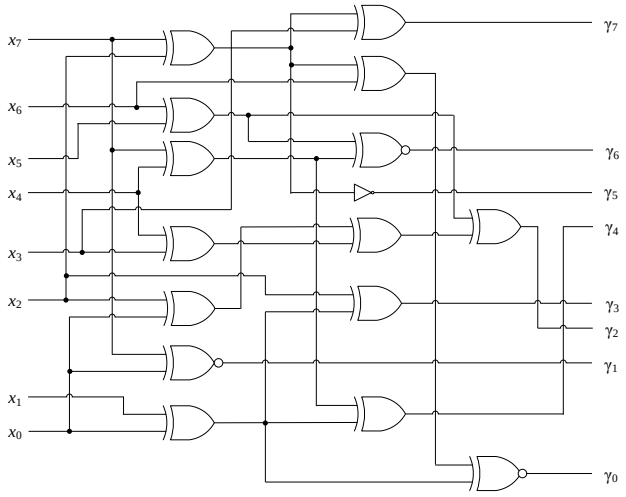


Figure 4.3: The optimized structure of combination of inverse isomorphic with Affine Transformation

Figure 4.4 shows the architecture of the proposed Sub-Bytes that includes the isomorphic function, 'γ' is the combination of inverse isomorphic and affine transformation, multiplica-

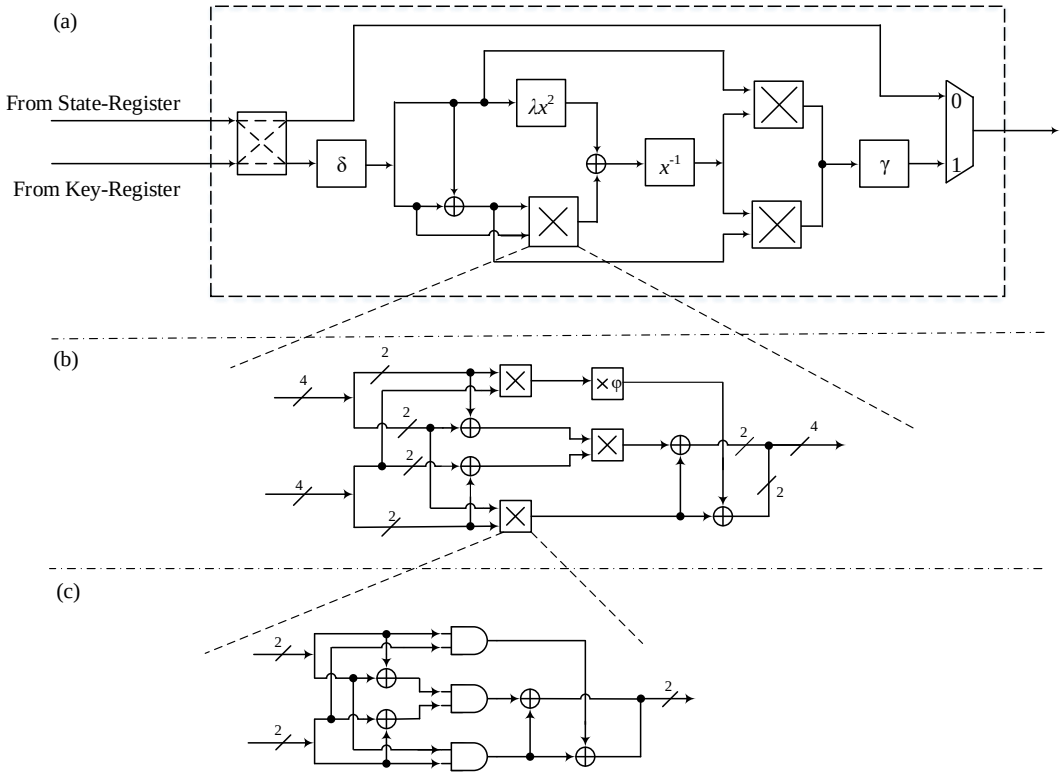


Figure 4.4: (a) The modified architecture of Sub-Bytes [3] with combination of inverse isomorphic with Affine Transformation (γ) and bypass circuit (b) The multiplication operation in $GF(2^4)$ (c) The multiplication operation in $GF(2^2)$

tive inverse based on composite field arithmetic. The proposed Sub-Bytes includes a bypass circuit to avoid the execution of this function for some phases.

4.2.2 8-bit Mix-Columns Optimization

As it was explained, the Mix-Columns is an 8-bit block, in which we employ four internal registers for storing the intermediate value of Mix-Columns. Each of the registers comprises an eight flip-flops with one or two inputs. For each column, the Mix-Columns is

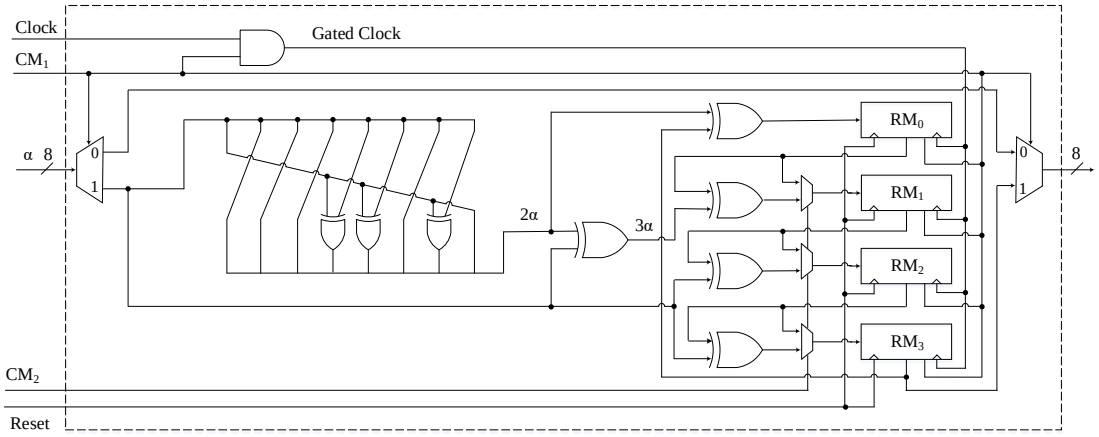


Figure 4.5: The proposed architecture of Mix-Columns with clock gating technique and bypass circuit

calculated by equation (4.8).

$$\begin{aligned} a'_0 &= 2.a_0 + 3.a_1 + a_2 + a_3 \\ a'_1 &= a_0 + 2.a_1 + 3.a_2 + a_3 \\ a'_2 &= a_0 + a_1 + 2.a_2 + 3.a_3 \\ a'_3 &= 3.a_0 + a_1 + a_2 + 2.a_3 \end{aligned} \tag{4.8}$$

For calculating the Mix-Columns using only one block, each element should be multiplied by 2 and 3 that $3.a$ can be calculated by $2.a+a$. In equation (4.8), '.' is a multiplication modulo with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. By considering $2.a$ and $m(x)$, $2.a$ is calculated as [66]:

$$(2.a) mod(x^8 + x^4 + x^3 + x + 1) = \begin{cases} x_7 = a_6 \\ x_6 = a_5 \\ x_5 = a_4 \\ x_4 = a_7 \oplus a_3 \\ x_3 = a_7 \oplus a_2 \\ x_2 = a_1 \\ x_1 = a_7 \oplus a_0 \\ x_0 = a_7 \end{cases} \tag{4.9}$$

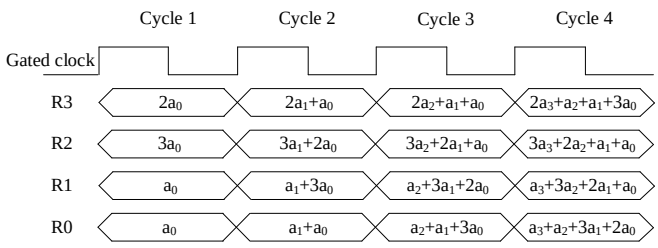


Figure 4.6: The timing diagram of the proposed Mix-Columns

Based on equation (4.9), to calculate 2.a, three XORs are needed. Figure 4.5 shows the proposed Mix-Columns block. As it is told, the Mix-Columns block includes a bypass circuit that is active for the first Add-Round-Key and the last round. For calculating Mix-Columns of each column of the state, a, 2.a, and 3.a should be available (according to equation (4.8)). Thus, the input data will be multiplied by 2, based on equation (4.9), followed by calculating 3.a. Then, a, 2.a, and 3.a are stored into internal registers of Mix-Columns (CM_2 should be '1'). After finishing the calculation of Mix-Columns for each column, the values of registers are sent one by one to the output bus followed by shifting vertically of the registers (from RM_0 to RM_3) by setting CM_2 to '0'. And finally, the internal registers will be set to zero to calculate Mix-Columns of the next column.

We depict the timing diagram for storing the data into internal registers of Mix-Columns in Figure 4.6. It takes four clock cycles to calculate one column of the State-Register and takes another four clock cycles to send from Mix-Columns' internal registers to State-Register. A complete Mix-Columns operation takes 32 clock cycles to transform the entire state. Unlike to the previous works, such as [27], the proposed Mix-Columns has a data path of 8-bit input and 8-bit output and only uses one block for calculating 2.a and 3.a that results to occupy less area.

4.2.3 Key Expansion

For each round of the algorithm, one 128-bit key is needed. The key in each round is generated from the previous round-key. Storing all of the keys in the design requires a huge memory, which is not a proper method for resource-constrained devices. On-the-fly key

expansion is a wise way that only requires one 128-bit register. The key expansion phase contains the shift of the last column of Key-Register, Sub-Bytes, RCON, and XOR. The key expansion phase is written in equation (4.10).

$$
\begin{aligned}
&Shift(Col_4) \\
&Col_1^{'} = Sub(Col_4) \oplus RCON \oplus Col_1 \\
&Col_2^{'} = Col_1^{'} \oplus Col_2 \\
&Col_3^{'} = Col_2^{'} \oplus Col_3 \\
&Col_4^{'} = Col_3^{'} \oplus Col_4
\end{aligned}
\tag{4.10}
$$

RCON is a block that provides constants values needed for the key expansion, which is related to the number of rounds and applied to the first element of the fourth column. As RCON includes constant values (these values are $\{01\}_8$, $\{02\}_8$, $\{04\}_8$, $\{08\}_8$, $\{10\}_8$, $\{20\}_8$, $\{40\}_8$, $\{80\}_8$, $\{1B\}_8$, and $\{36\}_8$), the straightforward implementation way is to use a LUT. The advantage of LUT implementation is to reduce power consumption and minimize the clock network [17]; however, it occupies more area and is not suitable for the proposed design. Previous works such as [20] used LUT for their RCON. [21] optimized RCON block by using the simple Karnaugh optimization. [27] implemented RCON by using the round counter and combinational logic.

Instead of $\{1B\}_8$ and $\{36\}_8$, the other numbers are gained by rotating a single '1' in an 8-bit shift register. Implementing $\{1B\}_8$ and $\{36\}_8$ requires additional logic gates. [26] implemented RCON by rotating a single '1' in an 8-bit shift register followed by four control signals, one 8-bit AND, one 4-bit NOR. We optimized the RCON design of [26] by a circular left shift register followed by *OR* gates and a control signal. The RCON block is shown in Figure 4.7.

The architecture of the proposed Key-Register is shown in Figure 4.8. We specify the Key-Register based on the proposed design. Key-Register includes sixteen 8-bit registers, which each register contains of 8 flip-flops with one or two input(s) and a 2-1 MUX, to store the keys. It also contains one 8-bit input and two 8-bit outputs. The reason that we design

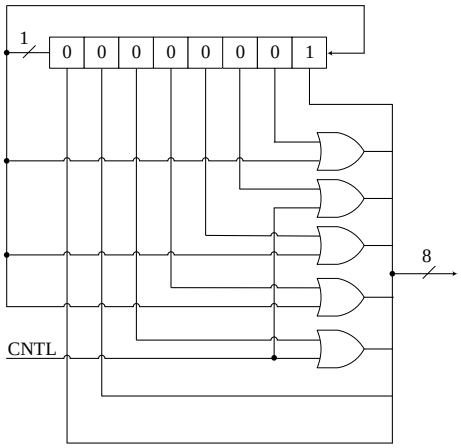


Figure 4.7: The proposed RCON block of the proposed design

the Key-Register with two outputs is that (according to the equation (4.10)) to expand the keys, two columns of the previous keys are required at the same time. Key-Register includes five main operations: first, store the initial keys in the Key-Register; second, feed the design by one 8-bit for encryption phase (from *Out 1* in Figure 4.1 and Figure 4.10) and store the same key again in the same cycle, thus after sixteen clock cycles the keys are stored in the same position before encryption phase; third, shift the fourth column; fourth, feed the design by the data that is stored in the first and fourth column and store the result at the same time; fifth, shift the third column. The third, fourth, and fifth operations are used for the key expansion phase.

The control part of Key-Register contains a *Clear* signal to set the registers into zero (this signal is not shown in Figure 4.8), and a 3-bit, $CK_2CK_1CK_0$, signal to execute the operations based on the phase of the design. CK_1 and CK_2 are used for activating the internal registers; CK_0 is control signal for Muxs. The description of control signal is available in Table 4.2. It is worth mentioning that the other values of control signals will not occur. The movement of Key-Register's values for executing one round of key expansion is depicted in Figure 4.9. In Figure 4.9, the values with apostrophe are expanded key's values. Based on equation (4.10) and the operations of Key-Register, which is explained before, the key expansion is executed by (the initial value of keys is shown in Figure 4.9 (a)):

Table 4.2: The description of control signal for Key-Register

$CK_2CK_1CK_0$	Description
111	All of the internal registers are activated; and all MUXs are set to '1'.
100	The internal registers of the fourth column of the Key-Register is activated to execute shift.
010	The internal registers of the third column of the Key-Register is activated to execute shift.

The other value of control signals will not occur

1. The fourth column is shifted (Figure 4.9 (b)).

2. The control signal of the RCON is activated. The first element of the first column ('RK_0' in Figure 4.8) is fed to the design. The first element of the fourth column goes to the Sub-Bytes block (RK_{12} in Figure 4.8). Then, the result is sent back to the Key-Register to store in 'RK_{15}', and all other elements are shifted by one element (from 'RK_{15}' to 'RK_0'), based on shift-register memory topology. This step is shown in Figure 4.9 (c).

3. The control signal of RCON is de-activated; the other three elements of the first and the fourth columns are fed to the design, and the results are stored in the fourth column. The execution of this step is shown in Figure 4.9 (d) to Figure 4.9 (f).

4. Thus, the first column of the new expanded key is stored in the fourth column, and the fourth column of the old keys is transmitted to the third column. The third column is shifted to put the element of the fourth column of the old keys in its correct position (this operation is done by the bi-directional data movement of the third column of Key-Register in Figure 4.9 (g)).

5. In each cycle, one element of the first and fourth columns are fed to the design, and

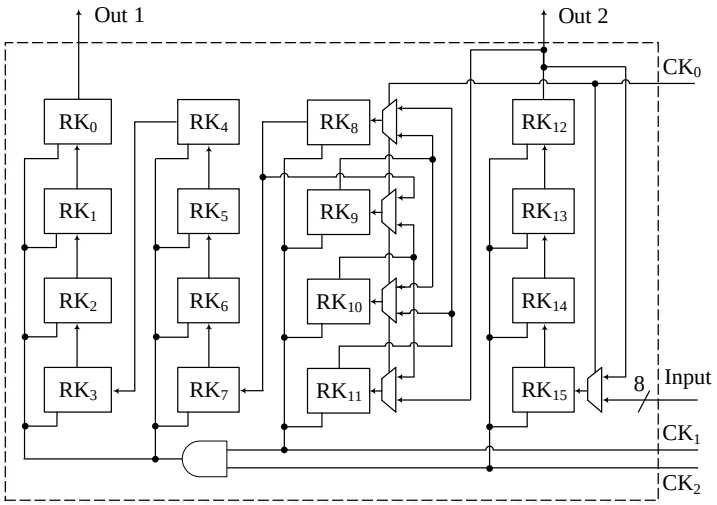


Figure 4.8: The structure of the proposed Key-Register

the result is sent back to the Key-Register. This step is continued over other columns of the Key-Register Figure 4.9 (h) to Figure 4.9 (i).

Only the last column of Key-Register needs Sub-Bytes. To minimize area overheads, Sub-Bytes is shared between key generation phase and data encryption phase, resulting in 15.5% area reduction on 65*nm* technology. It is worth mentioning that the structure of Key-Register is independent of other parts of the main architecture; and for expanding keys no more registers are needed. However, the effect of a shared data-path between key expansion and data encryption is increasing the latency as these two phases cannot be executed simultaneously. Compared to [27] that used two Sub-Bytes, the proposed design only use one Sub-Bytes. Also, the key expansion block of [27] had two inputs and three outputs; however, the proposed Key-Register has only one input and two outputs. Besides that, [27] used one more 8-bit register to store the intermediate results, which in the proposed design it is not necessary. As a result, these differences have a high impact on reducing the area over [27].

