

**A Scalar Multiplication in Elliptic Curve  
Cryptography with Binary Polynomial Operations in  
Galois Field**

Thesis submitted to

The Faculty of Computer Science and Information Technology

University Malaya

By

**Hero Modares**

In Partial fulfilment of the Requirement for the Degree of

Master of Computer Science

Supervisor:

Dr. Rosli Salleh

October 2009



## Abstract

A fundamental building block for digital communication is the Public-key cryptography systems. Public-Key cryptography (PKC) systems can be used to provide secure communications over insecure channels without exchanging a secret key. Implementing Public-Key cryptography systems is a challenge for most application platforms when several factors have to be considered in selecting the implementation platform. The most popular public-key cryptography systems nowadays are RSA and Elliptic Curve Cryptography (ECC). ECC is considered much more suitable than other public-key algorithms. It uses lower power consumption, has higher performance and can be implemented on small areas that can be achieved by using ECC. There is no subexponential-time algorithm in solving the Elliptic curve discrete logarithm problem. Therefore, it offers smaller key size with equivalent security level compared with the other public key cryptosystems. Finite fields (or Galois fields) is considered as an important mathematical theory. Thus, it plays an important role in cryptography. As a result of their carry free arithmetic property, they are suitable to be used in hardware implementation in ECC. In cryptography the most common finite field used is binary field  $GF(2^m)$ . Our design performs all basic binary polynomial operations in Galois Field (GF) using a microcode structure. It uses a bit-serial and pipeline structure for implementing GF operations. Due to its bit-serial architecture, it has a low gate count and a reduced number of I/O pins. The proposed design is implemented in Verilog HDL. Xilinx ISE is used for synthesis and simulation. The result of Verilog code is checked by using the previous written Matlab code.

## **Acknowledgement**

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I want to thank the Faculty of Computer Science and Information Technology for giving me permission on commences this thesis to do necessary research work.

In the first place I would like to record my gratitude to Dr. Rosli Salleh and Mr. Fatemi for their supervision, advice and guidance from the very early stage of this research as well as giving me extraordinary experiences throughout the work. Above all and the most needed, they provided me unflinching encouragement and support in various ways. Which exceptionally inspire and enrich my growth as a student, a researcher wants to be.

Words fail me to express my appreciation to my family whose dedication, support and persistent confidence in me, has taken the load off my shoulder and enabled me to complete this thesis.

I also want to thank Mr. Yasser Salem and Mr. Majid Talebi for all their help, support, interest and valuable hints.

This work was supported in part by the University of Malaya, Kuala Lumpur Malaysia under Grant (SF051/2007A).

## **Publications**

- 1- H. Modares, M. R. H. Fatemi , R. Salleh,2009,” Elliptic Curve Cryptography: The Next Generation Encryption Technology for Secure Data Communication”,E-CASE 2009 conference-Singapore.
- 2- M. T. Shahgoli, H. Modares, Y. Salem, 2009,” Elliptic Curve Cryptography in E-commerce and Smart Card”,E-CASE 2009 conference-Singapore.
- 3- Modares H., Salem Y., M. R. H. Fatemi, R. Salleh, 2009,”Application of Elliptic Curve Cryptography”, 3rd International Conference on Informatics and Technology 2009, Kuala Lumpur, Malaysia.

## Contents

Abstract .....	I
Acknowledgement .....	II
Publications .....	III
List of Figures .....	VIII
List of Tables .....	IX
List of Algorithms .....	X
Acronyms .....	XI
<b>Chapter 1</b>	
Introduction .....	1
1.1. Overview .....	1
1.2. Problem statement .....	3
1.3. Motivation .....	4
1.4. Objective of the research .....	5
1.5. Scope of the Research .....	6
1.6. Thesis Organization .....	6
<b>Chapter 2</b>	
Cryptography .....	8
2.1. Introduction .....	8
2.2. Cryptography components .....	9
2.3. Symmetric-key .....	9
2.3.1. Advantages of symmetric-key cryptography .....	10
2.3.2. Disadvantages of symmetric-key cryptography .....	10
2.4. Public-key Cryptography (Asymmetric-key) .....	11
2.4.1. Advantages of public-key cryptography .....	11
2.4.2. Disadvantages of public-key encryption .....	12
2.5. Hybrid .....	12
2.6. El-Gamal .....	13
2.7. RSA .....	14
2.8. Hash .....	14
2.8.1. Hash Algorithms .....	15
2.8.2. LanManager (LM) .....	15
2.8.3. Message Digest algorithm 5 (MD5) .....	15
2.8.4. SHA-0/SHA-1 .....	16
2.8.5. SHA-2 .....	16
2.8.6. Uses of Hashing .....	17
	IV

## Chapter 3

Elliptic Curve Cryptography .....	20
3.1. Introduction .....	20
3.2. Mathematical of Elliptic Curve Cryptography .....	21
3.2.1. Point Multiplication .....	21
3.2.2. Point Addition .....	22
3.2.3. Point Doubling .....	23
3.3. Finite Fields .....	23
3.3.1. Introduction .....	23
3.3.2. EC on Prime field $F_p$ .....	25
3.3.2.1. Point Addition .....	25
3.3.2.2. Point Subtraction .....	25
3.3.2.3. Point Doubling: .....	26
3.3.3. Elliptic Curve (EC) on Binary field $F_{2^m}$ .....	26
3.3.3.1. Point Addition .....	26
3.3.3.2. Point Subtraction .....	27
3.3.3.3. Point Doubling .....	27
3.4. Elliptic Curve Domain parameters .....	27
3.4.1. Domain parameters for EC over field $F_p$ .....	27
3.4.2. Domain parameters for EC binary fields .....	28
3.5. Field Arithmetic .....	28
3.5.1. Addition .....	29
3.5.2. Subtraction .....	29
3.5.3. Multiplication .....	30
3.5.4. Division .....	30
3.5.5. Multiplicative Inversion .....	30
3.6. Elliptic Curve Cryptography Standards .....	31
3.7. Elliptic Curve Diffie-Hellman .....	33
3.8. Applications of ECC .....	34
3.8.1. Radio Frequency Identification (RFID-Tags) .....	34
3.8.2. Elliptic Curve Digital Signature Algorithm .....	35
3.8.2.1. ECDSA key generation .....	36
3.8.2.2. ECDSA signature generation .....	36
3.8.2.3. ECDSA signature verification .....	37
3.8.3. WIRELESS SENSOR NETWORK (WSN) .....	37

3.9.	Related work.....	38
3.10.	Conclusion.....	40
<b>Chapter 4</b>		
ECC Algorithms and its Applications.....		42
4.1.	Introduction.....	42
4.2.	Methodology.....	42
4.3.	Multiplication.....	43
4.3.1.	Integer multiplication.....	43
4.3.2.	Right-to-left shift-and-add field multiplication in $F_{2^m}$ .....	43
4.3.3.	Montgomery Multiplication Algorithm.....	44
4.4.	Inversion.....	46
4.4.1.	Extended Euclidean algorithm for polynomials.....	47
4.4.2.	Binary inversion algorithm in $F_{2^m}$ .....	48
4.4.3.	Montgomery Modular Inverse Algorithm.....	49
4.5.	Double-and-Add Algorithm.....	50
4.6.	Conclusion.....	52
<b>Chapter 5</b>		
Design and Implementation and Testing.....		53
5.1.	Introduction.....	53
5.2.	Modular Multiplication.....	55
5.2.1.	Bit-Serial Multiplier Structure.....	55
5.3.	Modular Inversion.....	60
5.3.1.	Bit-Serial Inversion Structure.....	61
5.4.	Scalar Multiplication.....	65
5.4.1.	Double-and-Add.....	65
5.5.	ECC Co-Processor.....	69
5.5.1.	Control Unit.....	70
5.5.2.	Arithmetic Unit.....	71
5.5.3.	Testing.....	71
5.6.	Conclusion.....	75
<b>Chapter 6</b>		
6.1.	Conclusion.....	76
6.2.	Contribution.....	77
6.3.	Recommendations for Further Research.....	78
Bibliography.....		79



Appendix A- Elliptic Curve over Field $F_{23}$ .....	84
Appendix B - Multiplication .....	86
Appendix C - Inversion .....	89