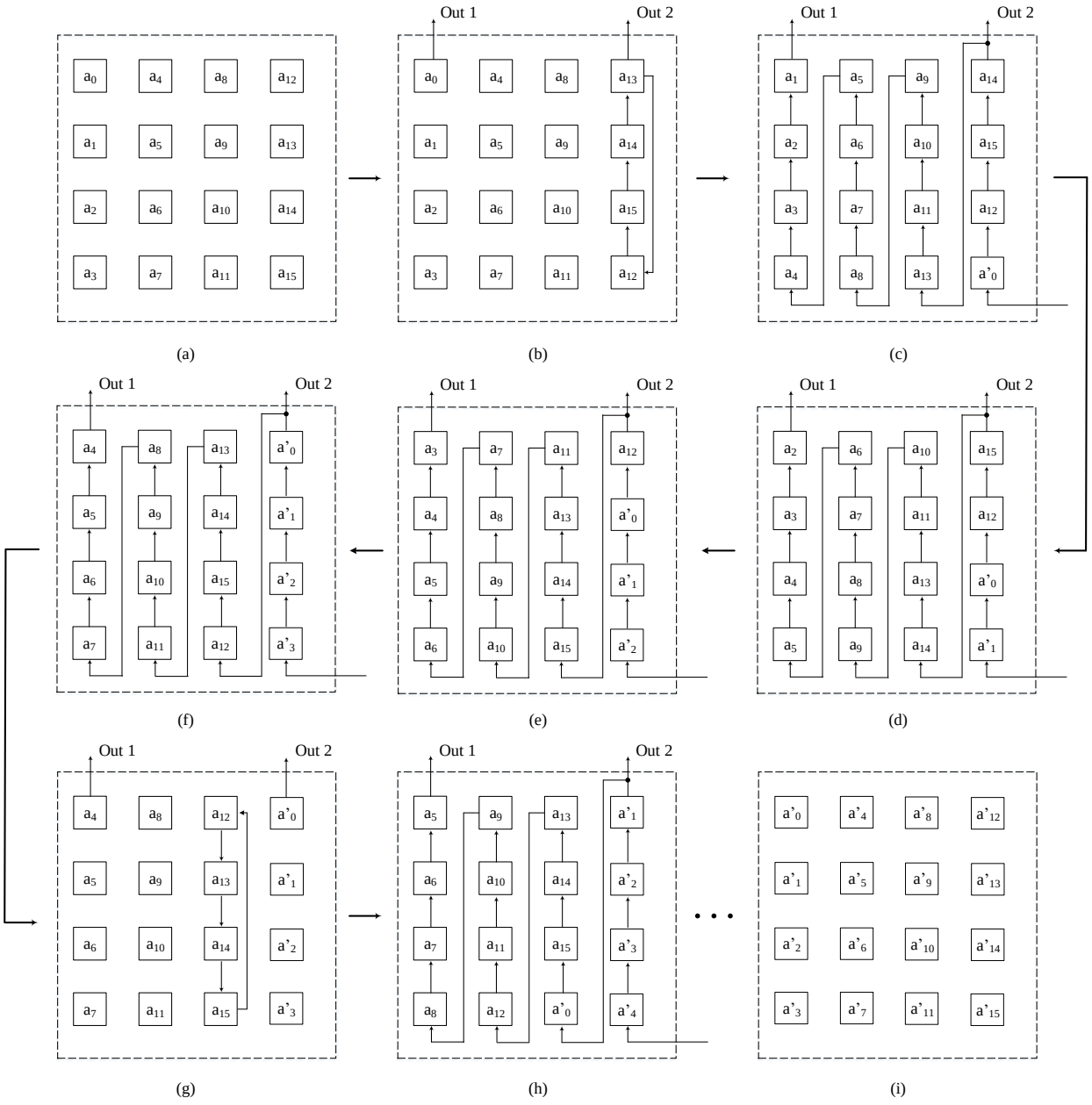


Figure 4.9: The movement of Key-Register's values for executing one round of key expansion. (a) The initial values of Key-Register. (b) The shift operation of the fourth column. (c) (d) (e) (f) (h) The first element of the first and fourth columns are fed to the design and the result is stored in Key-Register (g) The shift operation of the third column. (i) The Key-Register with the expanded key.

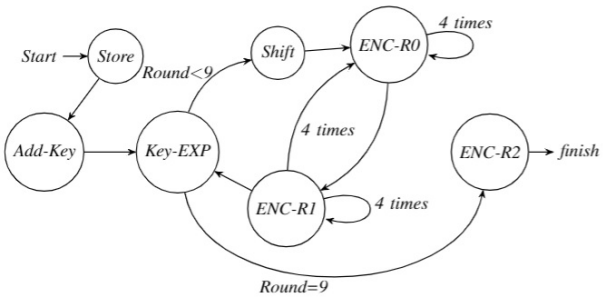


Figure 4.10: The finite state machine for the proposed design

4.2.4 Control Unit

The finite state machine (FSM) of the proposed nano-AES accelerator is drawn in Figure 4.10. The initial key and plain-text are stored in Key-Register and State-Register at the same time. The first Add-Round-Key is executed by activating the bypass signals for Sub-Bytes and Mix-Columns blocks. After the first Add-Round-key and execution each round, the key is expanded in *Key-EXP*. Shift-Rows executes in one clock cycle inside the State-Register. In *ENC-0*, one column of State-Register goes through Sub-Bytes and Mix-Columns and stores in the Mix-Columns registers that takes four clock cycles for execution Mix-Columns over one column. In *ENC-1*, the stored data in Mix-Columns' registers are sent back to State-Register followed by *XORing* with keys in another four clock cycles. The execution of one round takes 32 clock cycles. After nine times of running *Key-EXP*, *Shift*, *ENC-0*, and *ENC-1*, the state of control unit goes to *ENC-2* that the bypass signal for Mix-Columns is active, and the data are sent to output signals. The control unit includes one 4-bit and 2-bit counters. As we explained before, during the data encryption, the Key-Register and State-Register put one element in data-path and store one element according to the shift-register topology. The total latency of the design is 527 clock cycles.

In the different parts of the design, we apply the clock gating technique to reduce the dynamic power consumption. The clock gating is separately applied on State-Register, the internal registers of Mix-Columns, Key-Register, and RCON. For instance, the most power consumption is saved during the key expansion phase; the clock of State-Register

and Mix-Columns are disabled to save power because these two blocks are not used in the key expansion phase. The timing diagram of the proposed design with the clock gating technique is drawn in Figure 4.11. The effect of applying the clock gating technique for reducing the dynamic power of the design is 18.9%.

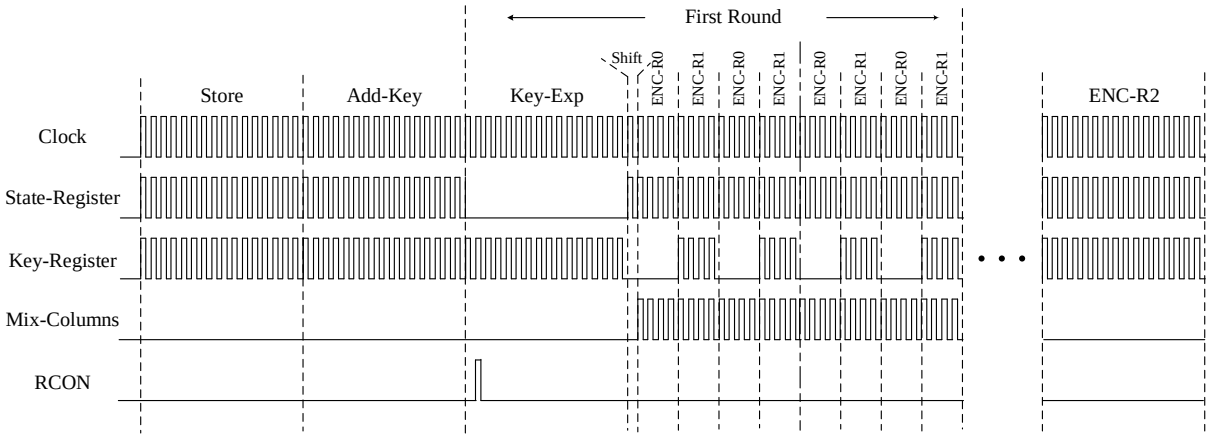


Figure 4.11: The timing diagram with clock gating technique of different blocks of the proposed design

4.3 Implementation Results and Analysis

In this section, the implementation results of the proposed architecture and its analysis are discussed. We implement the proposed nano-AES accelerator on both ASIC and FPGA platforms with VHDL. We use the test vector sets from NIST [64] to make the functional tests of the design. Also, different test vectors are generated using Python language. These test vectors are used to simulate the proposed design in ISim and used to validate the output of the proposed architecture on FPGA and ASIC implementation and estimate the power consumption on ASIC implementation. The proposed design is implemented on FPGA (Virtex-5). After that, by doing some minor changes on control unit and signals, the proposed design is synthesized in Synopsys Design Compiler using TSMC-65*nm* technology by the standard library cells with typical case parameters 1.1V and 25°C to generate timing, area, and power metrics. The proposed design is synthesized

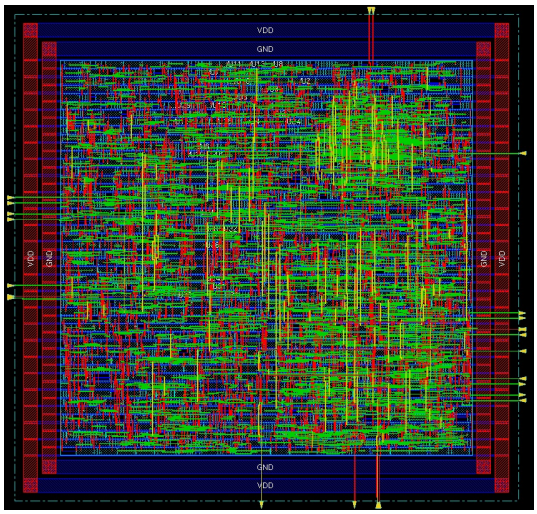


Figure 4.12: Layout of the proposed 8-bit AES core using 65nm technology

under different timing constraints (10, 20, 30, 40, 50, 100, 1000, and 10000ns) with a 10% input/output delay margin. The reason that we use clock frequency of 1MHz and 100KHz is that these frequencies are widely used operating for RFID applications. The maximum frequency of the design at 1.1V is 500MHz.

The layout of proposed 8-bit AES core is depicted in Figure 4.12. The area of the design contains 5448.59μm^2, 7783.77μm^2, and 11713.57μm^2, on core without power rings, core with power rings, and chip area on 65nm, respectively. Figure 4.13 depicts the percentage of the occupied area and power consumption for various building blocks. Approximately 58.4% of the design is occupied by the Key-Register and State-Register; and the Sub-Bytes and Mix-Columns consumes 25.7% of the area. As it can be observed, the area summation of Sub-Bytes and Mix-Columns is not dominant in nano-AES design; the register banks for storing plain-text and keys take up the largest portion of the total area because typically flip-flops are used, which require high area.

The detailed ASIC results and comparison with other similar works are written in Table 4.3. As it is obvious from Table 4.3, the proposed design occupies less area in comparison to similar works on the same technology. [23] has a 32-bit data-path with one shared Sub-Bytes. Their design included a big 20-to-1 8-bit MUX, and four 32-bit registers for storing

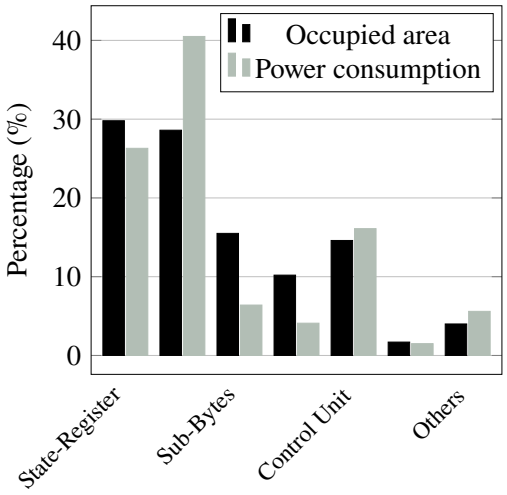


Figure 4.13: The percentage of occupied area and power consumption of different blocks of the proposed design on 65nm.

the plain-text and intermediate results, each 32-bit register has four 8-bit outputs and three 32-bit and one 8-bit input, and one 128-bit register for storing keys that occupy more area. The main goal of [20] was to design a low power AES architecture. They considered two Sub-Bytes and one specific block for Shift-Rows, LUT for storing RCON, and two more registers for storing the intermediate results; also, their Mix-Columns block was 32-bit, which contained eight modules to calculate 2.a and 3.a. Their work used more areas and resources than the proposed design.

The design of [19] transmitted Sub-Bytes and Mix-columns to their native functions. Also, their design included two native Sub-Bytes, which expanded key simultaneously. This native design, followed by two Sub-Bytes, led to an increase in the area by adding more blocks to the design. The authors of [24] also considered a specific block for Shift-Rows that contained eight 8-bit registers. They designed two different architectures with one Sub-Bytes and two Sub-Bytes with different values of data-path. Their architecture included more registers for storing keys, data, and intermediate results. There were a big 32×8-bit RAM and one internal register to store the intermediate results in [25]. [21] did not explain more details about their architecture. However, their architecture had a 8-bit data-path that

contained two Sub-Bytes, a Mix-Columns block with two 8-bit outputs, a parallel to serial converter, and a byte permutation unit. They did not report the frequency of their design for their simulated results.

In comparison to the previous similar works, we reduce the size of the internal memories (which is used to store the plain-text, keys, and intermediate results) and the number of blocks of the design. Also, we design an advanced register for Key-Register and State-Register in which some tasks can be done by these two registers, such as Shift-Rows in the State-Register. Furthermore, the data-path of the proposed design is 8-bit, which has a high impact on reducing the occupied area. According to the data-path of the architecture, we design an 8-bit Mix-columns. In comparison to other designs that were implemented on the same technology, the proposed design improves the area by 22.1% over [23] (core area with power rings) and 2.4% over [19] (chip area). Also, we calculated the Area-Delay-Product (ADP) of the proposed design and [19]. ADP is a common metric for evaluating the effect of executing time over occupied area on ASIC implementation. The ADP of the proposed design at 100MHz is $61.73\times10^3\mu s\times\mu m^2$. The ADP of [19] is equal to $218.2\times10^3\mu s\times\mu m^2$. The proposed design improved the ADP by 71.7% over [19].

To show the superiority of the proposed design against other similar works that were implemented on different technology, we also compare the proposed design with [26]. They designed a nano-AES on $22nm$ CMOS technology. They used one Sub-Bytes among Shift-Rows and a Mix-Columns block, which included 8-bit input and 32-bit output, that encrypted data in 336 clock cycles. Also, they employed three register banks (each contained sixteen 8-bit registers) for storing keys, plain-text, and intermediate results. The design of their RCON included an 8-bit shift register followed by four control signals, one 8-bit AND gate, and one 4-bit NOR gate, which compared to the proposed design occupied more area and made the control unit more complicated. Compared to the proposed design that the result of Mix-Columns is sent back to the State-Register, in [26], however, the result of Mix-Columns is sent to the intermediate registers; after that Shift-Rows is applied and data is stored into another register bank. Similar to the proposed design, the key expansion

of [26] should be done in a separate phase. That is why adding the intermediate register does not reduce the number of clock cycles significantly in [26] (compared to [27] that executed in 160 clock cycles). As they used different technology and in order to make a fair comparison, we borrow their normalized results that were calculated by [23]. Their normalized area was $19000\mu m^2$ on $65nm$ technology that the proposed design reduces the area by 59%.

Although low power design was not the main goal of this work, we calculate and report the power consumption of the proposed architecture. To calculate the power, we perform all simulations by using ISim; and then, SAIF files are generated from these simulations. After that, we simulate the proposed design using the SAIF files by the synthesis tool to calculate the average power consumption. We implement the proposed design with the clock gating technique and without the clock gating technique to see its impact on power consumption. The power-delay curve with and without the clock gating technique of the proposed design is drawn in Figure 4.14. By applying the clock gating technique, the dynamic power of the design is reduced by 18.9%. Figure 4.13 shows the average power consumption of each block by applying the clock gating technique. Key-Register is almost active in all cycles in the design; it, thus, consumes a high percentage of the total power. The power consumption of the embedded Shift-Rows function is approximately $4.96\mu W$ at 100MHz.

The power consumption for some RFID tag implementations is in the range of 1–$10\mu W$ with an operational frequency range of 0.1–1MHz [67] [68]. The power consumption of the proposed design is $3.32\mu W$ and $0.245\mu W$ for 1MHz and 0.1MHz. Thus, the proposed nano-AES accelerator can also be used as a crypto-system for RFID tags. Also, the energy consumption of the proposed crypto-system at 33.3MHz is equal to $1.3nj$. According to NIST report criteria [58], moreover, the proposed design can be supplied by low-power energy harvesting devices, such as Vibration Piezo or EM.

It is worth mentioning that power consumption for different standard cell libraries cannot be compared in a fair manner. However, we compare the proposed design with other similar works that were implemented on $65nm$ technology. The power consumption of a design

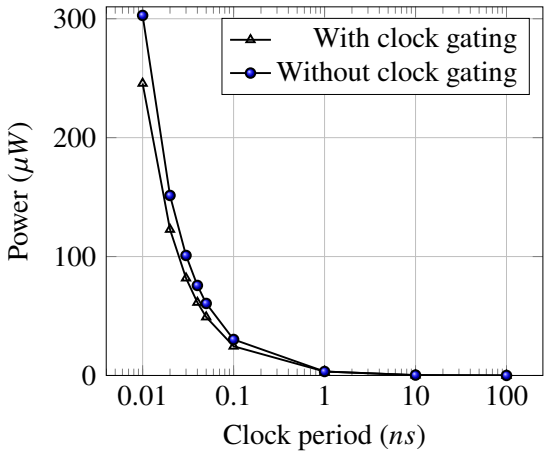


Figure 4.14: Power-delay curve with and without clock gating technique of the proposed design at 1.1V and 25°C.

is calculated by $P = CV^2 f$. As it is obvious, the load capacitance, supply voltage, and clock frequency have a high impact on power consumption. The load capacitance is one of the features of each technology and based on the area and the number of used gates on the die. The straightforward way for reducing the power consumption is to reduce the voltage and clock frequency of the design. As other previous works used different frequencies and different voltages, we normalized their power by using equation (4.11). In this equation, the letters with subscript '2' index are the new value that we want to consider, and the letters with subscript '1' index are the current values.

$$P_2 = P_1 \times (V_2/V_1)^2 \times (f_2/f_1) \tag{4.11}$$

The power of the proposed architecture is smaller than other works instead of [20]. The main goal of [20] was to design a low power AES crypto-system. Thus, they employed low power blocks and techniques, such as using LUT for RCON and low power Sub-Bytes, which some of the employed techniques were not area efficient. As the area was not their priority, their design occupied more than 45%, compared to the proposed design.

The design of [27] used two blocks for Sub-Bytes and 32-bit data path for Mix-Columns

that enabled parallel operation of data encryption along with on-the-fly key generation, which led to encrypting data in 160 clock cycles. Even though [27] executed in less clock cycles, as the delay of their design was high, the execution time for encrypting data of the proposed design and their design are approximately the same at the maximum frequency. The maximum frequency is 500MHz and 152MHz for the proposed design and the lightweight design of [27], respectively (the execution time is equal to $1.054\mu s$ and $1.052\mu s$, respectively). The power consumption of the proposed architecture is $1.35mW$ at 500MHz. Furthermore, the proposed design improved the power consumption by 58.7% over the normalized power of [27], which is $3.27mW$ at the maximum frequency (130MHz) of their low power design.

As it is explained, AES is a secure cryptography algorithm that is widely used in different platforms and applications. To make a comprehensive view for implementing on FPGA, we report the result of implementation on FPGA-Virtex-5. The total number of occupied slices is 191, and the maximum frequency is 147MHz.

Table 4.3: Results and Comparison for lightweight implementation on TSMC-65nm

Design	Data path	#Sub-Bytes	#Clock Cycles	Frequency (MHz)	Area ($\times 10^3 \mu m^2$)	Power (μW)	Normalized Power (μW)
This design[1*]	8-bit	1	527	100	11.7	245.60 @ 1.1 V	245.6
[23][2]	32-bit	1	242	200	10	3460 @ 1.2V	1453.6
[20][3]	8, 32-bit	2	186	10	>10	10.01 @ 0.9 V	149.53
[19][1]	8-bit	2	N/A	11	12	14.6 @0.5 V	642.4
[24][1]	8, 16,32, 64-bit	1	210	127.2	13	97.9 @ 0.55 V	307.862
[25][1]	8-bit	1	1142	0.322	18	0.85 @ 0.4 V	1996.31
[21][2]	8-bit	2	160	N/A	14.4	0.38 $\mu W/MHz$	N/A

[1] The reported area is chip area. [2] The reported area is core with power rings [3] The reported area is core without power rings.