## List of Figures

Figure 2.1 - Two-party communication .....	8
Figure 2.2 - Symmetric-key .....	10
Figure 2.3 - Public-key.....	11
Figure 3.1 - Point Addition .....	22
Figure 3.2 - Point Doubling. ....	23
Figure 3.3 - Hierarchy of Elliptic Curve.....	24
Figure 3.4 - (ECDH) key exchange method.....	33
Figure 5.1 - Points over the curve1 .....	54
Figure 5.2 - Points over the curve2 .....	54
Figure 5.3 - Flowchart for Right-to-left algorithm. ....	56
Figure 5.4 - Most significant bit first (MSB) multiplier for $F2^5$ .....	57
Figure 5.5 - The n-bit Multiplier.....	58
Figure 5.6 - Technology Schematic of Modular Multiplication to test 8 bits.....	59
Figure 5.7 – Test Bench Multiplications1.....	59
Figure 5.8 – Test Bench Multiplications2.....	59
Figure 5.9 - Flowchart for Montgomery Inversion algorithm, Contain Phase I and II. ....	61
Figure 5.10 - Bit-Serial Inversion block diagram. ....	62
Figure 5.11 - Inversion Architecture (5-bits).....	63
Figure 5.12 – Test Bench (Phase I) (5-bits).....	64
Figure 5.13 – Test Bench (Phase II) (5-bits).....	64
Figure 5.14 - Addition Data Flow in $GF(2^m)$ .....	67
Figure 5.15 - Doubling Data Flow in $GF(2^m)$ .....	67
Figure 5.16 - ECC Co-Processor.....	70

## List of Tables

Table 1.1 - Mathematical Problem.....	2
Table 1.2 - Public-key System .....	2
Table 2.1 - Hashing Algorithm. ....	16
Table 3.1 - NIST Recommended Key Sizes. ....	20
Table 3.2 - Standards .....	31
Table 5.1 - XOR ( $\oplus$ ), AND ( $\wedge$ ). ....	57
Table 5.2 - Scalar Multiplication. ....	67
Table 5.3 - Comparison of Computation Steps. Double-and-add Scalar Multiplication Algorithm. ....	69

## List of Algorithms

Algorithm 4.1- Integer multiplication .....	43
Algorithm 4.2- Right-to-left shift-and-add field multiplication in $F_{2^m}$ .....	44
Algorithm 4.3- Montgomery Multiplication over $GF(p)$ .....	45
Algorithm 4.4- Montgomery Multiplication over $GF(2^m)$ .....	45
Algorithm 4.5- Extended Euclidean algorithm for binary polynomials .....	47
Algorithm 4.6- Inversion in $F_2^m$ using the extended Euclidean algorithm .....	48
Algorithm 4.7- Binary algorithm for inversion in $F_2^m$ .....	49
Algorithm 4.8- Montgomery Modular Inverse Algorithm .....	50
Algorithm 4.9- Double-and-Add Algorithm .....	51
Algorithm 5.1- Shift Right .....	64
Algorithm 5.2- Shift Left .....	65
Algorithm 5.3- Double-and-Add (Most Significant Bit) .....	66
Algorithm 5.4- Double-and-Add (Last Significant Bit) .....	66

## Acronyms

ALU	Arithmetic Logic Unit
CU	Control Unit
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
EEA	Extended Euclidean algorithm
FFs	Flip-Flops
GF	Galois Field
HDL	Hardware Description Language
IFP	Integer Factorization Problem
LSB	less Significant Bit
MAC	Message Authentication Codes
MSB	Most Significant Bit
PKC	Public key cryptography
RSA	Riverst-Shamir-Adleman
RTL	Register Transfer Level

# Chapter 1

## Introduction

This chapter is made up of four main sections. The first section explains the background of the research purpose. The second section describes the motivation and objectives. The last two sections explain the scope of research and research organization.

### 1.1.Overview

Since a lot of sensitive data such as credit card numbers and social security numbers are transmitted over the Internet during transactions. Securing electronic transaction becomes a very important issue. An efficient way to protect and secure the information is by using cryptography which can be used to provide and assure confidentiality and integrity of the transactions (Mackenzie, et al. 1996).

The history of cryptography is long and interesting. It had a very considerable turning point when two researchers from Stanford, Whitfield Diffie and Martin Hellman, published the paper “New Directions in Cryptography” in 1976. They preface the new idea of public key cryptography in the paper.

Public-key cryptography and symmetric-key cryptography are two main categories of cryptography. The Well-known public-key cryptography algorithms are RSA (Rivest, et al. 1978), El-Gamal and Elliptic Curve Cryptography. Presently, there are only three problems of public key cryptosystems that are considered to be both secure and effective (Certicom, 2001). Table 1.1 shows these mathematical problems and the cryptosystems that rely on such problems. Table 1.2 shows the complexity of calculative for each of these problems where ‘ $n$ ’ is the length of the keys used. (Sandoval 2008), (Kumar 2006)

	Mathematical Problem	Detail	Cryptosystems
1	Integer Factorization Problem (IFP).	Given an integer 'n', find its prime factorization.	RSA
2	Discrete Logarithm Problem (DLS).	Given integer 'g' and 'h', find 'x' such that $h = g^x \text{ mod } n$	ELGedal, DSA Diffie-Hellman(DH)
3	Elliptic Curve Discrete Logarithm Problem (ECDLP).	Given points 'P' and 'Q' on curve, find 'x' such that $Q=xP$	ECDSA, EC-Diffie-Hellman(DH)

**Table 1.1- Mathematical Problem**

	Public-key System	Best known methods for solving mathematical problem	Runing times
1	Integer Factorization Problem (IFP).	Number field sieve: $e^{1.923} (\log n)^{1/3} (\log \log n)^{2/3}$	Sub-exponential
2	Discrete Logarithm Problem (DLS).	Number field sieve: $e^{1.923} (\log n)^{1/3} (\log \log n)^{2/3}$	Sub-exponential
3	Elliptic Curve Discrete Logarithm Problem (ECDLP).	Pollard-rho algorithm: $\sqrt{n}$	Fully exponential

**Table 1.2- Public-key System**

Providing an equivalent level of security with smaller key size is an advantage of ECC compared to RSA. It is very efficient to implement ECC. ECC obtains lower power consumption, and faster computation. It also gains small memory and bandwidth because of its key size length (Dormale, Bulens and Quisquater 2004), (Huang 2007). Such attributes are mainly fascinating in security applications in which calculative power and integrated circuit space are limited. Wireless devices and smart cards present a good example for the constrained devices with limited resources. Cryptography companies such as Certicom Corporation have already implemented ECC in their products for some commercial purposes which are RFID and Zigbee. This company has an agreement with NSA on a set of cryptographic algorithms called suite B. This suite uses Elliptic curves and works over the prime field.

A modular arithmetic performs a main role in public key cryptographic systems (Dormale, et al. 2004). Some of these PKC are the Diffie-Hellman keys exchange

algorithm (Diffie and Hellman 1976), the decipherment operation in the RSA algorithm (Quisquater and Couvreur 1982), the US Government Digital Signature Standard (FIPS 2000), and also elliptic curve cryptography (Koblitz 1987).

Arithmetic in elliptic curves requires a number of modules to calculate ECC operations (modular multiplication, modular division, and modular addition/subtraction operations) (Menezes 1993). The division modular is one of the most critical operations, which is expensive and computationally extensive. Many implementations are completed using projective coordinates in order to represent the points on the curve by reducing inversion/division to one. However, a final division is still needed to convert the projective coordinates into affine coordinates. In some other cases, modular division can be replaced by modular inversion followed by modular multiplication.

In this field, modular multiplication gets much attention and numerous algorithms have been published. The modular inversion can be performed using Fermat little theorem or the well-known extended Euclidian algorithm and Montgomery inverse as well (Kaliski 1995), (Savas and Koc 2000).

## **1.2. Problem statement**

Even though many of ECC implementations have been reported, however, demands for high performance and low power ECC architectures increase. Several implementations use general purpose microprocessor such that presented by (Koschuch, et al. 2006) and (Eberle, et al. 2005)) where their data path and instruction set do not fit the operations on the finite field. Microprocessors, which are the set of their instructions, have also been published and are vulnerable to power analysis attack. Another disadvantage are the co-processor is supplied with its data and a high bandwidth interface is required (Leong and Leung 2002). In order to avoid these disadvantages, a scalar multiplication design is proposed. It is considered as the main module in calculating ECC that performs binary polynomial basis operations in Galois



Field (GF). The design utilizes a pipeline bit-serial for Galois field multiplication. It computes the reduced results based on its irreducible polynomial input. A reduced number of pins can be achieved due to its pipelined bit-serial architecture, low power consumption and low memory capacity. By having these characters, it is suitable to be used in portable devices where the power consumption and memory bandwidth are limited. In addition it has a compact microcoded architecture with its own instruction set. It can improve the reliability without depending on encryption algorithms.

### **1.3.Motivation**

There are many applications using ECC as an authentication for encryption, transactions or signature for secure messaging. For example, ECDSA has been used to sign the product key by Microsoft since Windows 95 (Liu 2007).

Scalar multiplications on elliptic curves are important to be implemented in ECC applications. It is further explained in Chapter 3. Scalar multiplication is used for point calculation over ECC on the operation layer. It is a variant for adding the point number of times to get to the last answer. Scalar multiplication can be computed using two different ways:

- 1- MSB (Most Significant Bit): In this case Double-and-Add Algorithm starts checking the bits starting with the third important bit in the multiplied number.
- 2- LSB (Least Significant Bit): Unlike the previous method, the Algorithm starts calculating from the least important bit.

In this research, the Double-and-Add alternative is used in our system as it is mainly necessary for our algorithms. Galois Field is a finite field that consists of a finite number of elements. It contains three operations which are Addition, Multiplication and division modulars. Galois Field Modular division is replaced by modular inverse followed by modular multiplication. Montgomery modular inversion method is chosen as an inversion algorithm, and Right-to-left shift method as a multiplication algorithm.

Most of the hardware implementations of ECC are based on bit-parallel but in this work bit-serial architecture is used. Bit-serial operators are noticeably smaller than those operators in bit-parallel. They do not depend on word width. A multiplier is said to be bit-serial if it produces only one bit of the product at each clock cycle. Moreover, bit-serial architectures only demand an equally small amount of input and output pins. An implemented multiplication in every bit-serial type has to be directly fitted to the data-width. As a result, the area complexity is reduced to  $O(n)$  and parallel multiplier  $O(n^2)$ .

Bit-serial design makes it compulsory to operate with particular registers. These registers are able to store one bit for the period of a clock cycle. After this cycle the information is passed on to the output and the next information can enter the register. These registers offer the core functionality of shifting numbers.

#### **1.4.Objective of the research**

Our design performs all basic binary polynomial operations in Galois Field (GF) using a microcode structure. It uses a Bit-Serial and pipeline structure for implementing GF operations. Due to its Bit-Serial architecture, it has a low gate count and a reduced number of I/O pins. The proposed design has been implemented in Verilog HDL and Xilinx ISE is used for synthesis and simulation.

Our objectives are

- 1) To analyse a scalar multiplication to calculate Elliptic Curve Cryptography operations using Matlab and Maple codes.
- 2) To design hardware architecture for scalar multiplication using a bit-serial architecture over binary field operation in Galois field.
- 3) To design Elliptic Curve Cryptography Co-processor architecture to work with our scalar multiplication design.

- 4) Examine the obtained result from the Elliptic Curve Cryptography Co-processor with the Matlab results.

## **1.5.Scope of the Research**

The research work includes a scope as listed below:

1. A study on cryptography technologies, which is the basis for the four security services that are authentication, confidentiality, data integrity and non-repudiation.
2. A study on Elliptic Curve Cryptography applications and its standards.
3. A study on algorithms, architectures and implementations of ECC.
4. A study on Binary Galois field and the implementations of Galois Field arithmetic units.
5. A study on bit-serial architectures.
6. A design of bit-serial hardware architecture.
7. The implementation of Scalar Multiplication.
8. A design of ECC Co-processor.
9. The implementation of Elliptic Curve Digital Signature Algorithm (ECDSA) in Matlab.
10. The Testing of Verilog output with Matlab output.

## **1.6. Thesis Organization**

### **Chapter 1- Introduction**

This chapter highlights an overview of the research including the background, motivation, objectives and scope of research.

### **Chapter 2- Cryptography**

This chapter covers the review on cryptography including cryptography components, symmetric-key, Asymmetric-key, their advantages and disadvantages, hashing Algorithm, Hybrid, El-Gemal and RSA.

### **Chapter 3- Elliptic Curve Cryptography**

The third chapter discusses about the introduction of Elliptic Curve Cryptography, Mathematical of ECC, Finite fields, EC Domain parameters, and ECC standards.

### **Chapter 4- ECC algorithms**

This chapter lists all Algorithms that need to be calculated in Galois Field operations. These algorithms are used to architecture design in chapter five.

### **Chapter 5- Design and Implementation and Testing**

This chapter illustrates in detail the designed aspect of Scalar Multiplication. It contains bit-serial architecture design with testing part, implementation of all GF operation, which is needed in scalar multiplication in Verilog HDL, synthesis and simulation that use Xilinx ISE.

### **Chapter 6- Conclusion**

This chapter contains a summary of the research and several suggestions for future research.

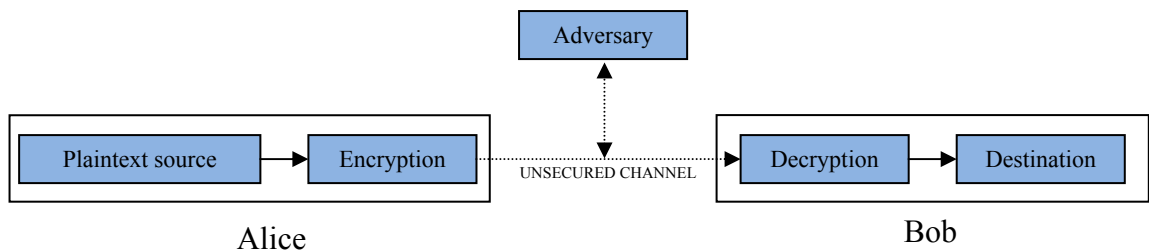
## Chapter 2

# Cryptography

### 2.1.Introduction

Cryptography uses mathematics to encrypt and decrypt data. It enables people to store or transmit sensitive information via insecure network. On the other hand, cryptanalysis is the science of breaking secure communication. There are two persons, Alice and Bob, (the beginning of cryptography: A and B are used as handy abbreviations of the names) communicate via an insecure channel in a secure way. The third person who is eavesdropper (Eve, abbreviated as E) should not be able to read the clearext or change it.

The goal of cryptography is to achieve the aim of allowing two people to exchange messages using cryptography which are not understood by other people (Wang, et al.). Figure 2.1 provides a sample model of a two-party communication using encryption. In this simple party, an entity is a person that sends, receives or manipulates data. *Sender* is an entity that legitimately transmits the information. On the other hand, a *receiver* is an entity that is the recipient of information. A *receiver* may be one of the entities that attempts to crush the information security service provided between the sender and receiver. An adversary plays the role either as the sender or the receiver. The other synonymous names for adversary are attacker, enemy, eavesdropper, opponent and intruder (Jesper 2006).



**Figure 2.1- Two-party communication.**

The cryptographic strength can be measured by the needed resources and time in recovering the plain text. In order to encrypt the plaintext, cryptographic algorithm works in a combination with a key (private key) to resolve the ciphertext. The ciphertext differs from one to another because of different values used in each time. The security of encrypted data depends on the strength of the cryptographic algorithm and the confidentiality of the key (B. Schneier 1996).

## **2.2. Cryptography components**

In order to secure messages, there are mathematical techniques that provide security services such as confidentiality, integrity, authentication and non-repudiation. (Stoneburner 2001) , (Riedel 2003)

**Confidentiality:** data is kept privately in an electronic communication. It is typically provided by encryption. It contains both protections of the transmitted data between two ends. It similarly secures the traffic flow analysis.

**Integrity:** data is not changed in an unauthorized manner. It is typically provided by digital signature and encryption as well.

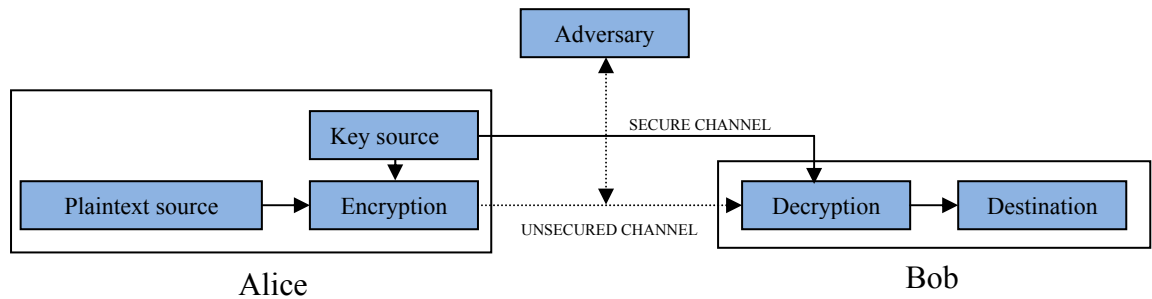
**Authentication:** receiver determines its source to confirm the sender's identity by using something that you have or you know. Normally, it is done by using the sender public key. It is the same integrity provided by digital signature.