[*] The area of the proposed design contains $5.4 \times 10^3 \mu m^2$ and $7.7 \times 10^3 \mu m^2$ for core without power rings and core with power rings, respectively.

Part III

Binary Ring-LWE implementations

5. Lightweight Design of Binary Ring-LWE

Internet of Things (IoT) connects a myriad of small devices over a huge network, encompassing many different and varied applications and environments. As the IoT network continues to grow, providing end-to-end security over IoT is becoming a paramount issue. To mitigate existing and future security risks within IoT, two important factors should be considered. First, some resource-constrained edge devices have an insufficient area to contain the security part. Second, the advent of quantum computers threatens the security of current public-key cryptography algorithms. In response to these challenges, lattice-based cryptography (LBC) has emerged as a promising technique for IoT security in the quantum era. The feasibility of LBC integration onto resource-constrained devices has been demonstrated in previous research. Multiplication is the main operation in Ring-BinLWE, a type of LBC. In this chapter, a new multiplication method is proposed, which is called In-place modular Reduction and anti-circular Rotation Column-based Multiplication (*In-place Rot-Col-Mul*), and new Ring-BinLWE architecture is designed. In-place Rot-Col-Mul performs a column-based multiplication in which one rotation is executed per cycle. The introduction of this chapter is in section 5.1. The proposed design, method, and implementations of Ring-BinLWE are described in Section 5.2. The implementation results and comparison are explained in Sections 5.3.

The content of this chapter is originally published in: K. Shahbazi, and Seok-Bum Ko "Area and power efficient post-quantum cryptosystem for IoT resource-constrained devices." Microprocessors and Microsystems. Vol. 84, PP 104280, July 2021, doi: 10.1016/j.micpro.2021.104280. The manuscript has been reformatted for inclusion in this thesis.

5.1 Introduction

Internet of things (IoT) is recognized as the future of an internet that contains billions of connected devices, in different applications and environments in which they are exchanging information. These applications and environments could include smart transportation, e-health, smart energy management, industrial control, or smart ecosystem management, and every day, the number of connected devices has been exponentially increasing. Providing end to end security in IoT is not an easy task as some resource-constrained devices do not have encryption safeguards. Because of that, they are at significant risk of eavesdropping. Designing very low power and low area hardware cryptography circuits is a growing topic of IoT for resource-constrained devices and mobile SoCs. One of the main issues and concerns in IoT is providing security against different attacks. Among different kinds of attacks, the cyberattack is one of the significant threats to a wide range of Internet services [69]. As a result, a reliable security system plays an essential role on the Internet.

Among different cryptography techniques, public-key cryptography (PKC), compared to private-key cryptography, provides more security over a large network, such as the IoT. One of the advantages of public-key over private-key is key management can be achieved effectively in wide-area networks. However, the current PKCs, such as Rivest-Shamir-Adleman (RSA) [70] and elliptic curve cryptography (ECC) [71], are not suitable for IoT nodes for two main reasons. The first is that they have more complex computations, which makes it hard to efficiently implement PKCs on IoT devices [69] [58]. Second, they are still vulnerable to quantum attacks [72]. The main reason for the latter is that the security of traditional PKC is based on the hardness of certain number theoretic problems (RSA is based on large integer factorization, and ECC is based on the discrete logarithm problem). According to Shor's algorithm [5], these problems can be solved very efficiently on a quantum computer; and cryptography algorithms will be insecure and weak as quantum computers increase in number. Therefore, it is critical to IoT security to design alternative crypto accelerators that are secure in the quantum era and provide long-term security.

According to NIST [7], lattice-based cryptography (LBC) is one of the accepted

quantum-resistant public-key cryptography, which no known quantum algorithms can solve the lattice problems [73]. In comparison to other post-quantum cryptography schemes, such as isogeny [74], LBC has relatively faster operations. As a result, LBC has received much attention in these years and has been selected as a great candidate for post-quantum cryptography system. It is worth mentioning that the post-quantum cryptosystems are hardware-independent.

Among the different varieties of LBC techniques, such as NTRU and learning with errors (LWE), Ring-LWE [8] is more practical and efficient in hardware and, compared to LWE, has a smaller key size [9] [10]. In recent years, many articles have been published; and research projects have been executed by employing Ring-LWE in different IoT-based applications and environments, such as healthcare networks, intelligent transportation systems, financial systems, insurance systems, and so on. Some works [75] utilized Ring-LWE for Fingerprint Authentication System, and [76] designed a low latency cryptosystem for Biomedical Images. Some hardware implementations for LWE and Ring-LWE have been proposed in [9] [10] [77] [78] [79].

A new variant of the ideal-lattice based encryption scheme was introduced in [11] that has a binary distribution and is called Ring-BinLWE. The security analysis and hardness of Ring-BinLWE were published by [11] and [80]. The implementation results showed that this optimized scheme is very suitable for resource-restricted devices. In [28], the low area and high-performance decryption phase of Ring-BinLWE was implemented on FPGA. An optimized variant of Ring-BinLWE for hardware implementation has been introduced in [1]. They designed two different architectures: high-speed and lightweight on FPGA and ASIC platforms for three phases of the scheme.

The main goal of this chapter is to design and implement a post-quantum crypto accelerator for IoT resource-constrained devices based on Ring-BinLWE. The design contains three phases: encryption, decryption, and key generation. The polynomial multiplication is the main operation in Ring-LWE and Ring-BinLWE. By employing the optimized version [1] of Ring-BinLWE, the modular reduction can be easily implemented in hardware platforms.

The main contributions of this chapter are summarized as follows:

- A novel multiplication technique is proposed for the design. The proposed multiplication is column-based that in each cycle, all coefficients of multiplier and multiplicand are involved in the multiplication.
- In the previous works [28] [1] [29], for every coefficient of each row, the modular reduction and anti-circular rotation should be executed. In the proposed multiplication, however, the modular reduction and anti-circular rotation are executed one time in each cycle of multiplication. Compared to the state-of-the-art design [1], this method reduces one 2-to-1 Mux and one NOT gate and causes the modular reduction and anti-circular rotation to be handled efficiently.
- The multiplication is executed by shift registers, which has a high impact on reducing the area and power. ASIC implementation of the proposed design has 57.8% and 48.42% improvement in terms of area and power, respectively, over the state-of-the-art design [1].

The proposed design and method have been implemented on ASIC-Tech $65nm$ and FPGA. According to the NIST lightweight cryptography [58], the ASIC implementation of the design is suitable for cryptosystems in IoT resource-restricted devices. The ASIC and FPGA frequency and area results appear to be better than previous works.

5.2 The proposed design

In this section, the proposed architecture is discussed. The main goal of this chapter is to design a cryptosystem for resource-constrained IoT devices so that the required security level is 80-bit [28] [81]. The parameters of Ring-BinLWE and Ring-LWE determine the security level of the design. By selecting the parameters of Ring-BinLWE to $n = 256$ and $q = 256$, the cryptosystem provide a security level of 84-bit (according to [80]), 88-bit (according to [82]) of conventional computers, and 73-bit [80] of quantum computers.

These parameters are smaller than Ring-LWE settings but are still suitable for lightweight applications [28] [1] [29]. This section is divided into two parts: in the first part, the method of In-place anti-circular rotation and modular reduction for column-based multiplication is introduced; later, the lightweight hardware architecture for small devices will be explained in detail.

5.2.1 In-place modular Reduction and anti-circular Rotation Column-based Multiplication

As previously mentioned, Ring-BinLWE does not use time and area consuming NTT-based polynomial multiplication. Instead, the polynomial multiplication in Ring-BinLWE is a sequence of additions that can be executed by *shift* and *add* operations. The multiplication is the main operation that is common in key generation, encryption, and decryption. One of the main characteristics of Ring-BinLWE using $f(x) = x^n + 1$ is an anti-circular rotation that should be implemented in a multiplication operation. The straightforward method of multiplication for Ring-BinLWE is the row of multiplication followed by anti-circular rotation and modular reduction, as shown in Figure 5.1(a). This method is explained as follow:

1. Each single coefficient of multiplier (a_i) is multiplied by all coefficients of multiplicand (b_i) to generate partial products.
2. The anti-circular rotation occurs in each row for partial products (the yellow coefficients Figure 5.1(a)).
3. Partial products are added into the intermediate sum (*IS*), followed by the modular reduction.
4. The final result of the polynomial multiplication is computed by accumulating the intermediate sum, followed by modular reduction.

In this work, a novel method of computing the polynomial multiplication is presented. In the proposed technique, the coefficient multiplication is performed in a column-wise

Figure 5.1: Two methods of doing multiplication for Ring-BinLWE (a) The conventional row-based multiplication for Ring-BinLWE. (b) The proposed column-based of multiplication for Ring-BinLWE. (1) The multiplicand (b_i) should be shifted to left. (2) Multiplier (a_i) should be started from the last coefficient (a_{n-1}). The anti-circular rotation occurs at yellow coefficients. IS stands for intermediate sum.

manner, which is followed by in-place modular reduction and anti-circular rotation; and thus, the proposed is termed *In-place Rot-Col-Mul*. In each cycle of the *In-place Rot-Col-Mul*, all multiplier coefficients (a_i) are utilized simultaneously to produce partial products. In each column, the corresponding multiplier is multiplied by multiplicand (b_i). The main difference between this method of multiplication and the conventional approach is that: (1) in each cycle, all of the multiplicands and multipliers are getting involved to calculate the final result; (2) the multiplication is column-based; (3) the multiplication starts from the last coefficient of the multiplier (a_{n-1}).

The main advantage of *In-place Rot-Col-Mul* over the conventional technique is the use of only one anti-circular rotation in each cycle. Since the final result is added to all of the coefficients in each column, this method is useful for lightweight implementation since it does not require registers for storing these additional partial products. It is worth

mentioning that as the range of q is determined by $(-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor - 1)$, the range of each coefficient matches the 2's-complement since the modular reduction is handled by normal overflow and underflow [1]. This method is illustrated in Figure 5.1(b). In this figure, the anti-circular rotation occurs in the yellow block of intermediate sum in each column. As it is obvious from Figure 5.1(b), in each column the position of every a_i is fixed and starts from a_{n-1} to a_0, and b_i should be shifted in each column. Also, by increasing the number of columns, the position of doing the anti-circular rotation is changed by one unit. Each iteration of the *In-place Rot-Col-Mul* involves the following steps:

1. Apply a circular left-shift to the multiplicand register (b_i). This part is shown is Figure 5.1(b) (1).

2. The multiplication starts from the last coefficient of multiplier (a_{n-1}). To better understand of doing multiplication, we depicted a_i from the last coefficient (a_{n-1}) to the first coefficient (a_0) in Figure 5.1(b) (2). In each cycle, the multiplier (a_i) and the multiplicand (b_i) should be shifted.

3. As all of the coefficients are in 2's-complement, the modular reduction is not required. The anti-circular rotation occurs one time in each column by changing the sign of the intermediate sum at the yellow coefficient in each column.

4. After doing the anti-circular rotation, the multiplication continues with the rest of the coefficients.

5. The position of doing the anti-circular rotation will be changed by one unit.

6. It will be repeated for the next columns.

To better understand the way of doing multiplication, the hardware design of the multiplication is shown in Figure 5.2 that the coefficients (a_i and b_i) are related to the first column. This process is the same as the other columns of the multiplication. The differences are that the position of doing anti-circular rotation will be transferred (based on the control

signals, the yellow box should move to the right in Figure 5.2) after finishing each column, by setting *EN* to '1' or '0'. At the end of each column, a circular left-shift should be applied to the multiplicand register (b_i).

As previously explained, the multiplication starts from the last coefficient of the multiplier (a_{n-1}); and the multiplicand (b_i) should have one circular left-shift (*Mul.0* in Figure 5.2). As the multiplicand is a binary vector, to do the multiplication, each bit of multiplicand should be extended to k-bit, where k is equal to log_2^q. Then the multiplication will be done for all of the coefficients until the position of anti-circular rotation (the yellow box in Figure 5.2). The anti-circular rotation occurs by one time activating *EN*=1. After that, the multiplication continues to finish all of the other coefficients for each column. For the next columns, the position of activating control signal *EN*=1 is changed by one unit to the right. The intermediate sums are stored in *REG*. As it is explained in Section 5.2.2, the position of anti-circular rotation is controlled by counters.

5.2.2 The Proposed Lightweight Design

In this section, the proposed lightweight architecture will be discussed. While the functionality of some IoT end-devices is limited to passing data forward in which they only need to encrypt or decrypt the data, the proposed architecture contains all three main phases of the Ring-BinLWE (key generation, encryption, and decryption). The proposed architecture is shown in Figure 5.3.

The maximum data flow of the design is k-bit (k is equal to 8). *REG_0* is a k-bit shift register for storing the plain-text (which is m in equations (2.4) and (2.5)). The *M_Code* is designed for coding the plaintext (according to equation (2.4) to calculate $\overline{m}$), which has 1-bit input and k-bit output. *REG_3, REG_4, REG_5*, and *REG_6* are k-bit. Based on the control signals and the phase of the design, $\overline{m}$ and C_2 are stored in *REG_4*, r_1 is stored in *REG_3* to do the *Add* operation of the algorithm. The partial products and intermediate sums are stored in *REG_5*. The multiplicand and multiplier are stored in the shift registers *REG_1* and *REG_2*, respectively (according to the phase of the design, a, p, and C_1 are

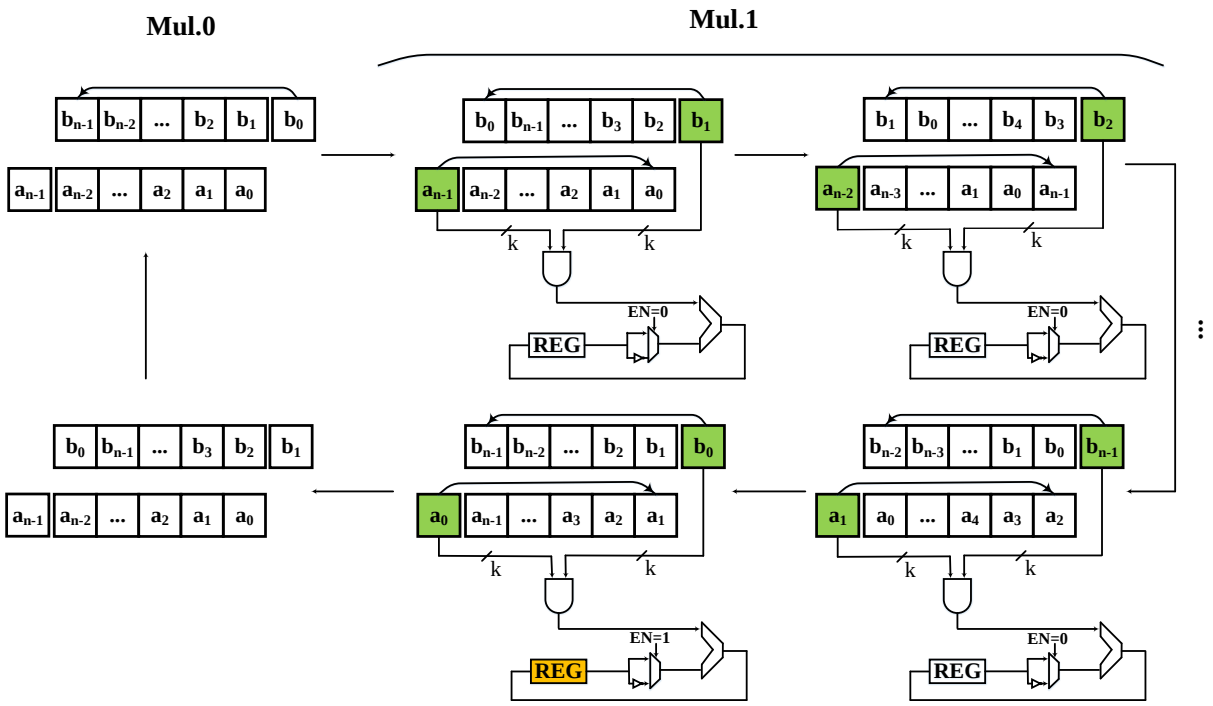


Figure 5.2: The hardware design of the proposed multiplication with the related coefficients of the first column. Note: as b_i is a binary vector, to do the multiplication, each bit of b_i should be extended to k-bit.

stored in *REG_1* and r_2 and e_1 are stored in *REG_2*). *REG_out* contains one shift register, and a k-bit register for storing the output result. The contents of registers are available in Table 5.1.

It should be mentioned that there are several ways to generate the required random numbers. Most of the similar works employed AES counter mode for generating random numbers, such as [11] [29] [83] [84]. In the proposed design, the same method as previous works by employing the AES design of [85] is applied. Based on the phase of the design, the random numbers are stored in *REG_2* and *REG_3*.

The purpose of Control signals Rot_CTL and EN is to change the function to the *In_place Rot-Col-Mul* and subtract when needed. The multiplication operation contains n rows and n columns. The design includes two counters for counting the number of row

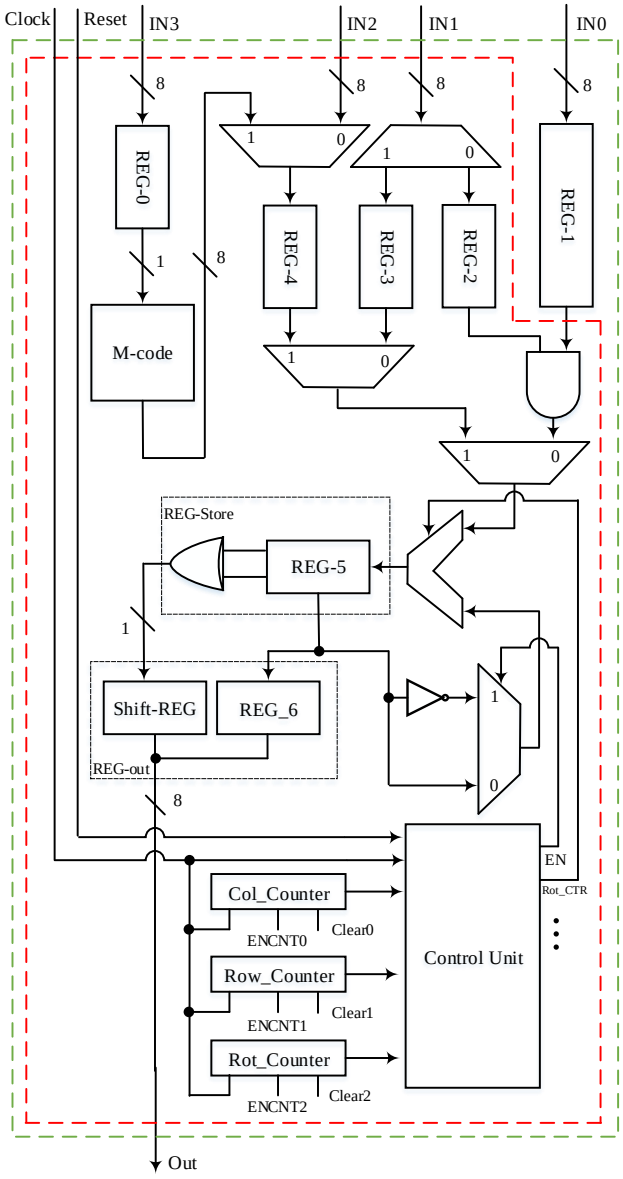


Figure 5.3: The proposed cryptosystem design. The green line contains *REG_1* in the design. The red line is the modified architecture to make a comparison with [1] and does not contain *REG_1*.