**Non-repudiation:** It ensures the sender and receiver from denying the sending or receiving of a message and the authenticity of their signature. Typically, it is provided by digital signature (Al-Kayali 2004).

## **2.3. Symmetric-key**

Symmetric-key is a form of cryptography based on the sharing of a secret key between the parties who want to make communications. It is also called as secret-key. Secret-key is used in the both encryption and decryption process. In this form of cryptography, each party must trust each other and not tell the secret-key to anyone else.

The efficient encryption of large amount of data is the advantage of the symmetric-key, however, the problem appears when key management is over the large number of user needs (Stoneburner 2001) , (Riedel 2003). Figure 2.2 is a sample of the symmetric-key (Al-Kayali 2004) , (Menezes, Oorschot and Vanstone 1996).



**Figure 2.2 - Symmetric-key**

### **2.3.1. Advantages of symmetric-key cryptography**

1. Symmetric-key ciphers can be considered in order to have high rates of data throughput. Some hardware implementations get hundreds of megabytes per second for encryption rate. Software implementations get megabytes per second in the range.
2. Symmetric-key ciphers contain keys that are relatively short.
3. Symmetric-key encryption is noticed to have an extensive history. The knowledge in this area is obtained due to the success of the development of the digital computer, especially the design of the Data Encryption Standard (DES) in the early 1970s.
4. Strong ciphers can be obtained by composing symmetric-key ciphers. Strong product ciphers can be constructed using simple transformations that are easy to analyze.

### **2.3.2. Disadvantages of symmetric-key cryptography**

1. The key must remain secret at both ends in communication.
2. Key management is one of the big problems in large networks.
3. Symmetric-key encryption typically requires a large key for the public verification function in Digital signature mechanisms (Menezes, et al. 1996) .

In 1976, Diffie and Hellman presented Asymmetric-key cryptography which is commonly known as public-key cryptography.

## 2.4. Public-key Cryptography (Asymmetric-key)

In 1976, Diffie and Hellman presented public key cryptography (PKC), which is unlike traditional public-key. Diffie Hellman keys are not used to encrypt or decrypt the message. They are used to create a single shared secret key between the units.

Public key cryptography contains two keys, which are public and private keys. A situation is assumed where Alice wants to send a message to Bob. Alice uses Bob's Public key to encrypt a message and her private key to sign the message. Bob (receiver) uses his Private Key to decrypt the message and he uses Alice's Public Key to verify the signature. The standard bodies have set the key size of the encryption key, in order to provide the desired security. The key size decides the hardship of recovering the encrypted data computationally without the use of the secret key. A scenario of a public key is depicted in Figure 2.3. The key pair  $(e, d)$  is selected by Bob. 'e' is the public key that Bob sends to Alice over any channel but the private key 'd' is kept. Alice encrypts the sending message to Bob using Bob's public-key. Bob decrypts the ciphertext 'c' using the 'd'.

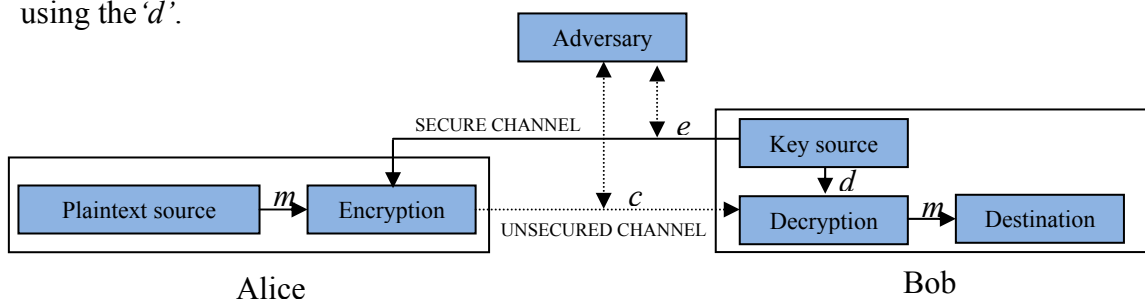


Figure 2.3 - Public-key

### 2.4.1. Advantages of public-key cryptography

1. In public-key, only the private key must be kept secret.
2. The number of keys needed in public-key may be significantly smaller than the symmetric-key in a large network.



3. Relatively, efficient digital signature mechanisms can be obtained by many public-key schemes.
4. The key used in describing the public verification function is smaller in general than in the symmetric-key.
5. A private key and public key remain unchanged for significant periods of time.

#### **2.4.2. Disadvantages of public-key encryption**

1. Data throughput rates of most popular public-key encryption methods are several orders slower than the best-known symmetric-key schemes.
2. Key sizes are much bigger than those used in symmetric-key encryption. The size of the signatures is bigger than what tags can handle. It provides data origin authentication in symmetric-key techniques.
3. There is no fully secure public-key scheme (as well as block ciphers). Most of the current public-key encryption schemes have their security strength. It is based on the presumed difficulty of a set of number-theoretic problems.
4. Public-key cryptography was only discovered in the mid of 1970s. compared with private-key encryption.

#### **2.5. Hybrid**

Symmetric and asymmetric cryptosystems has its advantage and disadvantage. Symmetric is significantly faster than asymmetric but it needs both sender and receiver to share the secret key. A hybrid cryptosystem is a technique of encryption in which combines symmetric and asymmetric encryption to get the advantages. Nowadays, most of the developed schemes use the combination. PKC is used as key exchange algorithms and symmetric is used for encryption purposes.

In the hybrid cryptography, the sender generates a random secret key. Then, the key is encrypted by using the recipient's public key with asymmetric encryption scheme. The sender uses the random secret key which is generated earlier to encrypt the message

(symmetric key). The encrypted secret key and the encrypted message are then sent to the receiver simultaneously. Firstly, the receiver decrypts the secret key using the receiver private key. Then the message is decrypted by using the key (Ihde 2003).

## 2.6. El-Gamal

El-Gamal encryption system was proposed by Taher Elgamal in 1984. It was based on the Diffie-Hellman key agreement which was an asymmetric key encryption algorithm for public-key cryptography (ElGamal 1985). Security of El-Gamal encryption and decryption system are based on the difficulty of discrete logarithm problem.

The El-Gamal encryption system parameters consist of ' $p$ ' and ' $g$ '. ' $p$ ' is a prime number and ' $g$ ' is an integer number. According to  $y=g^a \text{ mod } p$  equation (' $y$ ' is a public key and ' $a$ ' is private key) to generate public key, a private key is used where ' $a$ ' is an integer number between 1 and ' $p-2$ '. A random integer ' $k$ ' is also chosen to encrypt a plaintext message ' $m$ ' which ' $k$ ' should be in interval  $[1, p-2]$ . The message can be text, image or sound but before producing the ciphertext it should be converted to numbers. Ciphertext contains of a pair  $(y_1, y_2)$  which is calculated as following equations:

$$y_1 = g^k \text{ mod } p \qquad y_2 = m y^k \text{ mod } p$$

$m = y_2 / y_1^a \text{ mod } p$  equation is used to decrypt the ciphertext and return plaintext. The length of modulo parameter is affects the security of the ciphertext. On the other hand, the private key does not affect directly on the length of the ciphertext (Mousa 2005).

The El-Gamal encryption system can be used for digital signatures and encryption aim. It is based on a multiplicative group of  $GF(p^m)$ . The El-Gamal encryption system is user friendly because the key generation and encryption/decryption algorithm are effective and simple (CIELNY 2004).

## 2.7. RSA

Over the years, in order to achieve high level of security, key size has grown up to more than 1000 bits. Long key size gives a high level of security, but it requires computing power, storage and bandwidth. So it cannot be used in devices that have limited processing power and battery life. In 1978, Ron Rivest, Adi Shamir and Len Adleman prepared RSA public key cryptography at MIT. It was the most popular public key algorithm over the past twenty years. RSA is an encryption algorithm that is used to encrypt messages, create digital signatures. RSA is based on integer factorization problem.

There is a pair  $(n,e)$  in the RSA public key. ' $n$ ' is the product of two large primes (100 or 200 digit or even longer) and ' $e$ ' is in interval  $[1, n-1]$ . The calculation  $m^x \bmod n$  is known as modular exponentiation. One efficient method to be applied on a computer is the binary left-to-right method. RSA takes time to encrypt and decrypt the message. Thus, ' $n$ ' determines the time.

## 2.8. Hash

Hashing algorithm is used to insure the integrity of the received data and is used to detect any single-bit errors. The one-way hashing algorithm takes variable-length of data even thousands or millions of bits and produces a fixed-length output. The hash function ensures the stability of the data from changing as it may produce a completely different output value (Redmon 2006).

Currently, there are several different hashing algorithms such as LM, MD5, SHA-1, and SHA-2. They can be used for many different aims like digital signatures, data fingerprinting and message authentication codes. The main idea of hashing is to provide confidence and security of the authentic data transmission to user. Cryptographers and hackers always attempt to find subject of each successive algorithm

that is released. If one algorithm is broken, another algorithm is developed and the cycle continues (Redmon 2006).

### **2.8.1. Hash Algorithms**

There are fourteen different types of hash algorithms used today (Wang, et al.). There is a discussion about cryptographic functions of LM (LanManager), MD5, SHA-0/SHA-1, and SHA-2. SHA-0 and SHA-1 are closely familiar and they are commonly approached as a single algorithm (Redmon 2006).

### **2.8.2. LanManager (LM)**

This hashing algorithm was developed by Microsoft in 1990's. It was first time is used in Windows 3.11 product but it was not very secure. LM is also known as LanManager used to protect password on most Windows operating systems. This algorithm works as follows (Jesper 2006):

1. Change the current user's password to uppercase.
2. Truncate to 14 bytes/characters or null-pad the password.
3. Take the resulting password and split it into two halves, 7 bytes each.
4. Create two DES keys by using the two 7-byte halves as inputs.
5. These keys are used to DES-encrypt the ASCII string "KGS!@#\$\$%" and produce two 8-byte ciphertext values.
6. Link the two resulted ciphertext values which will form a 16-byte value – this is the LM Hash.

### **2.8.3. Message Digest algorithm 5 (MD5)**

MD5 hashing algorithm was created in 1991 by Ronald Rivest (an MIT professor), as a successor to MD4 (Jesper 2006) , (B. Schneier 1996). MD5 starts by adding the entering data to result in a file length. Then, it takes the data in 512-bit blocks, each of these blocks is divided into sixteen 32-bit sub-blocks. Next, each of these sub-blocks is used to affect the 128-bit state variable. It is used in the algorithm in

which there are four 32-bit variables such as 'a', 'b', 'c' and 'd'. These variables go through four different nonlinear functions. The fourth step (the final state) (a', b', c', d') is added to the original entries in this step of the algorithm. After processing all the data, the 128-bit state variable remains.

#### 2.8.4. SHA-0/SHA-1

There are many similarities between SHA-0 and SHA-1 algorithms and MD5 operation (B. Schneier 1996). The input data is divided into sixteen 32-bit sub-blocks. However, in SHA-0/SHA-1 algorithms, there is a 160-bit state variable. They are five 32-bit variables such as 'a', 'b', 'c' and 'd'. The algorithm has four steps. Each step has 20 operations. The used data is state variable and the entered data is input. At the end of processing the incoming data, 160-bit state variable remains.

#### 2.8.5. SHA-2

SHA-2 modifications include SHA-224, SHA-256, SHA-384, and SHA-512. The entering data is divisible by 512-bits. The data is divided either into 32-bit words (SHA-224/256) or 64-bit words (SHA-384/512). SHA-2 output variants are specified by their suffix (SHA-512 has a 512-bit digest as its output) (Markoff 2009).

**Table 2.1-Hashing Algorithm.**

Hashing algorithm	Detail
LM	The First weakness is the hash is insensitive-case. Secondly, the character set is limited to 142 characters. Finally, the hash is broken down into two 7-byte. The best security for hashing algorithm is by keeping out the hash result from unauthorized people.
MD5	MD5 is more secure than LM but it is still vulnerable to compromise (B. Schneier 1996). The key vulnerability for MD5 lies in the length of its final hash.
SHA-0/SHA-1	There is the same security result for SHA-0/SHA-1's with 160-bit hash and MD5's 128-bit hash. Some SHA-1 vulnerabilities have been found.
SHA-2	SHA-2's security is not acknowledged as its variants have passed the cryptographic community analysis.

SHA-0/SHA-1, SHA-2 and MD5 are of the Merkle-Damgård construction. These hash functions can accept an incoming data of arbitrary-length in which resulting prescribed output length (Al-Kayali 2004).

### **2.8.6. Uses of Hashing**

Hash algorithms were used to detect errors in a file's transmission. However, presently Hash algorithms have more functions. Therefore, it needs to be more secure. One of its abilities is to reduce the data to a manageable signature element in digital signature. After reducing the data, signature element is sent to authority such as RSA or ECC to be digitally signed. (Bellovin and Rescorla 2005)

Other ability is Message Authentication Codes (MAC) which is one-way hash algorithm with the addition of a secret key (B. Schneier 1996). Secret key is the only way to verify a MAC value and the hashing. The use of the key happens in multiple times in some MAC algorithm (Bellovin and Rescorla 2005).

Hashing is as a Pseudo-random function. By inputting data through a hashing algorithm, it results in random-seeming bits. Use Diffie-Hellman to exchanges these random-seeming bits to be used later for generating cipher keying material (Bellovin and Rescorla 2005).

Data fingerprinting is the fourth use of hashing and probably the most common use of hashing. The output identifier can be used to verify integrity of file or a message. Even there is a slight modification in the file; the resulting hash of the file is totally different. Hashing algorithm input data can change up to 50% of the produced hash even if it is single bit.

The modern study of symmetric-key ciphers relates mainly to the study of block ciphers and stream ciphers. Lucifer is the first block cipher that is developed at IBM by (Feistel 1971). A revised version of the algorithm was adopted by FIPS which is known

now by Data Encryption Standard (DES). DES is used to across a wide range of applications such ATM and secure remote access. National Institute of Standards and Technology (NIST) in 2001 adopted Advanced Encryption Standard (AES) (Daemen and Rijmen 2002). Many researches were conducted to strengthen the block cipher algorithms such as triple-DES which is defined by many standard bodies such ANSI, FIPS and NIST. Also, IDEA (International Data Encryption Algorithm) (Massey, et al. 1992) which developed in 1990 and it uses 128bit keys is a commercial symmetric cipher developed in 1990 which uses 128-bit keys. Another symmetric algorithm is Blowfish; it is a symmetric cipher with a variable key length from 32 to 448 bits (Schneier 1993).

Symmetric-key encryption uses the same key for encryption and decryption. The advantage in this scheme is the key size is small and it is fast. However, symmetric encryption suffers from key management which necessary to be secure. Each different pair of communicating parties must, ideally, share a different key. The number of the keys increases as the square of the number network members.

Numerous of researches were conducted to solve this disadvantage resulting in the first key exchange algorithm proposed by Whitfield Diffie and Martin Hellman authors of the first paper on public-key cryptography. Later on in 1978 RSA (Rivest, Shamir and Adleman) was described at MIT (Massachusetts Institute of Technology). RSA is much slower than DES and AES and it is uses much bigger key size to meet the security level.

RSA is the dominant public key cryptography the last 20 years providing high security level. One of RSA disadvantage is the key size, it is too big to be implemented on small devices such as RFID. That is because of the key size and the extensive computation power that is needed to finish RSA computations. In 1985 Miller and Koblitz independently presented ECC which based on algebraic structure of Elliptic

curve. Currently ECC is the most efficient public key cryptosystem that uses shorter keys while providing the same security level as the RSA. Several researches were conducted on implementing public key cryptography especially ECC in Embedded devices. ECC can be used on RFID as anti-counterfeiting; the fact that ECC is a fitting technology for RFIDs was concluded in the work of (Wolkerstorfer 2005). Also, many other researches were conducted to study the possibility of implementing PKC on constrained devices such as WSN and Smart cards. A research was conducted (Gura, et al. 2004) to compare ECC and RSA on 8-bit CPU, this work show how ECC outperforms RSA. To meet the security requirement, RSA need to work on keys not less than 1024bit, which made it difficult to implement RSA on embedded applications. Compared to RSA key size, ECC provides the same security level with smaller key size. For instance 160bit key size in ECC provides the same security level that is RSA provides with 1024bit.

## **Conclusion**

This chapter gives a brief introduction to cryptography and its components. We discussed the symmetric-key and the asymmetric-key cryptography by showing their advantages and disadvantages. The public key cryptography algorithms such as Hybrid, El-Gamal and RSA are discussed and they give some information of how they work. There is also a brief introduction to the hashing algorithms i.e. SHA-1 and MD5. In the next chapter, Elliptic Curve Cryptography mathematical and some of its protocols are discussed.