Table 5.1: The list of abbreviations, parameters, and the contents of registers

Abbreviation	**Description**
In-place Rot-Col-Mul	In-place modular reduction and anti-circular rotation
REG	Register
Rot	Anti-circular rotation
EN	Enable
Pre	Preparation
Mul	Multiplication
Dec	Decision
Add	Addition
Fin	Finish
REG_0	Containing k-bit shift register for storing the plaintext
REG_1	Storing a, p, and C_1 based on the phase of the design
REG_2	Storing r_2 and e_1
REG_3	Storing r_1
REG_4	Storing $\overline{m}$ and C_2, based on the phase of the design
REG_5	Storing the intermediate results
REG_out	Containing one shift register, and a k-bit register (REG_6) for storing the output result

Pseudo-code 1: The functionality of counters for executing the multiplication

```
Set the counters to initial values: Rot_Counter: 'n − 2', Row_Counter: '0', and Col_Counter: '0'
Result: EN and Rot_CTR
for Col_Counter : '0' to 'n − 1' do
    EN = '0'
    Rot_CTR = '0'
    for Row_Counter : '0' to 'n − 1' do
        if Rot_Counter == Row_Counter then
            EN = '1'
            Rot_CTR ='1'
        end
        Return EN and Rot_CTR
    end
    Rot_Counter −−
end
```

(*Row_Counter*) and column (*Col_Counter*), and one down counter to determine the position of doing anti-circular rotation (*Rot_Counter*). The initial value of *Rot_Counter* is set to '$n-2$'. As it was explained before, the multiplication starts from the last coefficient of the multiplier. For the first column, the anti-circular rotation occurs between the row number '$n-2$' and '$n-1$'. When the *Row_Counter* is equal to *Rot_Counter*, the *Control_Unit* activates the related signals for applying the anti-circular rotation and reduces *Rot_Counter* by one for the next column, this process will be done until *Rot_Counter* will be equal to '0'. The functionality of counters is written in Pseudo-code 1.

The finite state machine of the architecture is shown in Figure 5.4. The multiplication is the common function among three phases: key generation, encryption, and decryption. According to the explained equations in section 2.1.4, key generation and decryption need one multiplication and one addition (one stage for getting the result); encryption requires two multiplications and three additions (two steps for getting the results). During the encryption and decryption, the message should be encoded and decoded, respectively. The calculation of encryption requires two stages: one for generating $C_1 = ae_1 + e_2$ and the other one for $C_2 = pe_1 + e_3 + \overline{m}$.

The finite state machine of the design has seven states. At a high level of abstraction,

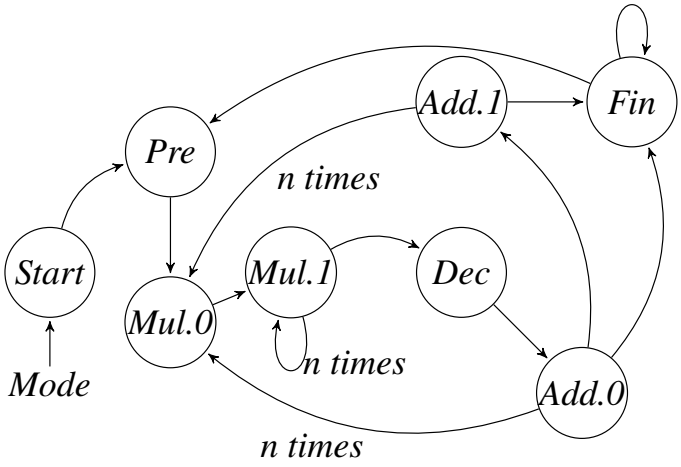


Figure 5.4: The finite state machine for the proposed Design

"*Start*" is the idle of the design and generates the signals for different modules based on the phase of the design, which are according to the input control signals (*Mode*). "*Pre*" stores the data in the registers (*REG_1* and *REG_2*), which requires n clock cycles for storing the data in *REG_1*. "*Mul.0*" is the pre-process for multiplication that prepares the registers for multiplication by performing one circular left shift of *REG_1*. "*Mul.1*" executes the multiplication and anti-circular rotation in each column which is executed $n = 256$ times. The position of the anti-circular rotation is decided based on the counters. The execution of "*Mul.0*" and "*Mul.1*" was shown in Figure 5.2. The list of abbreviations is written in Table 5.1.

After finishing each column, the design's state goes to "*Dec*" that prepares the design for addition based on the phase of the design and stores the required data for addition operation in the registers. If the phase of the design is decryption, key generation, or encryption (that is calculating C_1), the next state is "*Fin*" and the result is ready; if the phase of the design is encryption for generating C_2, the next state will be "*Pre*", and "*Add.1*" will be executed after "*Add.0*" to add $\overline{m}$ to the stored value in *REG_5*. This process will be executed n times. The final result of the decryption phase is a polynomial in R_q and must be decoded to a binary vector. The decode function is written in equation (2.5); and the method from [28] and [1] is borrowed to implement the decode function, which is implemented by XORing

the most significant two bits of each coefficient of the result.

According to the above description, the cryptosystem executes in constant clock cycles, where decryption and key generation require $(n+1)+n\times(n+3)$ clock cycles, and encryption requires $(2n+1)+n\times(2n+7)$ clock cycles. The timing diagram based on activating the registers and modular reduction and anti-circular rotation for the decryption phase is shown in Figure 5.5.

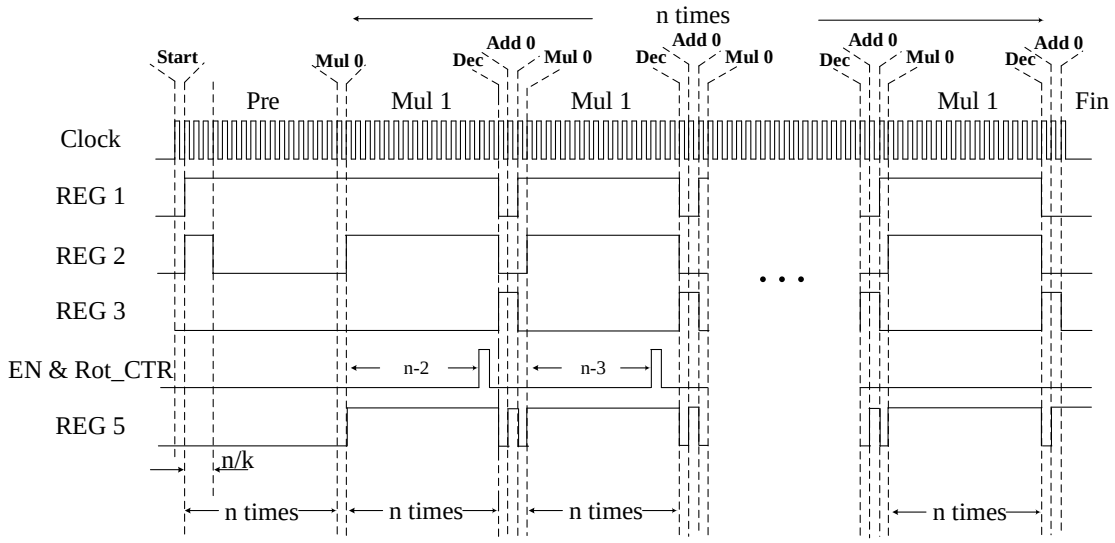


Figure 5.5: The timing diagram for activating registers and modular reduction and anti-circular rotation of decryption phase period

5.2.3 Security Analysis

The security of a cryptosystem contains two parts: the algorithm and the implementation. In the proposed architecture, the operations and functions of Ring-BinLWE have not been altered. Thus, it is expected the design keeps the same level of confidentiality, hardness, and security, according to [11] [80]. There are some attacks and threats, such as side-channel, that depends on the implementation design. The main goal of side-channel attacks is exploiting the secret key by achieving the relationship between physical attributes of the design, such as power consumption and timing behavior. Because of three reasons,

the proposed design is secure against timing attacks; 1) there is no conditional branches and dependency between the inputs and cipher-text; 2) for decryption of each cipher-text, the proposed design executes in a constant number of clock cycles (also clock cycles for encryption and key generation are constant); and 3) the critical path delay of the proposed design is constant for all three phases [29] [48] [66].

In the row-based multiplication, in each cycle of multiplication one coefficient of the private key (multiplicand) gets involved on multiplication with the entire coefficients of multiplier (a_i). If $r_2(i)$ (private key) is equal to '0', the switching activity is reduced because addition and internal registers update will not occur by default, and it can be recognized since this is done through the entire multiplier (a_i). An SPA attack can thus extract the secret key with a few measurements by interpreting differences in power traces caused by these operations. However, in the proposed design, the power activity has been normalized as: I) the multiplication is column-based that in each cycle of multiplication all of the multiplicand (b_i) and multiplier (a_i) participate followed by in-palace reduction and anti-circular reduction. II) In each clock cycle, addition and register read and write are executed. Therefore, regardless of the value of $r_2(i)$, in each clock cycle, there is always an execution of an addition and a register read and write that provide the resistance against exploring the private key by SPA caused by inactivity. To increase the security against differential power analysis (DPA), it is possible to apply the DPA countermeasures, such as the hardware design used in [28] [86], or modify the design based on the method of [29] to the proposed design. However, as DPA countermeasure is a supplementary module that is added to the main architecture and occupies more area, it is not common to use resource-consuming DPA countermeasure for IoT constrained-resource devices [1].

5.3 Implementation Results, Simulation, and Comparison

In this section, the implementation results of the proposed method and architecture are reported. The proposed architecture has been implemented on both ASIC and FPGA platforms with Very-high-speed-integrated-circuit Hardware Description Language (VHDL).

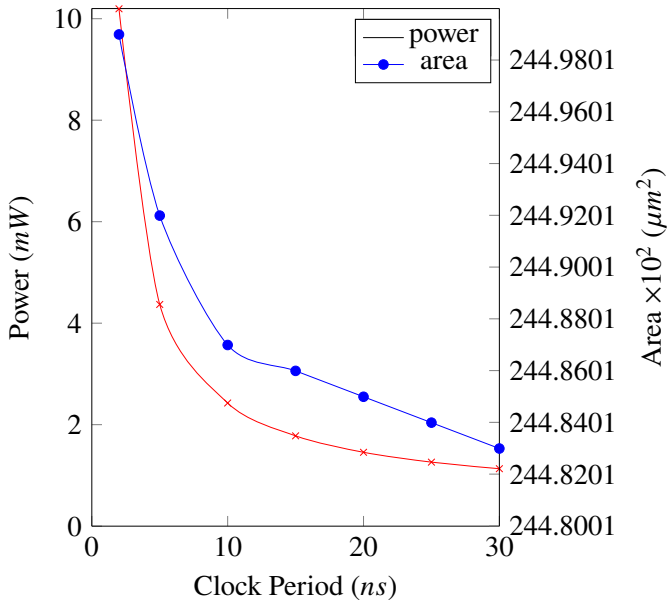


Figure 5.6: Area-delay curve and Power-delay curve of the proposed design

To verify the proposed architecture and method, different test vectors were generated using Python and used to simulate the design in ISim 14.7.

5.3.1 ASIC implementation results and comparison

The proposed design is synthesized in Synopsys Design Compiler using TSMC-65nm technology with typical case parameters 1.00V and 25°C. The design's timing, area, and power metrics are generated. To find the trade-off between delay and area, and delay and power, the proposed design is synthesized under different timing constraints (5, 10, 15, 20, 25, and 30ns) with a 10% input/output delay margin. The worst delay of the design is 2ns that the total power consumption (the addition of static power and dynamic power) and area of the design are 10.1952mW and 24499.08μm^2, respectively. The area-delay curve and power-delay curve are drawn in Figure 5.6. The minimum power and area occur at 30ns that are 1.1321mW and 24483.240μm^2. The detailed ASIC results and comparison with other similar works have been written in Table 5.2.

Among the other works, the lightweight design of [1] is a similar one to this work, which is implemented the same algorithm (with the same parameters) in the same technology.

The advantage of the design of [1] is that the reductions can be handled easily in hardware implementation by using 2's complement range. There are three main differences between the proposed design and the design of [1]. First, the anti-circular rotation was performed at every coefficient in [1]. In the proposed design, however, the modular reduction and anti-circular rotation occur for one time in each column. Second, for executing the anti-circular rotation, two *Mux*s, two *Not* gates, and one *Adder* are used, which made the *Control_Unit* be complicated, the delay was increased, and also their design occupied more area. In comparison to [1], by employing the *In-place Rot-Col-Mul* only one *Mux*, one *Not* gate, and one *Adder* are used for handling the anti-circular rotation. And third, in their design, they considered multiplicand (b_i) and addition vector as input (without storing in any register) of their design into two big multiplexers (n to 1 *Mux*). These two *Mux*s imposed an unbreakable delay and occupied more area. To make a fair comparison with [1], some changes have been made in the proposed design. These changes are: the modified design does not include *REG_1*, and the value of multiplicand (b_i) is considered as the input of the design that feeds to the design in each clock cycle (similar to [1]); some control signals have been changed according to the modified design. In Figure 5.3, the red line is the modified version of the proposed design to compare with [1] (the control signals are not drawn).

The finite state machine of the modified cryptosystem is the same as the main design. The main difference is on '*Pre*' that is run for $\frac{n}{k}$ clock cycles for storing the value of multiplier (a_i) in '*REG_2*'. In the modified design, the number of clock cycles for executing decryption and key generation is $n\times(n+3)+(\frac{n}{k})+1$, and for encryption is $n\times(2\times n+7)+2\times(\frac{n}{k})+1$. As the maximum frequency of [1] is 33.3 MHz, the results in the timing constraint $30ns$ were generated. In this timing constraint, the proposed design occupied an area of $4637.51\mu m^2$ and average power consumption of $0.196mW$. Compared to the [1], the proposed design has improved the area by 57.8% and power by 48.42%. As the worst delay of the design is $2ns$, the execution time for encryption and decryption are $265.986\mu s$ and $132.674\mu s$, respectively, which is very faster than the state-of-the-art design [1].

5.3.2 FPGA implementation results and comparison

Although most of the resource-constrained devices are implemented on ASIC platforms, the proposed design is also implemented on FPGA platforms. The target FPGAs are Xilinx Virtex-7 and Spartan-6, which are the most used FPGA in previous articles, by using ISE 14.7 and ISim 14.7 for synthesis, simulation, and post-placement timing result. The hardware design merits of timing, resources on FPGA of this work and other similar implementations are written in Table 5.3. For a fair comparison of our proposed Ring-BinLWE design, we should have selected other Ring-BinLWE; however, we also considered other Ring-LWE. Since the different designs used different FPGA resources, we normalized the occupied area by calculating the Equivalent Number of Slices (*ENS*), based on [87]. Furthermore, we employed *Area* $\times$ *Time* metric (*AT*) as a measurement for comparing the effect of execution time over the occupied area, which is a common and widely used measurement in previous work [1].

The FPGA implementation results of the proposed design and some other lightweight cryptosystems are reported in Table 5.3. The throughput of the proposed design is $26.12kbps$ and $52.64kbps$ for encryption and decryption on Virtex-7, respectively. Also, the FPGA efficiency (FPGA-Eff), which is throughput per slice, is $0.158kbps/slice$ and $0.319kbps/slice$ for encryption and decryption, respectively.

It is worth mentioning that the implementation of all of the Ring-BinLWE and some of Ring-LWE works contain the main architecture, and do not contain the occupied area consumption for uniform polynomial a, random number generation, or Gaussian sampler. The work [28] implemented the decryption phase of the Ring-BinLWE scheme on Xilinx Spartan-6 with the same parameters as the proposed design ($n = 256$ and $q = 256$), and with different range for q that reduction is mandatory after each operation. Also, for storing the coefficients, they used two BRAMs in their design. As doing the multiplication requires $n \times n$ clock cycles, the number of clock cycles is almost equal between different designs. In comparison to [28], the proposed design improves the execution time and *AT* measurement by 50% and 45.3% on the same platform, respectively.

Table 5.2: Results and Comparison for cryptosystem implementation on TSMC-65nm

Reference	Frequency	Area		#Clock Cycles		Time (μs)		Power	Energy(nj)	
	(MHz)	μm^2	GE	Enc.	Dec.	Enc.	Dec.	(mW)	Enc.	Dec.
This work	500[1]	24499.08	17k	133377	66561	266.754	133.122	10.1952	2719.61	1357.20
	33.33[2]	24483.24	17k	133377	66561	4001.31	1996.83	1.1321	4529.88	2260.61
	33.33[3]	4637.51	3.2k	132929	66337	3987.87	1990.11	0.196	781.622	390.06
Lightweight of [1]	33.33	11k	7.9k	131840	65792	3.8k	1.9k	0.38	1.4k	722
High-Speed of [1]	10	46k	32k	515	257	51	26	1.30	66	33

[1] The implementation results of the complete design at the maximum frequency (the green line design in Figure 5.3)

[2] The implementation results of the complete design at 30ns (the green line design in Figure 5.3)

[3] The modified architecture (the red line design in Figure 5.3) of the proposed design in the timing constrain 30ns in order to compare with [1]

Table 5.3: Results and comparison for cryptosystem implementation on FPGA

Reference	Scheme	Bit Sec. Qtm/Clsc[1]	Device	Type	Freq. (MHz)	LUT/FF/Slice	DSP /BRAM	#CC[2] Enc./Dec.	Time (μs) Enc./Dec.	ENS[3]	AT[4] Enc./Dec.
This work	R-BLWE	73/84	Virtex-7	Enc./Dec.	434.32	380/640/165	0	133k/66k	307.94/153.67	165	50.81k/25.35k
			Spartan-6	Enc./Dec.	279.64	444/642/146	0		477.98/238.93	146	69.78k/34.88k
[1]	R-BLWE	73/84	Virtex-7	Enc./Dec.	540/560	2k/2k/652	0	512/256	0.95/0.46	652	619.4/299.92
[28]	R-BLWE	73/84	Spartan-6	Dec.	-/135	57/30/19	0/2	-/65.79k	-/487.4	131	-/63.84k
[10]	R-LWE	80/106	Virtex-6	Enc./Dec.	313	1349/860/-	1/2(18k)	6.3k/2.8k	20.1/9.1	-	-
[88]	R-LWE	80/106	Spartan-6	Enc.	80	1307/889/406	0/1(18k),3(8k)	360.5k	4500	690.2	3105.9k
				Dec.	80	1307/889/406	0/1(8k)	72k	900	462	415.8k
[87]	R-LWE	80/106	Kintex-7	Enc.	304.69	898/815/303	1/3(8k)	69.6k	229	573.4	131.3k
				Dec.	303.40	635/190/194	1/1(8k)	34.4k	114	352.4	40.17k
[89]	R-LWE	-	Kintex-7	Enc./Dec.	275	1381/1179/479	2/2(8k)	35.45k/17.73k	129/64	795.8	102.65k/50.93k

[1] Qtm and Clsc stand for quantum and classic bit security, respectively. The bit security analyses are based on [11] and [1].