## Chapter 3

### Elliptic Curve Cryptography

#### 3.1. Introduction

Elliptic curve cryptography (ECC) was proposed in 1985 by Neal Koblitz and Victor Miller. Elliptic curve cryptographic schemes can provide the same functionality as RSA schemes which are public-key mechanisms. The security is based on the difficulty of a different problem, which is called the Elliptic Curve Discrete Logarithm Problem (ECDLP). In order to solve the ECDLP, the best algorithms have fully exponential time. In contrast, the integer factorization problem has to be solved with subexponential-time algorithms (Hankerson, et al. 2004). It makes Elliptic Curve Cryptography offers similar security. It is offered by other traditional public key cryptography schemes used nowadays, with smaller key sizes and memory requirements. (As shown in Table 3.1) (Kumar 2006). For example, it is generally accepted that a 1024-bit RSA key provides the same level of security as a 160-bit elliptic curve key. The advantages can be achieved from smaller key sizes including storage, speed and efficient use of power and bandwidth. The use of shorter keys means lower space requirements for key storage and quicker arithmetic operations. These advantages are essential when public-key cryptography is applied in constrained devices, such as in mobile devices or RFID. These advantages are the reason behind choosing ECC as the cryptography system in this thesis.

**Table 3.1- NIST Recommended Key Sizes.**

Symmetric-key	ECC	RSA	Comment
64 bit	128 bit	700 bit	Short period security
80 bit	160 bit	1024 bit	Medium period Security
128 bit	256 bit	2048	Long period Security

In brief, ECC based algorithms can be easily included into existing protocols to get the same backward compatibility and security with smaller resources. Therefore, more low-end controlled devices can use such protocols to be considered unsuitable for such systems.

A group structure used to implement the cryptographic schemes is provided by using Elliptic curves and is determined over a finite field. The elements of the group are the points on the elliptic curve. They act as the identity element of the group. On the other hand the group operation can be executed by arithmetic operations based on finite field. It is discussed in detail in the next section (Kumar 2006).

### **3.2. Mathematical of Elliptic Curve Cryptography**

There are many ways to calculate the points over the prime field elliptic curve. A direct method is by applying the next equation (Tata,. 2007)

$$y^2=x^3+ax+b \text{ where } 4a^3+27b^2 \neq 0$$

Different elliptic curve is produced by changing the values of 'a' and 'b'. In elliptic curve cryptography, calculating the public-key can be done by multiplying the private key with the generator point 'G' in the curve. The generator point 'G' is the point on the curve. The private key is the random number in the interval [1, n-1], 'n' is the curve's order (Tata, 2007).

The strength of ECC security comes from the difficulty of Elliptic Curve Discrete Logarithm Problem. If 'P' and 'Q' are points on the curve, then  $kP=Q$  where 'k' is a scalar. Thus, point multiplication is the basic operation in ECC. For example, the multiplication of a scalar 'k' with any point 'P' on the curve in order to obtain another point 'Q' on the curve.

#### **3.2.1. Point Multiplication**

Scalar point multiplication is a block of all elliptic curve cryptosystems. It is an operation of the form  $k.P$ . 'P' is a point on the elliptic curve and 'k' is a positive integer.

Computing  $k.P$  means adding the point 'P' exactly  $d-1$  times to itself, which results in another point 'Q' on the elliptic curve. Point multiplication uses two basic elliptic curve operations:

- 1- point addition (add two point to find another point)
- 2- point doubling (adding point p to itself to find another point)

For example to calculate  $kP=Q$  if 'K' is 23 then  $kP=23P=2(2(2(2P) + P) + P) + P$  so to get the result point addition and point doubling is used repeatedly (Tata, 2007).

### 3.2.2. Point Addition

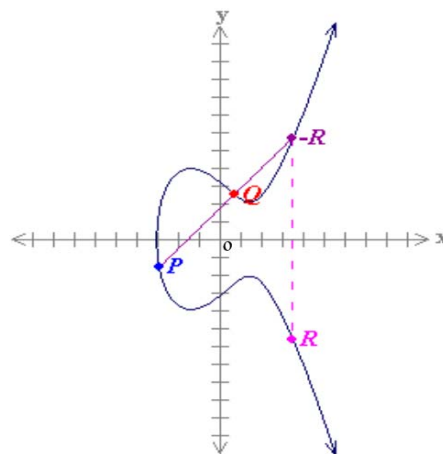
The resulted point of adding two different points on the elliptic curve is computed as shown below (Al-Kayali 2004).

$$(X_1, Y_1) + (X_2, Y_2) = (X_3, Y_3) ; \text{where } X_1 \neq X_2$$

$$\lambda = (Y_2 - Y_1)/(X_2 - X_1)$$

$$X_3 = \lambda^2 - X_1 - X_2$$

$$Y_3 = \lambda(X_1 - X_3) - Y_1$$



**Figure 3.1- Point Addition**

Figure 3.1 illustrates how to add two points graphically on the elliptic curve. Assume that P and Q are two different points on the elliptic curve E. Since we are crossing a line with a cubic curve, the line that connecting P and Q must intersect through a third point on the curve; the point is noted as -R. another point called R will be resolved from the reflection of this point -R in the x-axis, where  $R = P+Q$ .

### 3.2.3. Point Doubling

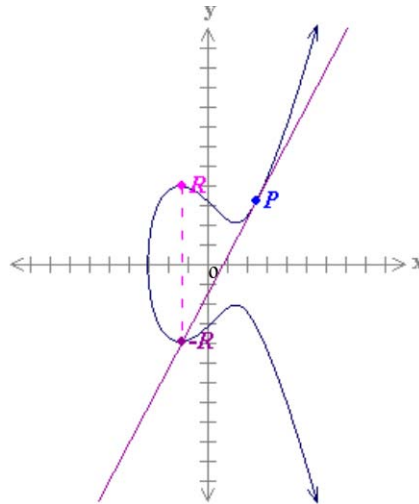
Adding a point to itself (doubling a point) on the elliptic curve can be computed as shown below (Al-Kayali 2004):

$$(X_1, Y_1) + (X_2, Y_2) = (X_3, Y_3) ; \text{where } X_1 \neq 0$$

$$\lambda = (3(X_1)^2 + a)/(2Y_1)$$

$$X_3 = \lambda^2 - 2X_1$$

$$Y_3 = \lambda(X_1 - X_3) - Y_1$$



**Figure 3.2- Point Doubling**

Figure 3.2 shows how a point can be doubled graphically on the elliptic curve. Suppose we want to double a point P on the elliptic curve. A tangent line to the curve and passing by P is taken to double the point. The line must cross the curve through another point; the point is noted as -R. Then we reflect the point -R in the x-axis to the point R where  $R=2P$ .