[2] #CC stands for the number of clock cycles.

[3] *ENS* stands for Equivalent Number of Slices. Based on [87], One DSP equivalent to 102.4 Slices, one 8k BRAM equivalent to 56 Slices, and one 18k BRAM equivalent to 116.2.

[4] AT = number of Slices $\times$ Time in microsecond ($Slices \times \mu sec$).

6. Fault Resilient Implementation of Binary Ring-LWE

Due to increasing the number of connected devices to IoT network, providing end-to-end security is essential. Lattice-based cryptography (LBC) is a promising method for IoT by providing the reasonable security against classic and quantum attacks. Binary Ring-LWE is a type of LBC that is suitable for IoT devices. However, a reliable crypto-system should also be secure against different side-channel attacks, such as power analysis or fault injection ones. In this chapter, a fault resilient hardware implementation for an optimized hardware design of Ring Binary LWE for resource-constraint IoT devices is presented. The design was implemented on the FPGA platform. This chapter is organized as follows. Section 6.1 provides a brief introduction to motivation and fault injection attack. The proposed design, method, and implementation of CCA2-Secure Ring-Bin LWE are described in Section 6.2. The implementation results and comparison are explained in Sections 6.3.

6.1 Introduction

As the Internet of Things (IoT) has been growing and the number of connected devices has been exponentially increased, providing end-to-end security becomes vital for a large network. In general, cryptography algorithms are divided into public key cryptography (PKC) and private key cryptography. Compared to private key, public key cryptography

The content of this chapter is originally accepted in: K. Shahbazi, and Seok-Bum Ko "Lightweight and CCA2-Secure Hardware Implementation of Binary Ring-LWE." 2022 IEEE International Symposium on Circuits and Systems (ISCAS). The manuscript has been reformatted for inclusion in this thesis.

provides more security for a large network, such as IoT. The other advantage of using PKC is that key management can be effectively done in a large network. Most of the current PKC, such as ECC and RSA, can be solved very efficiently on a quantum computer by using Shor's algorithm [5]. As a result, they are vulnerable and insecure when quantum computers increase in number; and it is essential to design reliable cryptosystems.

Lattice-based cryptography (LBC) is one of the quantum-resistant PKC, which is acceptable by NIST [7]. Until now there is no known quantum algorithms that can solve the lattice problems [73] that makes LBC a great candidate for post-quantum cryptosystems. Ring Learning With Errors (Ring-LWE) [8] is the efficient and practical type of LBC in hardware and requires a smaller key size. Modular multiplication by number theory transform (NTT) and discrete Gaussian sampler to generate errors are two main units in Ring-LWE. These two units occupy more area and have more latency on hardware designs. Binary Ring-LWE (Ring-Bin LWE) is a new variant of the LBC and was introduced in [11]. The security and hardness of Ring-Bin LWE were analyzed by [11] and [80]. The main features of Ring-Bin LWE are using a binary distribution, instead of Gaussian distribution; and the modular multiplication is based on shift-and-add, instead of NTT multiplication. This causes implementation of Ring-Bin LWE be a suitable cryptosystem for resource-restricted devices.

In recent years, many research projects have been done and published by employing Ring-LWE and Ring-Bin LWE in different IoT-based applications and environments. Some hardware implementations for LWE and Ring-LWE have been proposed in [9] [77] [78] [79] [75] [76]. An optimized variant of Ring-Bin LWE for hardware implementation has been introduced in [1]. [90] presented an optimized hardware implementation of Ring-Bin LWE on ASIC and FPGA for end-node IoT devices by introducing a column-based multiplication. In [28], the low area and high-performance decryption phase of Ring-Bin LWE was implemented on FPGA.

In fault injection attacks, the data can be manipulated by injecting an error into a memory or a signal to obtain the faulty execution of the system and try to extract message or private

key by evaluating the results. Skipping, zeroing, and randomization are different fault attacks that an adversary can execute over a cryptosystem. In randomization, some part of memory is set to random values. For Ring-Bin LWE, randomization fault does not have any impact over key generation and encryption phases. Skipping is avoiding to run certain operations in the algorithm, such as addition and multiplication. Zeroing contains setting some parts or the entire value of a coefficient to zero. Skipping and zeroing attacks have a high impact on Ring-Bin LWE.

Beside the security of cryptography algorithms, the hardware implementation of a crypto-system should be evaluated and be secure against different side channel attacks. Because of the distributed nature of the IoT network, the crypto-systems should be secured and analyzed against fault injection [91] [30]. The main goal of the fault injection is to extract the secret key and plain-text by injecting errors into signals. Among the fault attacks, Chosen Ciphertext Attacks (CCA and CCA2) are the main threat on Ring-Bin LWE [91] [30] [29]. It is worth mentioning that most of the previous implementations are vulnerable against fault attacks due to the fact that no fault attack countermeasures had been applied in their implementations. Very few works were done on fault detection of post-quantum cryptography (PQC). Previous works [30] [92] [29], added the fault-resilient architecture to Ring-LWE and Ring-Bin LWE designs. [29] proposed the fault resilient software implementation of the Ring-Bin LWE design of [1].

[90] introduced a novel method of multiplication and implemented Ring-Bin LWE on both FPGA and ASIC for IoT resource-constraint devices. In this paper, we design and apply the fault attack countermeasure to the hardware implementation of the optimized Ring-Bin LWE architecture of [90]. Compared to the similar hardware implementation of Ring-Bin LWE, [90] occupied less area and is more suitable for resource-constraint devices.

The main contributions of this work are as follows:

- Fault resiliency is evaluated for three phases of Ring-Bin LWE, key generation, encryption, and decryption. Then a fault resiliency architecture is introduced for

Ring-Bin LWE.

- The fault resilient of Ring-Bin LWE is implemented on FPGA Virtex-7. The proposed fault resilient implementation of Ring-Bin LWE occupies only 1% of total available slices and executes in an acceptable number of clock cycles, based on IoT criteria.

6.2 The Proposed Design

In this section, the detailed information related to the implementation of the fault resilient Ring-Bin LWE is discussed. The parameter sets of the proposed design are $n = 256$ and $q = 256$ that provide the security level of 73/84 bits for quantum/classical computers. The selected parameter sets and the security level are suitable for lightweight applications and resource-constraint devices. The CCA2-Secure is applied to the architecture of [90], which is an optimized implementation of Ring-Bin LWE by using a column-based novel multiplication method. Since some of fault attacks on decryption phase to gain secret keys and the plain-text, it is important to provide resistance against fault attacks on decryption phase by employing CCA2-secure implementation [91] [30] [29] [93]. Based on the Ring-Bin LWE algorithm, the message is decoded based on:

$$\begin{aligned} m &= Decode(C_1 r_2 + C_2) \rightarrow \\ m &= Decode(r_2(ae_1 + e_2) + e_1(r_1 - ar_2) + e_3 + \bar{m}) \rightarrow \\ m &= Decode(r_2 ae_1 + r_2 e_2 + e_1 r_1 - e_1 ar_2 + e_3 + \bar{m}) \rightarrow \\ m &= Decode(r_2 e_2 + e_1 r_1 + e_3 + \bar{m}) \end{aligned} \tag{6.1}$$

According to equation (6.1) and [11], the noise polynomial is equal to $N = e_1 r_1 + e_2 r_2 + e_3$ and $E(e_j r_j) = (-n + 2i + 2)/4$, which $j \in \{1, 2\}$ and $i \in \{0, ..., n-1\}$. As it was mentioned before, the main goal of fault attacks is to recover secret key or plain-text by manipulating the input data. Also, fault attacks on Ring-Bin LWE can be done by skipping and zeroing on key generation, encryption, and decryption phases. Key is generated by $p = r_1 - a.r_2$; skipping and zeroing fault can be done by setting a, r_1, or r_2 to zero which results producing weak public key, recovering the secret key and plain-text. When a or r_2 set to zero, the weak

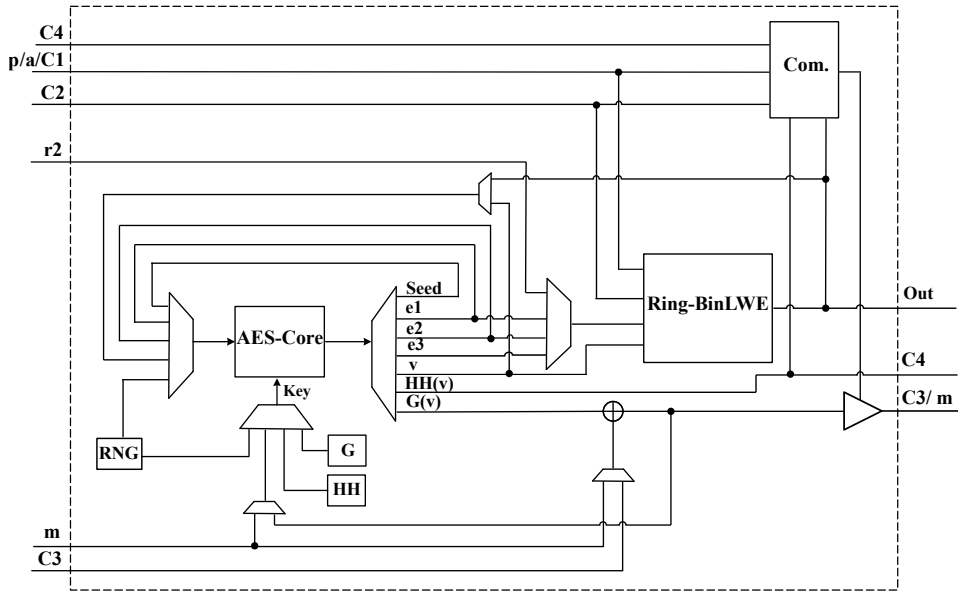


Figure 6.1: The hardware design of the proposed fault resilient Ring-Bin LWE

public key is $p = r_1$, which results two weak cipher-texts $C_1 = e_2$ and $C_2 = r_1 e_1 + e_3 + \bar{m}$. As $r_1 e_1 + e_3 < N$, the message could be recovered. When r_1 is equal to zero, $p = -ar_2$ and secret key can be recovered by evaluating the result of public key from a.

The adversary can recover the plain-text from the faulty data by zeroing e_1 or skipping the multiplication. By doing that the results of cipher-texts are $C_1 = e_2$ and $C_2 = e_3 + \bar{m}$. Thus, the message is decoded by $e_2 r_2 + e_3 + \bar{m}$. There is not a in the message's equation and since r_2 is a binary vector, for each random selected of r_2, $(e_2 r_2 + e_3) < N$ and the message could be recovered. The cipher-text is decrypted by $\bar{m} = C_2 + C_1.r_2$. In the decryption phase, the secret key can be extracted by calculating the differentiating between correct and faulty decryption results by setting C_2 to zero. The proposed design of fault resilient is presented in Figure 6.1.

[30] pointed that three random oracles are necessary for the transformation of quantum security to provide classic and quantum attack resistance. Random number generators (RNGs) and oracles are necessary to resist fault attacks. These random oracles are $H : \{0,1\}^{2n} \rightarrow \{0,1\}^n$, $G : \{0,1\}^n \rightarrow \{0,1\}^n$, and $HH : \{0,1\}^n \rightarrow \{0,1\}^n$. The Ring-Bin

LWE core in Figure 6.1 is borrowed from the the design of [90]. In Figure 6.1, G and HH provide the security check in which will be used during the decryption phase. *RNG* of the design is implemented based on [11] [29] [83] in which runs the AES in counter mode and encrypts successive values of an incrementing counter.

In the architecture, we used the design of AES [85]. The AES design of [85] is an optimized lightweight AES implementation with 8-bit datapath for IoT resource-constraint devices and requires 527 clock cycles. The datapath of the proposed design is 8-bit and is suitable with the modulo 256 of the algorithm. Random oracles are implemented with different vectors. Similar to [29], the inputs are used as encryption key and the encrypted outputs are used as the returned value of random oracles. The pseudo-code of the proposed CCA2-Secure cryptosystems is written in Pseudo-code 1.

In Pseudo-code 1, HH and G are random oracles that HH and G are implemented as constant numbers (different for each random oracle) that are input key for AES. $H(a, b)$ is a function that a is the plain-text and b is the key for AES algorithm. The required errors are generated separately and fed to the Ring-Bin LWE. At first, the random vector v is generated by *RNG* and AES core; as *RNG* is executed over a counter mode AES, the value of v is unique for each execution. Then random $seed$ is produced from message m (as the key for AES core) and the random vector v (as the plain-text for AES core). Errors (e_1, e_2, and e_3) are generated by random $seed$ and oracle HH. To reduce the area, only one AES core is used that results a recursive structure. In Figure 6.1, *Mux-0* and *InvMux-1* are employed to return back data to the AES core for generating the remaining values.

To verify that the fault injection has not occurred (beside C_1 and C_2) C_3 and C_4 should be calculated by oracles in encryption phase, and will be used in decryption phase for verifying that fault has not been happened. As a result, fault injection attacks on cipher-texts can be detected during decryption phase.

The decryption phase should contain the followings: first, calculating the decryption result of Ring-Bin LWE to get the value of v, based on the inputs (C_1 and C_2); second,

Pseudo-code 1: The proposed CCA2-Secure Ring-Bin LWE

A: The proposed Ring-Bin LWE Encryption with CCA2-Secure

Data: r_2, a, m

Output: c_1, c_2, c_3, c_4

begin

$v \leftarrow RNG$

$seed \leftarrow H(v, m)$

$e_1 \leftarrow HH(seed)$

$e_2 = HH(e_1)$

$e_3 = HH(e_2)$

$c_1 = a.e_1 + e_2, c_2 = p.e_1 + e_3 + \overline{v}$

$c_3 = G(v) \bigoplus m, c_4 = HH(v)$

end

B: The proposed Ring-Bin LWE Decryption with CCA2-Secure

Data: $c_1, c_2, c_3, c_4, a, r_2$

Output: m

begin

$\overline{v} = c_1.r_2 + c_2, v = decode(\overline{v})$

$m = G(v) \bigoplus c_3$

$seed \leftarrow H(v, m)$

$e_1 \leftarrow HH(seed)$

$e_2 = HH(e_1)$

$e_3 = HH(e_2)$

$c_1^{'} = a.e_1 + e_2, c_2^{'} = p.e_1 + e_3 + \overline{v}, c_4^{'} = HH(v)$

if $c_1^{'} == c_1$ and $c_2^{'} == c_2$ and $c_4^{'} == c_4$ **then**

return m

end

end

message is recovered by oracle and C_3; then errors are generated by the $seed$, which is generated from v and m; after that, the calculated results came back to the Ring-Bin LWE core to produce the encrypted results; and finally, the encrypted results and the input results are compared to make sure that the fault has not been happened. The comparison unit in Figure 6.1 is designed by XOR in which when the result is 1 fault has been injected. If C_3 and C_4 have been injected by fault, it is recognized during the decryption phase. It is worth mentioning that the proposed design is resistance against timing attacks since it is executed in constant number of clock cycles; the critical path delay of the proposed architecture is constant during the execution of different phases; there is no conditional branches and dependency between the inputs and cipher-text [66].

6.3 Implementation Results, Simulation, and Comparison

In this section, the implementation results of the CCA2-Secure and comparison with similar works are reported. The architecture has been implemented on FPGA platforms with Very-high-speed-integrated-circuit Hardware Description Language (VHDL). To verify the design and method, different test vectors were generated and used to simulate the design in ISim 14.7. The target FPGA is Xilinx Virtex-7 that is mostly used FPGA in previous articles. The synthesis, simulation, and post-placement timing result are obtained by ISE 14.7. The hardware design merits of timing, resources on FPGA of this work are written in Table 6.1. The achieved frequency for the design after the place and route for Virtex-7 is 210.5MHz, and the number of occupied slices is 423 out of 51000. It is obvious from the results that the proposed fault resilient Ring-Bin LWE consumes very low area on FPGA (only 1% on Virtext-7 that has 51000 Slices) can be used as a crypto accelerator on FPGA.

The fault resilience of our implementations requires $655\mu s$ and $969\mu s$ for encryption and decryption on Virtex-7, respectively. Adding fault resilient causes increasing more resources and latency due to having more cipher-texts (C_3 and C_4). Furthermore, the proposed design contains error generation parts that results occupation more area compared to the other works. It is worth mentioning that most of the previous Ring-Bin LWE, such as [90], reported the results for the main architecture of the algorithm that do not contain the occupied area consumption for error generation module.

The design of CCA2-Secure implementation of Ring-Bin LWE has 53% and 84% slower compared to [90] in encryption and decryption operations on Virtex-7, respectively. The main reason for increasing the latency of the design, compared to [90], is that the latency of AES module is dominant, which the maximum frequency is 147 MHz on Virtex-5 [85]. Compared to [90], the proposed design of CCA2-Secure requires more clock cycles for decyption phase since to validate the final results, the process of encryption and generating the errors should be executed again to make sure fault has not been injected to the signals. As the main goal is to occupy less area on FPGA, only one AES core has been used, which results executing on more clock cycles for generating errors and verifying the decryption.