## 3.3. Finite Fields

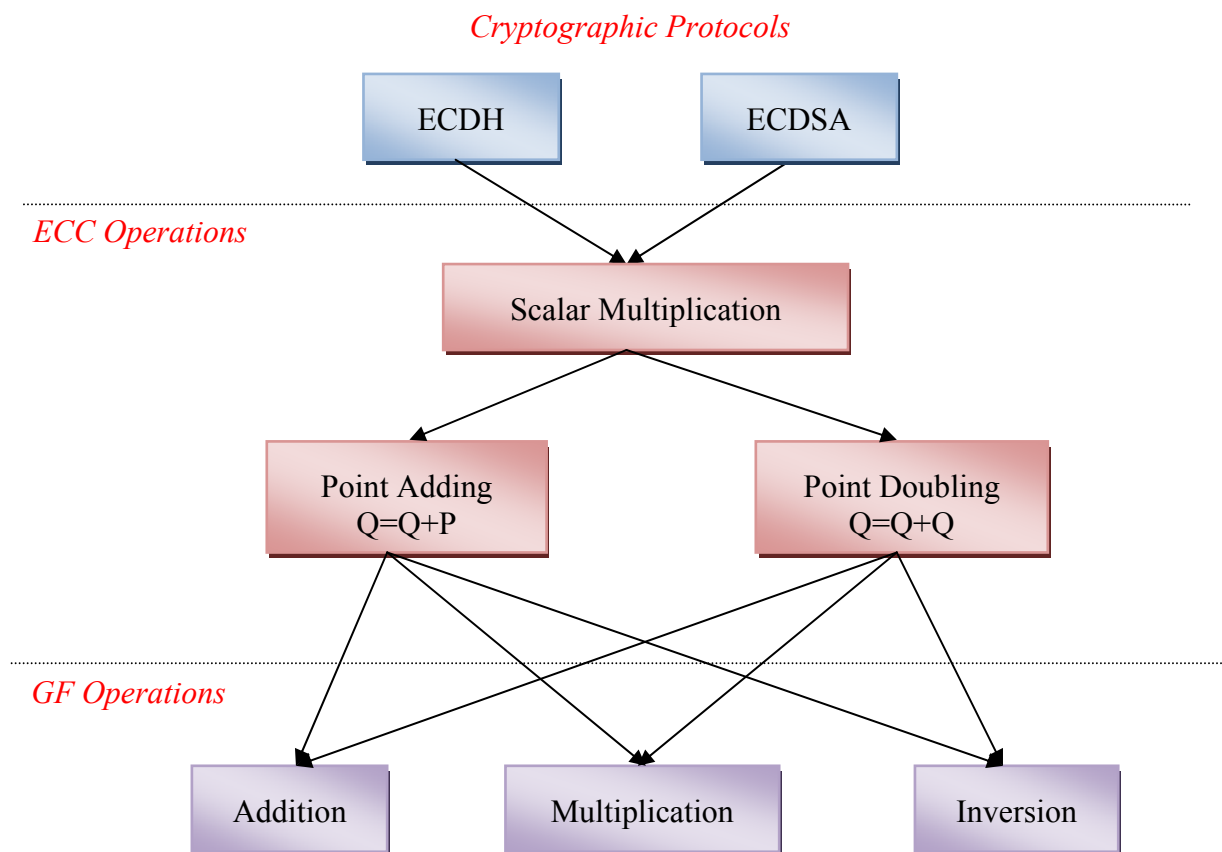
### 3.3.1. Introduction

All elliptic curve operations mentioned earlier are based on real numbers. However, operations over the real numbers are inaccurate and slow, whereas cryptographic operations need to be accurate and fast. Therefore, the curve

cryptology can be defined over finite fields to operate EC efficiently and accurately.

A finite field is a set of a finite number of elements.

Figure 3.3 describes elliptic curve operation hierarchy. It consists of cryptographic protocols, Elliptic Curve point operations and basic Galois Field operations.



**Figure 3.3- Hierarchy of Elliptic Curve.**

ECC hierarchy consists of three levels. The highest level is cryptography protocols. ECDH and ECDSA are two categories of cryptography protocol that are used to provide services like Public-key, signature and key agreement. Elliptic Curve point operation level consists of Scalar Multiplication, in which utilizes point adding and point doubling. In the last level, the operations are GF addition, GF multiplication, GF squaring and GF inversion.

The rest of this part explains EC operations on finite fields. The operations are defined on affine coordinate system. Affine coordinate system is a normal coordinate system that represents each point by the vector  $(x, y)$ .

### 3.3.2. EC on Prime field $F_p$

The equation to retrieve all the possible points on the curve over the prime field is  $y^2 \bmod p = x^3 + ax^2 + b \bmod p$ , where  $4a^3 + 27b^2 \bmod p \neq 0$ . The elements of the finite field are integers between 0 and  $p - 1$  thus all the operations such as addition, subtraction, multiplication, division include integers between 0 and  $p - 1$ . The prime number ' $p$ ' is the finitely large number of points on the EC in order to make the cryptosystem secure (Tata, 2007).

#### 3.3.2.1. Point Addition

Let ' $J$ ' and ' $K$ ' be the two points on the curve which  $J = (x_J, y_J)$  and  $K = (x_K, y_K)$  and  $L=K+J$  where  $L=(x_L, y_L)$  and ' $s$ ' is the incline of the line through ' $J$ ' and ' $K$ ' then:

$$S = (y_J - y_K) / (x_J - x_K) \bmod p,$$

$$x_L = s^2 - x_J - x_K \bmod p$$

$$y_L = -y_J + s(x_J - x_L) \bmod p$$

So if  $K=-J$  for example  $K = (x_J, -y_J \bmod p)$  then  $J+K=O$ , where ' $O$ ' is the point at infinity. If  $K=J$ , then  $J+K=2J$  needs to use point doubling equation too  $J + K = K + J$ .

#### 3.3.2.2. Point Subtraction

If ' $J$ ' and ' $K$ ' are two points on the curve which  $J = (x_J, y_J)$  and  $K = (x_K, y_K)$  Then:

$$J - K = J + (-K) \text{ where } -K = (x_k, -y_k \bmod p)$$

In certain implementation of point multiplication such as NAF point subtraction is used (Stoneburner 2001) & (Riedel 2003).

### 3.3.2.3. Point Doubling:

Assume  $J=(x_J, y_J)$  and  $y_J \neq 0$  so the calculating of  $L= 2J$  where  $L = (x_L, y_L)$  is:

$s = (3x_J^2 + a) / (2y_J) \bmod p$  ('S' is the tangent at point 'J' and 'a' is one of the parameters selected with the EC)

$$x_L = s^2 - 2x_J \bmod p$$

$$y_L = -y_J + s(x_J - x_L) \bmod p$$

And 'O' is the point at infinity if  $y_J=0$  then  $2J= O$ .

### 3.3.3. Elliptic Curve (EC) on Binary field $F_{2^m}$

The equation to retrieve all the possible points on the curve over the binary field is  $F_{2^m}$  is  $y^2 + xy = x^3 + ax^2 + b$ , where  $b \neq 0$ . The elements of the finite field are integers of length at most 'm' bits, which they can be considered as a binary polynomial of degree  $m - 1$ . On the other hand, in binary polynomial the coefficients can only be 0 or 1. Thus, all the operations such as addition, subtraction, multiplication and division include polynomials of degree  $m - 1$  or lesser. The m is the finitely large number of points on the EC to make the cryptosystem secure (Tata, 2007).

#### 3.3.3.1. Point Addition

'J' and 'K' are two points on the curve which  $J = (x_J, y_J)$  and  $K = (x_K, y_K)$  and  $L=K+J$  where  $L=(x_L, y_L)$  and s is the incline of the line through 'J' and 'K' then:

$$s = (y_J + y_K)/(x_J + x_K)$$

$$x_L = s^2 + s + x_J + x_K + a$$

$$y_L = s(x_J + x_L) + x_L + y_J$$

So if  $K=-J$  for example  $K = (x_J, x_J+y_J)$  then  $J+K= O$ , where 'O' is the point at infinity. And if  $K=J$  then  $J+K=2J$ . It needs to use point doubling equation also  $J + K = K + J$ .

### 3.3.3.2. Point Subtraction

If 'J' and 'K' are two points on the curve which  $J = (x_J, y_J)$  and  $K = (x_K, y_K)$

Then:

$$J - K = J + (-K) \text{ where } -K = (x_k, x_k + y_k)$$

In certain implementation of point multiplication such as NAF, point subtraction is used (Stoneburner 2001) & (Riedel 2003).

### 3.3.3.3. Point Doubling

Assume  $J=(x_J, y_J)$  and  $y_J \neq 0$  so calculating of  $L= 2J$  where  $L = (x_L, y_L)$  is:

$$s = x_J + y_J / x_J, \text{ ('S' is the tangent at point 'J')}$$

$$x_L = s^2 + s + a \text{ ('a' is one of the parameters selected with the EC)}$$

$$y_L = x_J^2 + (s + 1) * x_L$$

And 'O' is the point at infinity if  $x_J=0$  then  $2J= O$ .

Galois Field  $GF(2^m)$  multiplier and inversion part are resumed in the chapter 4 using its algorithm.

## 3.4. Elliptic Curve Domain parameters

### 3.4.1. Domain parameters for EC over field $F_p$

Elliptic curve over  $F_p$  has list of domain parameters which includes 'p', 'a', 'b', 'G', 'n' and 'h' parameters.

'a' and 'b': define the curve  $y^2 \text{ mod } p = x^3 + ax + b \text{ mod } p$ .

'p': prime number defined for finite field  $F_p$

'G': generator point  $(X_G, Y_G)$  on the EC that selected for cryptography operations.

'n': The Elliptic curve order.

'h': if  $\#E(F_p)$  is the number of points on an elliptic curve then 'h' is cofactor where  $h = \#E(F_p) / n$ .



### 3.4.2. Domain parameters for EC binary fields

Elliptic curve over  $F_{2^m}$  has a list of domain parameters which includes 'm',  $f(x)$ , 'a', 'b', 'G', 'n' and 'h' parameters.

'm': an integer to finite field  $F_2^m$ .