Table 6.1: The FPGA implementation results and comparison

References		This Work	[92][1]	[29]	[90][2]
Platform		Virtex-7	Virtex-7	AVR/ARM	Virtex-7
Frequency (MHz)		210.5	-	-	434.32
LUT		1206	1234	-	380
FF		1241	792	-	640
Slices		423	-	-	165
Clock Cycles	Enc	$138k$	-	$2691k$	$133k$
	Dec	$204k$	-	$4037k$	$66k$
Time (μs)	Enc	655.60	21.67	80200	307.94
	Dec	969.12	-	120300	153.67
AT ($Slices \times \mu Sec$)	Enc	$277.31k$	-	-	$50.81k$
	Dec	$409.93k$	-	-	$25.35k$

[1] The result of this design is only for encryption phase

[2] This design does not contain fault resilient

It should be mentioned that the fault attacks are applicable in previous similar FPGA implementation as fault attack countermeasures were not applied in their designs. One of the similar work that applied fault resilient on Ring-Bin LWE is [29], which added fault resilient to [1] and was implemented in AVR and ARM. As the implemented platforms are not the same, making a fair comparison is difficult. However, the number of clock cycles of

the proposed CCA2-secure Ring-Bin LWE is much fewer than [29] (which is 2691×1000 and 4037×1000 for encryption and decryption, respectively); and required $80.2ms$ and $120.3ms$ for execution of encryption and decryption, respectively. The proposed design improved the execution time by 99% over [29] on encryption and decryption. Furthermore, as the implementation results of [90] is better than [1], the proposed design would occupy less area than [29] in the same platform.

[92] presented three different architectures for three phases of Ring-Bin LWE (key generation, encryption, and decryption), compared to our work which is all-in-one architecture. In Table 6.1, the results of encryption architecture of [92] are written. Their design contained the main structure of Ring-Bin LWE and did not have the error generation module. Thus, it required less number of clock cycles for execution. Although the proposed architecture contains three phases and error generation, [92] occupied more look-up tables (LUT) than our proposed one.

7. An Optimized Implementation of Modular Multiplication for Binary Ring-LWE

Providing end-to-end security is vital for most of the networks. By emerging quantum computers, it is necessary to design crypto-systems that are secure against quantum attacks. Binary Ring Learning With Error (Ring-Bin LWE) is a Lattice-based cryptography which is hard to solve by quantum computers. Also, this algorithm does not have costly operations in terms of area in which make Ring-Bin LWE a suitable algorithm for resource-constraint devices. This chapter presents a lightweight hardware implementation of Ring-Bin LWE. In the proposed design, a new multiplication method and design for Ring-Bin LWE is introduced which results in latency reduction by factor of two. The multiplication is based on the column-based multiplication and in each cycle two consecutive coefficients are processed. The architecture is designed based on the proposed multiplication and contains one specific register bank with two sub-bank registers. The design is implemented on the FPGA platform. The implementation results show an impressive improvement in execution time and Area-Time metrics over the previous similar works. This chapter is as follows: Section 7.1 provides an introduction to motivation and main idea of this work. The novel multiplication method is introduced in Section 7.2. The proposed architecture and implementation are described in Section 7.3. The implementation results and comparison are explained in Sections 7.4.

The content of this chapter is originally submitted in: K. Shahbazi, and Seok-Bum Ko "An Optimized Hardware Implementation of Modular Multiplication of Binary Ring LWE." IEEE Transactions on Very Large Scale Integration (VLSI) Systems. The manuscript has been reformatted for inclusion in this thesis.

7.1 Introduction

Cryptography is an essential part of every network to keep the security and confidentiality. Providing end-to-end security is vital for most networks, as there is not enough area for security part for most of the end-node devices. Furthermore, there is a lack of lightweight designs and implementations. In these years, designing lightweight hardware cryptography circuits is a hot growing topic. Among the different ways of cryptography, Lattice-based cryptography (LBC) is one of the most promising types of cryptography. LBC is one of the secure methods against quantum attacks due to its small implementation and strong security.

Learning with Errors (LWE) [94] is a public key cryptography based on LBC. LWE occupies more area and resources on hardware platforms. Ring-LWE [8] is a variant of LWE that uses the polynomials over the ring $R_q = Z_q/(x^n+1)$. Using $f(x) = (x^n+1)$ makes Ring-LWE practical on hardware platforms. Ring-LWE requires Gaussian distribution for generating errors and modular multiplication by Number Theoretic Transform (NTT). This algorithm is not suitable for resource-constraint devices since Gaussian distribution and NTT multiplication occupy more area and need more clock cycles. In 2016, a new variant of Ring-LWE was introduced by [11] which is called Binary Ring LWE (Ring-Bin LWE). In this algorithm, the errors are binary; thus, it does not require Gaussian distribution. The other advantage of Ring-Bin LWE is that multiplication does not require NTT multiplication as the errors are binary values. As a result, Ring-Bin LWE needs less key size and values and can be implemented on less area, makes this algorithm suitable for resource-constraint devices. In this work, we optimize the multiplication to reduce the number of clock cycles. The architecture is designed based on the optimized multiplication and implemented on FPGA platform. The contribution of the paper is summarized as:

- The location of each coefficient of polynomials in the multiplication in Ring-Bin LWE is evaluated, and an optimized column-based multiplication is introduced. The proposed multiplication method optimizes the column-based multiplication and rotation that requires $\frac{n}{2} \times n$ cycles to execute the multiplication. This optimization results

almost 50% improvement over the previous similar works.

- Compared to the other similar works that used *AND* gate for multiplication, a 4-1 *Mux* is used for multiplication. The values of two consecutive coefficients of matrix $\mathbb{A}$ and B are fed to the design.

- The proposed architecture is designed efficiently based on the proposed method. The register bank of the design for storing $\mathbb{A}$ contains two sub-register banks for better controlling the multiplication and rotation.

- The proposed method and architecture is implemented on FPGA platform. The proposed method and architecture improved the *Area-Time* metrics by 41% and 97% over [90] and [95], respectively.

7.2 Optimized Column-Based Multiplication

As it was explained in the previous section, the Ring-Bin LWE contains two main operations. Based on the phases of the algorithm, $W = AB \ mod \ f(x) + C$ is the main equation of Ring-Bin LWE that contains modular addition and polynomial multiplication. Where W, A, B, and C are polynomials and defined as $W = \sum_{i=0}^{n-1} w_i x^i$, $A = \sum_{i=0}^{n-1} a_i x^i$, $C = \sum_{i=0}^{n-1} c_i x^i$, and $f(x) = x^n + 1$ in which w_i, a_i, and c_i are log_2^q-bit integers in Z_q; and $B = \sum_{i=0}^{n-1} b_i x^i$ in which $b_i \in \{0, 1\}$.

The multiplication in Ring-LWE and Ring-Bin LWE contains a $n \times n$ matrix and $n \times 1$ matrix (that is the polynomial B). The $n \times n$ matrix (which is defined as $\mathbb{A}$) is generated by expanding the polynomial A (which could be one of the a, p, and C_1). The first row of matrix $\mathbb{A}$ is the coefficients of polynomial A, and the remaining rows are generated by multiplying the previous polynomial by x in module $x^n + 1$. As it was mentioned before, selecting $x^n + 1$ provides the anti-circular rotation since $x^n \equiv -1$. That is the cyclically shifted of the previous polynomial and changing the sign of the shifted coefficients. Based on that, each row and column of matrix $\mathbb{A}$ is connected to each other. Common method of multiplications requires $n \times n$ registers to store the intermediate results. To implement

the Ring-Bin LWE on resource-constraint devices, we propose and optimize the column-based multiplication in which requires only one register to store the intermediate results and execute in half number of clock cycles.

We define each element of matrix $\mathbb{A}$ as $\mathbb{A}^i_j$ that i and j are $0 \leqslant i < n$, $0 \leqslant j < n$ in which i is the number of the column and j is the number of row for $\mathbb{A}$. The first row ($j = 0$) of $\mathbb{A}$ is equal to A, and the remaining rows of generated from the first row. We have:

$$\begin{aligned} \mathbb{A}_0 &= A \\ &= a_0 x^0 + a_1 x^1 + ... + a_{n-1} x^{n-1} \end{aligned} \tag{7.1}$$

The row number 1 of matrix $\mathbb{A}$ is $\mathbb{A}_1 = \mathbb{A}_0 \times x \; mod \; (x^n + 1)$. Thus, the last coefficient is $a_{n-1} \times x^n$; since $x^n \equiv -1$, the sign of this coefficient should be changed and rotated. Thus, $\mathbb{A}_1$ is:

$$\begin{aligned} \mathbb{A}_1 &= (\mathbb{A}_0 \times x) mod (x^n + 1) \\ &= (a_0 x^1 + ... + a_{n-2} x^{n-1} + a_{n-1} x^n) mod (x^n + 1) \\ &\xrightarrow{x^n \equiv -1} \\ &= -a_{n-1} x^0 + a_0 x^1 + ... + a_{n-2} x^{n-1} \end{aligned} \tag{7.2}$$

The other rows of matrix $\mathbb{A}$ are calculated similar to equations (7.1) (7.2) and the aforementioned explanation by $\mathbb{A}_k = \mathbb{A}_{k-1} \times x \; mod \; (x^n + 1)$ and considering $x^n \equiv -1$. It is obvious that the number of rotated coefficients are equal to the number of row of $\mathbb{A}$ (starting from 0 to $n-1$). By doing the rotation at each row, the coefficients in each column of matrix $\mathbb{A}$ are unique. The coefficient in each column of $\mathbb{A}^i$ is the coefficient $\mathbb{A}_j$ with having the factor of x^i. Also, the number of rotated coefficients in each column is equal to $n - 1 - i$. The polynomial of column number i of $\mathbb{A}$ is calculated by:

$$\mathbb{A}^i = \sum_{j=0}^{n-1} ((\mathbb{A}_j x^i) x^j) \tag{7.3}$$

Based on the above equation, the value of columns are:

$$\begin{aligned}
\mathbb{A}^0 &= \sum_{j=0}^{n-1}((\mathbb{A}_j x^0)x^j) \\
&= a_0x^0 - a_{n-1}x^1 - ... - a_2x^{n-2} - a_1x^{n-1} \\
\mathbb{A}^1 &= a_1x^0 + a_0x^1 - a_{n-1}x^2 - ... - a_3x^{n-2} - a_2x^{n-1} \\
&\cdot \\
&\cdot \\
&\cdot \\
\mathbb{A}^{n-2} &= a_{n-2}x^0 + a_{n-3}x^1 + ... + a_0x^{n-2} - a_{n-1}x^{n-1} \\
\mathbb{A}^{n-1} &= a_{n-1}x^0 + a_{n-2}x^1 + ... + a_1x^{n-2} + a_0x^{n-1}
\end{aligned} \tag{7.4}$$

From equation (7.4), each column of matrix $\mathbb{A}^k$ is gained by $\mathbb{A}^k = \mathbb{A}^{k-1} \times x \ mod \ (x^n + 1)$. To better understand, the value of matrix $\mathbb{A}$ is written in Table 7.2. Executing the column-based multiplication, according to equation (7.4), requires at least $n \times n$ clock cycles. To efficiently execute the column-based multiplication on hardware platforms, we rewrite $\mathbb{A}^i$ as follows:

$$\begin{aligned}
\mathbb{A}^0 &= a_0x^0 - a_{n-1}x^1 - (a_{n-2}x^2 ... + a_2x^{n-2} + a_1x^{n-1}) \\
\mathbb{A}^1 &= a_1x^0 + a_0x^1 - (a_{n-1}x^2 + ... + a_3x^{n-2} + a_2x^{n-1}) \\
&\cdot \\
&\cdot \\
&\cdot \\
\mathbb{A}^{n-2} &= a_{n-2}x^0 + a_{n-3}x^1 + ... + a_0x^{n-2} - a_{n-1}x^{n-1} \\
\mathbb{A}^{n-1} &= a_{n-1}x^0 + a_{n-2}x^1 + ... + a_1x^{n-2} + a_0x^{n-1}
\end{aligned} \tag{7.5}$$

Equation (7.5) require $n \times \frac{n}{2}$ clock cycles to generate the entire of $\mathbb{A}$. To reduce the number of clock cycles, we rewrite the equation (7.5) by considering processing two coefficients at the same time. By defining $\sigma_{(k,k-1)}x^{\frac{l}{2}} = a_kx^l + a_{k-1}x^{l+1}$ and $\bar{\sigma}_{(k,k-1)}x^{\frac{l}{2}} = a_kx^l - a_{k-1}x^{l+1}$,

we rewrite equation (7.5):

$$\begin{aligned}
\mathbb{A}^0 &= \bar{\sigma}_{(0,n-1)}x^0 - (\sigma_{(n-2,n-3)}x^1 + ... + \sigma_{(2,1)}x^{\frac{n}{2}-1}) \\
\mathbb{A}^1 &= \sigma_{(1,0)}x^0 - (\sigma_{(n-1,n-2)}x^1 + ... + \sigma_{(3,2)}x^{\frac{n}{2}-1}) \\
&\vdots \\
\mathbb{A}^{n-2} &= \sigma_{(n-2,n-3)}x^0 + \sigma_{(n-4,n-5)}x^1 + ... + \bar{\sigma}_{(0,n-1)}x^{\frac{n}{2}-1} \\
\mathbb{A}^{n-1} &= \sigma_{(n-1,n-2)}x^0 + \sigma_{(n-3,n-4)}x^1 + ... + \sigma_{(1,0)}x^{\frac{n}{2}-1}
\end{aligned} \tag{7.6}$$

The multiplication and generating $\mathbb{A}$ based on equation (7.6) can be done in $n \times \frac{n}{2}$ clock cycles, which has an impressive optimization. Thus, in each clock cycle, one odd and one even coefficients can be processed. The value of the optimized matrix $\mathbb{A}$ is written in Table 7.3. It is obvious from Table 7.3 and equation (7.6) that $\bar{\sigma}$ exists in columns with even numbers. Also, after summation of all of the $\sigma_{(k,k-1)}$, the rotation occurs at the bold coefficients in Table 7.3. We borrowed the column-based multiplication method from [90] in which in each columns, all coefficients of b_j are participated and are unique. We rewrite $V = A \times B \ mod \ (x^n + 1)$ based on column-based multiplication as:

$$V_j = \left\{ \sum_{i=0}^{n-1} (\mathbb{A}_i^j \times B_i) \right\} mod(x^n + 1)x^j, 0 \leqslant j < n \tag{7.7}$$

All of the previous works used *AND* gate for multiplication of $A \times B$. The multiplication can be calculated by a chain of two consecutive coefficients of $\mathbb{A}$ and B as $v_k = a_m b_k x^k + a_{m-1} b_{k+1} x^{k+1}$. Since B includes binary numbers, the summation of multiplications of two coefficients is determined by β_k (which is two bits and includes $b_{2k} b_{2k+1}$). By considering σ, $\bar{\sigma}$, and β_k, the multiplication can be calculated by $v_k = \sigma_{(m,m-1)} \times \beta_k$ and $v_k = \bar{\sigma}_{(m,m-1)} \times \beta_k$. It is worth mentioning that $v_k = \bar{\sigma}_{(m,m-1)} \times \beta_k$ is needed only one time on even columns of matrix $\mathbb{A}$. The result of multiplication is written is Table 7.1. Based on Table 7.1 and aforementioned explanation, the multiplication can be executed by one 4-1 Mux. Compared to multiplication by *AND* gate, the proposed multiplication is run half number of clock cycles. When β_k is equal to '11', the result is σ or $\bar{\sigma}$.

Table 7.1: The multiplication result of two consecutive coefficients

β_k	v_k
00	0
01	a_{m-1}
10	a_m
11	$a_m \pm a_{m-1}$

Table 7.2: The content of Column-Based matrix $\mathbb{A}$ that has $n \times n$ cycles to execute the multiplication by b_k. The bold coefficients are rotated coefficients

j	$\mathbb{A}_j^{n-1}$	$\mathbb{A}_j^{n-2}$	$\mathbb{A}_j^{n-3}$	$\mathbb{A}_j^{n-4}$	...	$\mathbb{A}_j^3$	$\mathbb{A}_j^2$	$\mathbb{A}_j^1$	$\mathbb{A}_j^0$	$\times b_k$
0	a_{n-1}	a_{n-2}	a_{n-3}	a_{n-4}		a_3	a_2	a_1	a_0	b_0
1	a_{n-2}	a_{n-3}	a_{n-4}	a_{n-5}		a_2	a_1	a_0	$-a_{n-1}$	b_1
2	a_{n-3}	a_{n-4}	a_{n-5}	a_{n-6}		a_1	a_0	$-a_{n-1}$	$-a_{n-2}$	b_2
3	a_{n-4}	a_{n-5}	a_{n-6}	a_{n-7}	.	a_0	$-a_{n-1}$	$-a_{n-2}$	$-a_{n-3}$	b_3
...	...	...	...	...	.	**...**	**...**	**...**	...	...
$n-4$	a_3	a_2	a_1	a_0	.	$-a_7$	$-a_6$	$-a_5$	$-a_4$	b_{n-4}
$n-3$	a_2	a_1	a_0	$-a_{n-1}$		$-a_6$	$-a_5$	$-a_4$	$-a_3$	b_{n-3}
$n-2$	a_1	a_0	$-a_{n-1}$	$-a_{n-2}$		$-a_5$	$-a_4$	$-a_3$	$-a_2$	b_{n-2}
$n-1$	a_0	$-a_{n-1}$	$-a_{n-2}$	$-a_{n-3}$		$-a_4$	$-a_3$	$-a_2$	$-a_1$	b_{n-1}

The architecture of the proposed multiplication is shown in the Fig. 7.1. As it is obvious from Fig. 7.1, the proposed multiplication contains one 4-1 Mux, one 2-1 Mux, one *NOT* gate, and one *Adder*. The functionality of the multiplication part of the proposed design is executing the rotation and calculating $\bar{\sigma}$ and σ over the even columns by signal *Rot-1*.

7.3 The Proposed Design

In this section, the hardware architecture of the optimized multiplication is discussed. The proposed architecture is shown in Fig. 7.1. As log_2^q is equal to 8, the dataflow of

Table 7.3: The content of optimized Column-Based matrix $\mathbb{A}$ that requires $n \times \frac{n}{2}$ cycles to execute the multiplication by β_k

j	$\mathbb{A}_j^{n-1}$	$\mathbb{A}_j^{n-2}$	$\mathbb{A}_j^{n-3}$	...	$\mathbb{A}_j^{2}$	$\mathbb{A}_j^{1}$	$\mathbb{A}_j^{0}$	$\times\beta_k$
0	$\sigma_{(n-1,n-2)}$	$\sigma_{(n-2,n-3)}$	$\sigma_{(n-3,n-4)}$		$\sigma_{(2,1)}$	$\sigma_{(1,0)}$	$\bar{\sigma}_{(0,n-1)}$	β_0
1	$\sigma_{(n-3,n-4)}$	$\sigma_{(n-4,n-5)}$	$\sigma_{(n-5,n-6)}$		$\bar{\sigma}_{(0,n-1)}$	$\sigma_{(n-1,n-2)}$	$\sigma_{(n-2,n-3)}$	β_1
2	$\sigma_{(n-5,n-6)}$	$\sigma_{(n-6,n-7)}$	$\sigma_{(n-7,n-8)}$	.	$\sigma_{(n-2,n-3)}$	$\sigma_{(n-3,n-4)}$	$\sigma_{(n-4,n-5)}$	β_2
...	...	...	...	.	...	...	...	...
$\frac{n}{2}-3$	$\sigma_{(5,4)}$	$\sigma_{(4,3)}$	$\sigma_{(3,2)}$	.	$\sigma_{(8,7)}$	$\sigma_{(7,6)}$	$\sigma_{(6,5)}$	$\beta_{\frac{n}{2}-3}$
$\frac{n}{2}-2$	$\sigma_{(3,2)}$	$\sigma_{(2,1)}$	$\sigma_{(1,0)}$		$\sigma_{(6,5)}$	$\sigma_{(5,4)}$	$\sigma_{(4,3)}$	$\beta_{\frac{n}{2}-2}$
$\frac{n}{2}-1$	$\sigma_{(1,0)}$	$\bar{\sigma}_{(0,n-1)}$	$\sigma_{(n-1,n-2)}$		$\sigma_{(4,3)}$	$\sigma_{(3,2)}$	$\sigma_{(2,1)}$	$\beta_{\frac{n}{2}-1}$

the design is selected to 8-bit. Three phases of the algorithm can be executed by the design. Based on the phases of the algorithm (encryption, decryption, or key generation), the architecture executes $W = \mathbb{A} \times B \; mod \; f(x) + C + D$ to generate the ciphertexts, secret key, and decrypt the data. Based on the phase of the algorithm, C and D are fed to the design. The coefficients of A are stored in *Reg-0* and act as the column of $\mathbb{A}$; *Reg-0* is a register bank includes two sub-bank registers of 128 units ($\frac{n}{2}$) of 8-bit register, one for storing the even and the other one for storing the odd coefficients of $\mathbb{A}$. Each sub-bank register has a shift signal (*Shift-even* and *Shift-odd* signals for even and odd sub-bank register) that only one of them is applied for one clock cycle based on the number of column of $\mathbb{A}$. The value of B is stored in *Reg-1*. The output of *Reg-1* is β_k that is the selector for the *Mux-0*. After each cycle, *Reg-1* is cyclically shifted. As the value of coefficients are in 2's complement, the two most significant bits of $\bar{m}$ determine the range of the coefficient. Thus, the decryption can be done by *Xor* gate.

To execute one time rotation in each column, similar to [90], we execute the multiplication from the last coefficients of $\mathbb{A}^i$ (from $j = \frac{n}{2} - 1$) and $\beta_{\frac{n}{2}-1}$. The proposed design includes two adders. The duty of *Adder-1* is calculating either σ or $\bar{\sigma}$ by controlling the signal *Rot-1*. The result of multiplication (the output of *Mux-0* and *Mux-1* in Fig. 7.1)

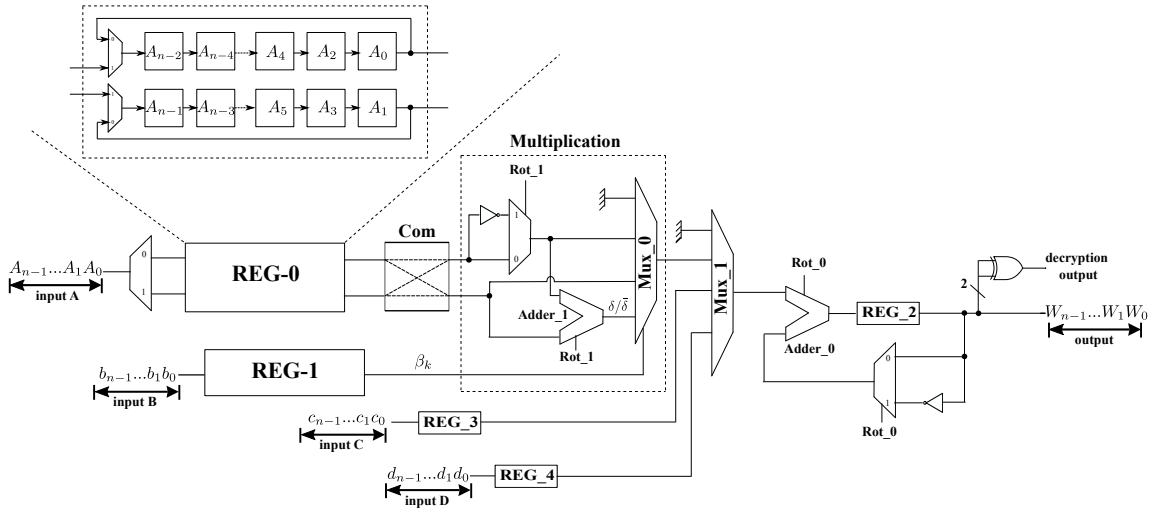


Figure 7.1: The hardware design of the proposed Binary Ring-LWE architecture

is added to the previous results by *Adder-0* and then stored in *Reg-2*. Based on equation (7.6), each polynomial of column of matrix $\mathbb{A}$ contains one negative polynomial with the summation of σ, one polynomial of σ, and one coefficient with $\bar{\sigma}$ for even columns. The rotation is executed by controlling the signals *Rot-0* and *Rot-1*. After the summation of all coefficients of the negative part of $\mathbb{A}$, the rotation is executed by activating the control signal *Rot-0*. When control signal *Rot-1* is active, the result of multiplication is equal to $\bar{\sigma}$ (based on the value of β_k). *Rot-1* is active on even columns and when *Rot-0* is equal to '1'.

The control unit of the proposed design contains three counters to control the proposed multiplication. Two 7-bit ($log_2^n - 1$ bit) counters for controlling the number of row (*Count-Row*) and the time for executing the rotation (*Count-Rot*). *Count-Rot* is a down-counter which is set to the rotation's position of the first column. One 8-bit (log_2^n bit) counter determines the number of column (*Count-Col*). When the value of *Count-Rot* is equal to *Count-Row*, *Rot-0* would be activated for one clock cycle. Then, when *Rot-0* is activated and the least significant bit of *Count-Col* is '0' (the number of column is an even number), *Rot-1* would be activated for one clock cycle. The multiplication finishes when the value of *Count-Row* and *Count-Col* are equal to $\frac{n}{2} - 1$ and $n - 1$, respectively. At this time, *Fin-Flag* will equal to '1'. The functionality of counters to generate the control signals *Rot-0* and *Rot-1* are shown in Fig. 7.2. Three control signals (*flag-Rot*, *flag-Row*, and *flag-Col*) handle

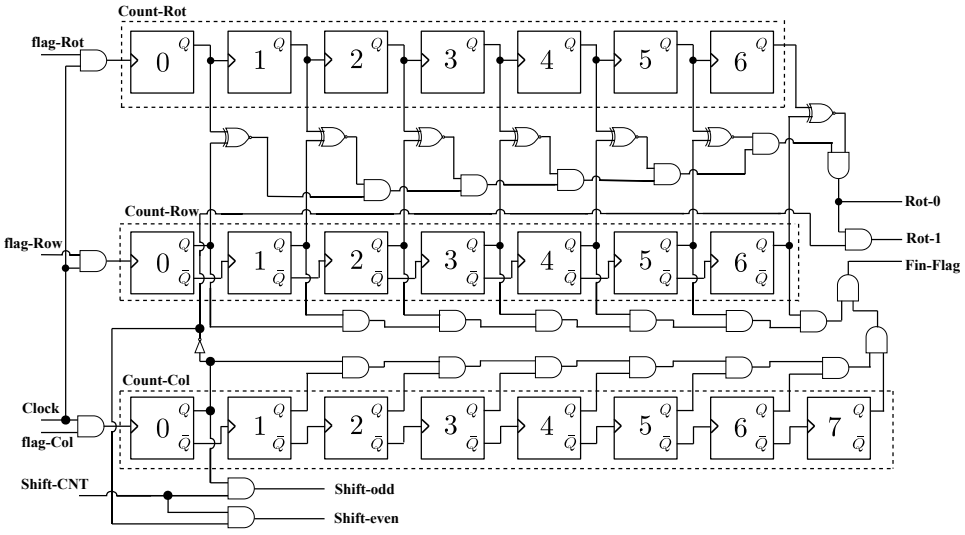


Figure 7.2: The proposed counters architecture of the proposed design

the value of the counters. The duty of *Shift-CNT* is applying the shift on the sub-bank registers of *Reg-0*, based on being odd or even of the number of column. The proposed design requires n clock cycles to store the coefficients into *Reg-0* and *Reg-1*. Since after each column of multiplication one shift should be applied to the sub-bank registers, the novel multiplication takes $\frac{n}{2} \times n + n$ clock cycles. Also, each addition to C and D polynomials need another n clock cycles. Thus, the proposed crypto-system can execute encryption in $7n + n^2$ clock cycles. The decryption and key generation take $\frac{n}{2} \times n + 3n$ clock cycles.

As it is obvious, the proposed architecture executes in a constant number of clock cycles, and there is no conditional branches and dependency between the inputs and ciphertext. Also, the latency of the design is the same for executing of the three phases of the algorithm. Based on [48] [66] [85], the design is secure against timing attacks. Furthermore, the proposed crypto-system is resistance against exploring the secret key by SPA [90]. The multiplication is column-based and two coefficients are processed at each clock cycle, and the entire coefficients of multiplier and multiplicand are participated in each cycle of the multiplication. Also, at least one operation including read or write and addition is executed.

7.4 Implementation Results, Simulation, and Comparison

The implementation results of the proposed architecture and comparison with similar works are presented in this section. The proposed architecture has been implemented and tested on FPGA Vertix-7 platform, which is the mostly used FPGA in previous articles, with VHDL. The synthesis, simulation, and post-placement timing result are obtained by using ISE 14.7 and ISim 14.7. The results of the proposed design and similar works are written in Table 7.4. The proposed design requires 67.5k and 33.5k clock cycles for encryption and decryption phases, respectively, with consideration the storing time in registers, which are less than most of the previous lightweight designs.

The hardware designs are mainly implemented on Xilinx FPGAs or Intel FPGAs. As the architecture of these two platforms are different, having a fair comparison between them is difficult. To make a fair comparison with other similar works, we normalized the occupation area by calculating the Equivalent Number of Slices (ENS). [87] converted the DSP blocks and BRAMs into ENS. Also, based on the empirical benchmark data of [96], one adaptive logic module (ALM) in Intel FPGA is equivalent to 1.3 Slices in Xilinx FPGA.

As it is obvious from Table 7.4, ENS of the proposed design is better than most of the previous works. [90] used add and shift column-based multiplication that requires more clock cycles and less area. The main reason that the architecture of [28] occupied less area, compared to our design, is that this work only executed the decryption phase. The structure of [97] and [92] included parallel architectures that can process more coefficients in a cycle and were designed for high-speed applications. [1] implemented a lightweight architecture for Ring-Bin LWE on ASIC platforms. The proposed architecture and method improved the number of clock cycles by 46.6% and 37% over [1] for encryption and decryption, respectively.

For better comparison between different implementations, we evaluated AT metric to consider the effect of execution time over the occupied area that is calculated $Area \times Time$. Compared to the previous similar works, the proposed multiplication and architecture

improved the AT by 41% and 97% over [90] and [95], respectively.

Table 7.4: FPGA Implementation Results and Comparison

Reference	Device	Type	Freq (MHz)	LUT/FF/Slice	DSP /BRAM	#Clock cycles	Time (μs)	ENS	AT Enc./Dec
This work	Virtex-7	Enc./Dec.	427.095	344/604/189	0	67.5k/33.5k	158.24/78.52	189	29.9k/14.84k
[90]	Virtex-7	Enc./Dec.	434.32	380/640/165	0	133k/66k	307.94/153.67	165	50.81k/25.35k
[1][1]	ASIC 65nm	Enc./Dec.	33.3	7.6k[2]	-	126.54k/53.28k	3.8k/1.9k	-	-
[28]	Spartan-6	Dec.	-/135	57/30/19	0/2	-/65.79k	-/487.4	131	-/63.84k
[95]	Straix V	Enc./Dec	316.96	1864[3]	0	131k/65k	414/207	2423.2	1003.2k/501.6k

[1,2] This design was implemented on ASIC platform, and the reported number is the number of gates in ASIC 65nm.

[3] This is the number of adaptive logic module (ALM) for Intel FPGA.

Part IV

Summary and Future Work

8. Conclusion

8.1 Summary and Conclusion

Cryptography plays an important role in every network to keep the privacy and confidentiality. Every network has three layers; cloud, edge, and end-node. The number of connected devices to the IoT is increasing day by day. Needless to say, security is an essential part of every communication network. Each layer of IoT requires a specific crypto-system to maintain the security. As most of the connected devices to IoT are tiny devices with limited resources, providing end-to-end security is very important. As a result, designing a lightweight crypto-system for resource-constraint devices is important. Meanwhile, with the Advent of quantum computers, most of the current crypto-systems are endangered; and they should be replaced by post-quantum cryptography (PCQ). AES is a secure symmetric cryptography algorithm with a high level of security, which is widely used in many applications and networks. Moreover, AES-256 is secure against quantum attacks. Also, AES is used as a random number generator for other crypto-systems. Lattice-based cryptography is a secure method against quantum attacks. Binary Ring-LWE is a LBC technique for providing the security for IoT devices.

In chapter 3, a high throughput and high FPGA efficiency (FPGA-Eff) implementation of AES-128 algorithm in counter (CTR) mode for high-traffic applications has been presented. In this design, loop unrolling, inner pipelining, and outer pipelining techniques are employed. Using the pipeline and loop unrolling techniques have increased the security against hardware-dependent attack. The main important factors for achieving the high throughput and efficiency are: reducing the design's delay by exchanging and merg-

ing AES functions; optimising the Sub-Bytes, by employing New-Aff, and Mix-Columns; and reducing the critical path of the composite field arithmetic. To achieve the low delay followed by a high frequency of the design, the efforts have been devoted to make an approximate equal delay between different sub-pipeline stages. The target FPGA was Vertix-5 (XC5VLX85-FF676-3) and the achieved frequency, throughput, and FPGA-Eff were 622.4MHz, 79.7Gbps, and 13.3Mbps/slice, respectively. In comparison with others, the results show that this implementation of AES in terms of throughput, maximum frequency, and efficiency provides the best performance.

In chapter 4, a lightweight AES architecture for resource-constrained IoT devices was designed. The design had 8-bit data-path and included two specified register banks for storing plain-text, keys, and intermediate results. To reduce the required logic, Shift-Rows was run inside of the State-Register. Also, the design had an optimized Sub-Bytes that was shared with encryption and the key expansion phase, which led to reduce the area by 15.5% on 65nm technology. Furthermore, Mix-columns with 8-bit input and output was designed, which is a proper block for low-area design. To reduce the power consumption, the clock gating technique in different blocks of the design was applied, which led to reduce the power consumption by 18.9%.

To verify the architecture, the design was implemented on Virtex-5 FPGA. After that, the proposed design was implemented and verified on 65nm technology on different timing constraints. The core area without power rings, core area with power rings, and chip area are 5448.59μm^2, 7783.77μm^2, and 11713.57μm^2, respectively. The proposed design improved the area and Area-Delay-Product (ADP) for chip design over the state-of-the-art work by 2.4% and 71.7%, respectively. Also, the core area with power rings was improved by 22.1% over the best similar implementation. The power consumption of the design was simulated in different timing constraints. To make a fair comparison with other similar works, the normalized power of previous works was calculated. The power consumption of the proposed design was better than most of the previous works. According to the result and NIST report, the proposed lightweight AES design is suitable for resource-constrained

devices and can be supplied by low power devices.

In chapter 5, an optimized architecture for all three phases of Binary Ring-LWE has been designed. This architecture is suitable for resource-constrained IoT devices and tiny end-nodes. The proposed method of performing multiplication, *In-place Rot-Col-Mul*, is a column-based multiplication followed by in-place modular reduction and anti-circular rotation. Instead of using each multiplier coefficient in each cycle, all multipliers are employed for calculating the result. The main advantage of this multiplication method is that only one anti-circular rotation is performed in each column that reduces latency and area of the design. So that, one *Mux*, one *Not* gate, one *Adder*, and one shift register are sufficient for anti-circular rotation and multiplication.

The design has been implemented on TSMC-65*nm* and FPGA technology. ASIC implementation results show that the proposed lightweight design is a feasible crypto-system for the area- and power-constrained edge IoT devices, as it can be implemented by only 3.2k gates and consumes $0.196mW$. The ASIC implementation achieved a maximum frequency of 500 MHz, and improvement of power and area over the state-of-the-art design (using similar time constraints) by 48.42% and 57.8%, respectively. Furthermore, the design has been implemented on FPGAs and the number of occupied slices is 165 and 146 for Virtex-7 and Spartan-6, respectively. Compared to similar works, the FPGA results show that the proposed design and method has less latency and fewer resources.

In chapter 6, in order to increase the security and resistance against fault attack, fault resilient countermeasure is added to the optimized implementation of Binary Ring-LWE. Because of the distributed nature of IoT network, fault attack is one major threat to end-node IoT devices. The adversary tries to gain the plain-text and secret key by manipulating the data. Zeroing and skipping faults are the main attacks over the crypto-systems. The fault resiliency design has been implemented on FPGA technology. The FPGA implementation achieved a maximum frequency of 210.5 MHz on Virtex-7 and occupied 423 Slices, which is only 1% of total available slices. The proposed fault resilient occupied very low area on Virtex-7 and can be used as a crypto accelerator for other FPGA implementation designs.

In chapter 7, a low latency and lightweight Binary Ring-LWE architecture for resource-constraint devices was designed. To reduce the number of clock cycles, a novel modular multiplication was designed. The proposed multiplication was done by a 4-1 *Mux*, one 2-1 *Mux*, one *Not* gate, and one adder that executed the multiplication in $n \times \frac{n}{2}$ clock cycles. Thus, the proposed method improved the latency by factor of two. Also, the hardware architecture was designed based on the proposed multiplication. The architecture included a specific register bank that contained two sub-bank registers; one for storing the odd coefficients and the other one for storing the even coefficients. The control unit had three counters to handle the multiplication and reduction. The proposed architecture is implemented on Xilinx FPGA platform. The implementation results showed the supremacy of the proposed design and method over the previous similar works. To make a fair comparison, hardware resources of FPGA implementation were normalized. The $Area \times Time$ and the number of clock cycles for the proposed design are 29.9k/ 14.84k and 67.5k/ 33.5k, for encryption and decryption respectively. Compared to the previous similar works, the proposed design improved the *AT* (*Area* $\times$ *Time*) by 41% and 97% over [90] and [95], respectively.

8.2 Future Research

Needless to say, it is necessary to re-design and implement the crypto-systems that are secure against quantum attack. In these years, most research projects have been done and most articles have been published in the area of post-quantum cryptography. Ring-LWE and Binary Ring-LWE are two examples of these cryto-systems. As post-quantum cryptography is at its first step, there are different aspect of hardware implementation that can be considered. Even though this thesis tried to reduce the execution time for Binary Ring-LWE, the latency of the proposed multiplication for Binary Ring-LWE is not best optimized. As a result, there will be a potential research to reduce the number of clock cycles for Birary Ring-LWE. Designing an architecture for multiplication by four coefficients will reduce the number of clock cycles efficiently.

Recently, NIST announced the list of candidates for post-quantum cryptography. As

there are few hardware implementation for NIST candidate, future research work will focus on NIST candidates and evaluate the hardware implementations on different platforms. For example, multiplication is one of the main operations of LBC. Designing and implementing an optimized multiplication unit for PQC has a high impact on execution time and occupied area. Moreover, different layers of the network require a specific crypto-system; thus, designing and implementing a specific crypto-system based on the application and layer of the network is important.

Even though the PQCs are secure against quantum attacks, evaluating the different physical attacks against the crypto-system is very important that should be evaluated for different hardware implementation. Timing, fault injection, and power attacks are some examples of physical attacks. A reliable crypto-system should be secure against these kind of attacks. Thus, designing a countermeasure or fault resilient for crypto-systems based on LBC should be considered.

References

[1] S. Ebrahimi, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "Post-Quantum Cryptoprocessors Optimized for Edge and Resource-Constrained Devices in IoT," *IEEE Internet of Things Journal*, 2019.

[2] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-quantum cryptography*. Springer, 2009, pp. 147–191.

[3] X. Zhang and K. 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.

[4] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 2018.

[5] 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.

[6] Z. Liu, K.-K. R. Choo, and J. Grossschadl, "Securing Edge Devices in the Post-Quantum Internet of Things Using Lattice-Based Cryptography," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 158–162, 2018.

[7] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on Post-Quantum Cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016, vol. 12.

[8] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[9] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. Huss, "On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes," in *Inter-*

national Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2012, pp. 512–529.

[10] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2014, pp. 371–391.

[11] J. Buchmann, F. Göpfert, T. Güneysu, T. Oder, and T. Pöppelmann, "High-Performance and Lightweight Lattice-Based Public-Key Encryption," in *Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security*. ACM, 2016, pp. 2–9.

[12] P. FIPS, "197: Advanced encryption standard (AES)," *National Institute of Standards and Technology*, vol. 26, 2001.

[13] K. Shahbazi, M. Eshghi, and R. F. Mirzaee, "Design and implementation of an ASIP-based cryptography processor for AES, IDEA, and MD5," *Engineering science and technology, an international journal*, vol. 20, no. 4, pp. 1308–1317, 2017.

[14] L. Ali, I. Aris, F. S. Hossain, and N. Roy, "Design of an ultra high speed AES processor for next generation IT security," *Computers & Electrical Engineering*, vol. 37, no. 6, pp. 1160–1170, 2011.

[15] A. Soltani and S. Sharifian, "An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA," *Microprocessors and Microsystems*, vol. 39, no. 7, pp. 480–493, 2015.

[16] N. Ahmad, R. Hasan, and W. M. Jubadi, "Design of AES S-Box using combinational logic optimization," in *IEEE Symposium on Industrial Electronics and Applications (ISIEA)*. IEEE, 2010, pp. 696–699.

[17] D.-H. Bui, D. Puschini, S. Bacles-Min, E. Beigné, and X.-T. Tran, "AES Datapath Optimization Strategies for Low-Power Low-Energy Multisecurity-Level Internet-of-

Things Applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3281–3290, 2017.

[18] S. Banik, A. Bogdanov, and F. Regazzoni, "Exploring Energy Efficiency of Lightweight Block Ciphers," in *International Conference on Selected Areas in Cryptography*. Springer, 2015, pp. 178–194.

[19] W. Zhao, Y. Ha, and M. Alioto, "AES architectures for minimum-energy operation and silicon demonstration in 65nm with lowest energy per encryption," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2015, pp. 2349–2352.

[20] H. K. Kim and M. H. Sunwoo, "Low Power AES Using 8-Bit and 32-Bit Datapath Optimization for Small Internet-of-Things (IoT)," *Journal of Signal Processing Systems*, vol. 91, no. 11-12, pp. 1283–1289, 2019.

[21] Van-Phuc, V.-L. Dao, and C.-K. Pham, "An ultra-low power AES encryption core in 65nm SOTB CMOS process," in *International SoC Design Conference (ISOCC)*. IEEE, 2016, pp. 89–90.

[22] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba Algorithm for Efficient Implementations," *IACR Cryptol. ePrint Arch.*, vol. 2006, pp. 1–17, 2006.

[23] A. Shreedhar, K.-S. Chong, N. Lwin, N. Kyaw, L. Nalangilli, W. Shu, J. Chang, and B.-H. Gwee, "Low Gate-Count Ultra-Small Area Nano Advanced Encryption Standard (AES) Design," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[24] V.-P. Hoang, V.-L. Dao, and C.-K. Pham, "Design of ultra-low power AES encryption cores with silicon demonstration in SOTB CMOS process," *Electronics Letters*, vol. 53, no. 23, pp. 1512–1514, 2017.

[25] C. Hocquet, D. Kamel, F. Regazzoni, J.-D. Legat, D. Flandre, D. Bol, and F.-X. Standaert, "Harvesting the potential of nano-CMOS for lightweight cryptography:

an ultra-low-voltage 65 nm AES coprocessor for passive RFID tags," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 79–86, 2011.

[26] S. Mathew, S. Satpathy, V. Suresh, M. Anders, H. Kaul, A. Agarwal, S. Hsu, G. Chen, and R. Krishnamurthy, "340 mV–1.1 V, 289 Gbps/W, 2090-Gate NanoAES Hardware Accelerator With Area-Optimized Encrypt/Decrypt GF$(2^4)^2$ Polynomials in 22 nm Tri-Gate CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 4, pp. 1048–1058, 2015.

[27] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core," in *9th EUROMICRO conference on digital system design (DSD'06)*. IEEE, 2006, pp. 577–583.

[28] A. Aysu, M. Orshansky, and M. Tiwari, "Binary Ring-LWE hardware with power side-channel countermeasures," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1253–1258.

[29] S. Ebrahimi and S. Bayat-Sarmadi, "Lightweight and Fault Resilient Implementations of Binary Ring-LWE for IoT Devices," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6970–6978, 2020.

[30] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu, "Practical CCA2-Secure and Masked Ring-LWE Implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 142–174, 2018.

[31] J. M. Granado-Criado, M. A. Vega-Rodríguez, J. M. Sánchez-Pérez, and J. A. Gómez-Pulido, "A new methodology to implement the AES algorithm using partial and dynamic reconfiguration," *Integration*, vol. 43, no. 1, pp. 72–80, 2010.

[32] K. Shahbazi and M. Eshghi, "Design of a specific instructions set processor for AES algorithm," *International Journal of Computer Applications*, vol. 93, no. 4, 2014.

[33] S. Oukili, S. Bri, and A. S. Kumar, "High speed efficient FPGA implementation of pipelined AES S-Box," in *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*. IEEE, 2016, pp. 901–905.

[34] R. R. Rachh, B. Anami, and P. A. Mohan, "Efficient implementations of S-box and inverse S-box for AES algorithm," in *TENCON 2009-2009 IEEE Region 10 Conference*. IEEE, 2009, pp. 1–6.

[35] P. Shastry, A. Agnihotri, D. Kachhwaha, J. Singh, and M. Sutaone, "A combinational logic implementation of S-box of AES," in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2011, pp. 1–4.

[36] M. I. Soliman and G. Y. Abozaid, "FPGA implementation and performance evaluation of a high throughput crypto coprocessor," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1075–1084, 2011.

[37] Z.-r. Li, Y.-q. Zhuang, C. Zhang, and J. Gang, "Low-power and area-optimized VLSI implementation of AES coprocessor for Zigbee system," *The Journal of China Universities of Posts and Telecommunications*, vol. 16, no. 3, pp. 89–94, 2009.

[38] H. Anwar, M. Daneshtalab, M. Ebrahimi, J. Plosila, and H. Tenhunen, "FPGA implementation of AES-based crypto processor," in *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, 2013, pp. 369–372.

[39] J. M. Granado-Criado and M. A. Vega-Rodríguez, "Hardware coprocessors for high-performance symmetric cryptography," *The Journal of Supercomputing*, vol. 73, no. 6, pp. 2456–2482, 2017.

[40] S.-M. Yoo, D. Kotturi, D. Pan, and J. Blizzard, "An AES crypto chip using a high-speed parallel pipelined architecture," *Microprocessors and Microsystems*, vol. 29, no. 7, pp. 317–326, 2005.

[41] Z. Guo, G. Li, and Y. Liu, "Dynamic reconfigurable implementations of AES algorithm

based on pipeline and parallel structure," in *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, vol. 3. IEEE, 2010, pp. 257–260.

[42] C. Wang and H. M. Heys, "Using a pipelined S-box in compact AES hardware implementations," in *Proceedings of the 8th IEEE International NEWCAS Conference 2010*. IEEE, 2010, pp. 101–104.

[43] D. Canright, "A very compact S-box for AES," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 441–455.

[44] S. S. M. Aldabbagh and I. F. T. Al Shaikhli, "Security of PRESENT S-box," in *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*. IEEE, 2012, pp. 219–222.

[45] S. Abed, R. Jaffal, B. J. Mohd, M. Alshayeji *et al.*, "FPGA modeling and optimization of a Simon lightweight block cipher," *Sensors*, vol. 19, no. 4, p. 913, 2019.

[46] B. Mazumdar, S. S. Ali, and O. Sinanoglu, "A compact implementation of Salsa20 and its power analysis vulnerabilities," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, pp. 1–26, 2016.

[47] S. Bhasin, T. Graba, J.-L. Danger, and Z. Najm, "A look into SIMON from a side-channel perspective," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2014, pp. 56–59.

[48] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Annual International Cryptology Conference*. Springer, 1996, pp. 104–113.

[49] S. Qu, G. Shou, Y. Hu, Z. Guo, and Z. Qian, "High throughput, pipelined implementation of AES on FPGA," in *2009 International Symposium on Information Engineering and Electronic Commerce*. IEEE, 2009, pp. 542–545.

[50] H. Kouzehzar, M. N. Moghadam, and P. Torkzadeh, "A high data rate pipelined architecture of AES encryption/decryption in storage area networks," in *Electrical Engineering (ICEE), Iranian Conference on*. IEEE, 2018, pp. 23–28.

[51] R. R. Farashahi, B. Rashidi, and S. M. Sayedi, "FPGA based fast and high-throughput 2-slow retiming 128-bit AES encryption algorithm," *Microelectronics journal*, vol. 45, no. 8, pp. 1014–1025, 2014.

[52] H. Li, J. Ding, and Y. Pan, "Cell array reconfigurable architecture for high-efficiency AES system," *Microelectronics Reliability*, vol. 52, no. 11, pp. 2829–2836, 2012.

[53] M. El Maraghy, S. Hesham, and M. A. Abd El Ghany, "Real-time efficient FPGA implementation of AES algorithm," in *2013 IEEE International SOC Conference*. IEEE, 2013, pp. 203–208.

[54] M. H. Rais and S. M. Qasim, "FPGA implementation of Rijndael algorithm using reduced residue of prime numbers," in *2009 4th International Design and Test Workshop (IDT)*. IEEE, 2009, pp. 1–4.

[55] M. Rais and S. M. Qasim, "Efficient hardware realization of advanced encryption standard algorithm using Virtex-5 FPGA," *International Journal of Computer Science and Network Security*, vol. 9, no. 9, pp. 59–63, 2009.

[56] J. S. Banu, M. Vanitha, J. Vaideeswaran, and S. Subha, "Loop parallelization and pipelining implementation of AES algorithm using OpenMP and FPGA," in *2013 IEEE International Conference ON Emerging Trends in Computing, Communication and Nanotechnology (ICECCN)*. IEEE, 2013, pp. 481–485.

[57] L. Alliance, "LoRaWAN specification," *LoRa Alliance*, pp. 1–82, 2015.

[58] C. Patrick and P. Schaumont, "The Role of Energy in the Lightweight Cryptographic Profile," in *NIST Lightweight Cryptography Workshop*, 2016, pp. 1–16.

[59] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the Limits: A Very Compact and a Threshold Implementation of AES," in *Annual International*

Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2011, pp. 69–88.

[60] T. Järvinen, P. Salmela, P. Hämäläinen, and J. Takala, "Efficient byte permutation realizations for compact AES implementations," in *13th European Signal Processing Conference*. IEEE, 2005, pp. 1–4.

[61] C. Paar, "Efficient VLSI architectures for bit-parallel computation in Galois fields," *PhD Thesis, Inst. for Experimental Math., Univ. of Essen*, 1994.

[62] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001, pp. 239–254.

[63] E. N. Mui, R. Custom, and D. Engineer, "Practical Implementation of Rijndael S-Box Using Combinational Logic," *Custom R&D Engineer Texco Enterprise Pvt. Ltd*, 2007.

[64] A. E. Standard, "Federal information processing standards publication 197," *FIPS PUB*, pp. 1–51, 2001.

[65] A. Reyhani-Masoleh, M. Taha, and D. Ashmawy, "New Area Record for the AES Combined S-Box/Inverse S-Box," in *25th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2018, pp. 145–152.

[66] K. Shahbazi and S.-B. Ko, "High throughput and area-efficient FPGA implementation of AES for high-traffic applications," *IET Computers & Digital Techniques*, vol. 14, no. 6, pp. 344–352, 2020.

[67] N. Cho, S.-J. Song, S. Kim, S. Kim, and H.-J. Yoo, "A 5.1-/spl mu/W UHF RFID tag chip integrated with sensors for wireless environmental monitoring," in *Proceedings of the 31st European Solid-State Circuits Conference (ESSCIRC)*. IEEE, 2005, pp. 279–282.

[68] Y. Hong, C. F. Chan, J. Guo, Y. S. Ng, W. Shi, L. K. Leung, K. N. Leung, C. S. Choy, and K. P. Pun, "Design of passive UHF RFID tag in 130nm CMOS technology," in

APCCAS IEEE Asia Pacific Conference on Circuits and Systems. IEEE, 2008, pp. 1371–1374.

[69] R. Chaudhary, G. S. Aujla, N. Kumar, and S. Zeadally, "Lattice-Based Public Key Cryptosystem for Internet of Things Environment: Challenges and Solutions," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4897–4909, 2019.

[70] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[71] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[72] R. A. Perlner and D. A. Cooper, "Quantum resistant public key cryptography: a survey," in *Proceedings of the 8th Symposium on Identity and Trust on the Internet*. ACM, 2009, pp. 85–93.

[73] R. O. Micciancio D., "Lattice-based Cryptography," *In: Bernstein D.J., Buchmann J., Dahmen E. (eds) Post-Quantum Cryptography*, pp. 147–191, 2009.

[74] D. Jao and L. De Feo, "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies," in *International Workshop on Post-Quantum Cryptography*. Springer, 2011, pp. 19–34.

[75] T. N. Tan and H. Lee, "High-Secure Fingerprint Authentication System Using Ring-LWE Cryptography," *IEEE Access*, vol. 7, pp. 23 379–23 387, 2019.

[76] T. Tan and H. Lee, "High-Secure Low-Latency Ring-LWE Cryptography Scheme for Biomedical Images Storing and Transmitting," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–4.

[77] T. Nguyen Tan and H. Lee, "Efficient-Scheduling Parallel Multiplier-Based Ring-LWE Cryptoprocessors," *Electronics*, vol. 8, no. 4, p. 413, 2019.

[78] C. P. Rentería-Mejía and J. Velasco-Medina, "High-Throughput Ring-LWE Cryptoprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2332–2345, 2017.

[79] T. Pöppelmann and T. Güneysu, "Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware," in *International Conference on Selected Areas in Cryptography*. Springer, 2013, pp. 68–85.

[80] F. Göpfert, C. van Vredendaal, and T. Wunderer, "A Hybrid Lattice Basis Reduction and Quantum Search Attack on LWE," in *International Workshop on Post-Quantum Cryptography*. Springer, 2017, pp. 184–202.

[81] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2007, pp. 450–466.

[82] T. Wunderer, "Revisiting the Hybrid Attack: Improved Analysis and Refined Security Estimates," *IACR Cryptology ePrint Archive*, vol. 2016, p. 733, 2016.

[83] Z. Liu, H. Seo, S. Sinha Roy, J. Großschädl, H. Kim, and I. Verbauwhede, "Efficient Ring-LWE encryption on 8-bit AVR processors," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 663–682.

[84] T. Prescott, *Random Number Generation Using AES*, available: http://ww1.microchip.com/downloads/en/Devic eDoc/article_random_number.pdf.

[85] K. Shahbazi and S.-B. Ko, "Area efficient nano-AES implementation for Internet of Things devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 1, pp. 136–148, 2020.

[86] T. Schneider, A. Moradi, and T. Güneysu, "ParTI-Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks," in *Annual International Cryptology Conference*. Springer, 2016, pp. 302–332.

[87] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized Schoolbook Polynomial Multiplication for Compact Lattice-Based Cryptography on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 10, pp. 2459–2463, 2019.

[88] D. Liu, C. Zhang, H. Lin, Y. Chen, and M. Zhang, "A Resource-Efficient and Side-Channel Secure Hardware Implementation of Ring-LWE Cryptographic Processor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 4, pp. 1474–1483, 2019.

[89] Y. Zhang, C. Wang, D. E. S. Kundi, A. Khalid, M. O'Neill, and W. Liu, "An Efficient and Parallel R-LWE Cryptoprocessor," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 886–890, 2020.

[90] K. Shahbazi and S.-B. Ko, "Area and power efficient post-quantum cryptosystem for IoT resource-constrained devices," *Microprocessors and Microsystems*, p. 104280, 2021.

[91] N. Bindel, J. Buchmann, and J. Krämer, "Lattice-based signature schemes and their sensitivity to fault attacks," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2016, pp. 63–77.

[92] A. Sarker, M. M. Kermani, and R. Azarderakhsh, "Fault Detection Architectures for Inverted Binary Ring-LWE Construction Benchmarked on FPGA," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 4, pp. 1403–1407, April 2021.

[93] F. Valencia, T. Oder, T. Güneysu, and F. Regazzoni, "Exploring the Vulnerability of R-LWE Encryption to Fault Attacks," in *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems*, 2018, pp. 7–12.

[94] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, p. 34, 2009.

[95] P. He, U. Guin, and J. Xie, "Novel Low-Complexity Polynomial Multiplication Over Hybrid Fields for Efficient Implementation of Binary Ring-LWE Post-Quantum Cryptography," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 383–394, 2021.

[96] "White Paper Stratix II vs. Virtex-4 Density Comparison," Augest 2005.

[97] J. Xie, P. He, X. M. Wang, and J. L. Imana, "Efficient Hardware Implementation of Finite Field Arithmetic AB+C over Hybrid Fields for Post-Quantum Cryptography," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–6, 2021.