$F(x)$ : the irreducible polynomial of degree m that it used for elliptic curve operations.

'a' and 'b': define the curves  $y^2 + xy = x^3 + ax^2 + b$ .

'G': the generator point  $(x_G, y_G)$  on the EC that selected for cryptography operations.

'n': the order of the elliptic curve.

'h': if  $\#E(F_{2^m})$  is the number of points on an elliptic curve then 'h' is cofactor where  $h = \#E(F_{2^m})/n$ .

### 3.5. Field Arithmetic

Modular arithmetic and polynomial arithmetic are two different types applied in ECC operations depending on the chosen field. In Modular arithmetic, over a number 'p' arithmetic covers the number in the interval  $[0$  and  $p - 1]$ . Modular Arithmetic contains Addition, Subtraction, Multiplication, Division, Multiplication Inverse and Finding  $x \bmod y$  operations. Polynomial Arithmetic contains Addition, Subtraction, Multiplication, Division, Multiplicative Inverse and Irreducible Polynomial (Tata, 2007). Polynomial arithmetic plays an important role in a number of areas of engineering and software verification. Especially, polynomial limitation solving has a long and successful history in the developing tools to provide termination of programs. (Borralleras, et al. 2009) This research relies on polynomial arithmetic. Therefore, this part gives an overview of polynomial arithmetic.

EC over field  $F_{2^m}$  includes arithmetic of integer with length  $m$  bits. The binary string can be declared as polynomial:

Binary String:  $(a_{m-1} \dots a_1 a_0)$

Polynomial:  $a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_2x^2 + a_1x + a_0$  where  $a_i = 0$ .

For example:  $x^3 + x^2 + 1$  is polynomial for a four bit number  $1101_2$ .

### 3.5.1. Addition

If  $A = x^3 + x^2 + 1$  and  $B = x^2 + x$  are two polynomial then  $A+B$  called polynomial addition that returns  $x^3 + 2x^2 + x + 1$  after taking *mod 2* over coefficients  $A + B = x^3 + x + 1$ .

On binary representation, polynomial addition can be achieved by simple XOR of two numbers. For example, over  $GF(2^4)$  there are 16 elements where  $f(x) = x^4 + x + 1$  as follow:

0 (0000)	1 (0001)	$x$ (0010)
$x + 1$ (0011)	$x^2$ (0100)	$x^2 + 1$ (0101)
$x^2 + x$ (0110)	$x^2 + x + 1$ (0111)	$x^3$ (1000)
$x^3 + 1$ (1001)	$x^3 + x$ (1010)	$x^3 + x + 1$ (1011)
$x^3 + x^2$ (1100)	$x^3 + x^2 + 1$ (1101)	$x^3 + x^2 + x$ (1110)
$x^3 + x^2 + x + 1$ (1111)		

So if:  $A = 1101_2$   
 $B = 0110_2$  }  $A+B=A \text{ XOR } B \rightarrow A+B=1011_2$

### 3.5.2. Subtraction

If  $A = x^3 + x^2 + 1$  and  $B = x^2 + x$  are two polynomials, then  $A-B$  is called polynomial subtraction that returns  $x^3 - x + 1$  after taking *mod 2* over coefficients  $A - B = x^3 + x + 1$ .

On binary representation, polynomial addition can be achieved by a simple XOR of two numbers same as Addition operation in  $F_{2^m}$ :

$A = 1101_2$   
 $B = 0110_2$  }  $A-B=A \text{ XOR } B \rightarrow A+B=1011_2$

### 3.5.3. Multiplication

If  $A = x^3 + x^2 + 1$  and  $B = x^2 + x$  are two polynomials then  $A*B$  is called polynomial multiplication that returns  $x^5+x^3 + x^2 + x$ ,  $m=4$ . The result should be reduced to a degree less than 4 by irreducible polynomial  $x^4 + x + 1$ .

$$\begin{aligned} x^5 + x^3 + x^2 + x \pmod{f(x)} &= (x^4 + x + 1)x + x^5 + x^3 + x^2 + x \\ &= 2x^5 + x^3 + 2x^2 + 2x = x^3 \pmod{2} \end{aligned}$$

$$A = 1101_2$$

$$B = 0110_2$$

$$A * B = 1000_2$$

### 3.5.4. Division

$a * b^{-1} \pmod{f(x)}$  has the same result of  $a/b \pmod{f(x)}$ . So in order to find  $a/b \pmod{f(x)}$ ,  $a * b^{-1} \pmod{f(x)}$  can be used. Instead of it,  $b^{-1}$  is the multiplicative inverse of 'b'.

### 3.5.5. Multiplicative Inversion

There are 16 powers for  $g$  where the element  $g = (0010)$  is a generator:

$$\begin{array}{cccc} g^0 = (0001) & g^1 = (0010) & g^2 = (0100) & g^3 = (1000) \\ g^4 = (0011) & g^5 = (0110) & g^6 = (1100) & g^7 = (1011) \\ g^8 = (0101) & g^9 = (1010) & g^{10} = (0111) & g^{11} = (1110) \\ g^{12} = (1111) & g^{13} = (1101) & g^{14} = (1001) & g^{15} = (0001). \end{array}$$

The multiplicative identity is  $g^0 = (0001)$  and the multiplicative inverse for  $g^9 = (1010)$  is:

$$g^9 = (1010) \text{ is } g^{-9} \pmod{15} = g^6 \pmod{15}$$

To assure that  $g^6$  is the multiplicative inverse of  $g^9$  their multiplication result should be one.

$$\text{Proofing that } g^6 \times g^9 \equiv 1 \pmod{f(x)}$$

$$g^6 \cdot g^9 = (1010) \cdot (1100)$$

$$(x^3 + x) \cdot (x^3 + x^2) \bmod f(x)$$

$$(x^6 + x^5 + x^4 + x^3) \bmod f(x) = 1 \text{ (which is the multiplicative identity)}$$

This means  $g^{-9} \equiv g^6 \bmod f(x)$

### 3.6. Elliptic Curve Cryptography Standards

There are some different types of standards for ECC. The table below lists the different types of ECC standards:

**Table 3.2- Standards**

Standard	Schemes included
<b>ANSI X9.62</b>	ECDSA
<b>ANSI X9.63</b>	ECIES, ECDH, ECMQV
<b>FIPS 186-2</b>	ECDSA
<b>IEEE P1363</b>	ECDSA, ECDH, ECMQV
<b>IEEE P1363A</b>	ECIES
<b>IPSec</b>	ECDSA, ECDH
<b>ISO 14888-3</b>	ECDSA
<b>ISO 15946</b>	ECDSA, ECDH, ECMQV

ANSI X9.62 standard gives a detailed description about ECDSA. It is supported with examples on both fields (prime and binary) for different key sizes. It recommends ECDSA to be used with the hash algorithm such as SHA-1. It also meets the stringent security requirements with elliptic curve domain parameters with  $n > 2^{160}$  (ANSIX9.62 1999)

The ANSI X9.63 standard indicates key transport schemes and key agreement based on Elliptic Curve Cryptography. On the other hand, the standard specifies various schemes made of ECDH, ECIES and ECMQV.

ANSI X9.63 standard is like ANSI X9.62. It puts various constraints like limitation to the hash algorithm SHA-1. It also needs elliptic curve domain parameters with  $n > 2^{160}$ . Additionally ANSI X9.63 uses an ANSI-approved pseudorandom number

generator. The other issue about ANSI X9.63 is the key transport of short keys may support TDES symmetric encryption.

FIPS 186-2 standard specifies signature schemes (FIPS 2000). It specifies ECDSA as specified in ANSI X9.62.

The IEEE P1363 standard based on the three parameters has a wide scope encompassing schemes (P1363A 2000). The traditional discrete logarithm problem, integer factorization Problem and the ECDLP are the three parameters. IEEE P1363 standard includes general descriptions of ECDSA that known as ECSSA. ECDH is known as ECKAS-DH1 and ECMQV is also known as ECKAS-MQV.

IEEE P1363A is similar with IEEE P1363 but it contains additional techniques. ECIES is included in IEEE P1363A (P1363A 2000). The IPsec standard contains support for a variant of ECDH. The default elliptic curve domain parameters in IPsec are limited over  $F_{2^m}$ .

The ISO 14888-3 standard identifies very general signature mechanisms. The draft ISO 15946 standards indicate cryptographic techniques based on ECC (ISO/IEC14888-3 n.d.). ISO 15946-2 identifies a variety of signature schemes including ECDSA (ISO/IEC15946 2002). On the other hand ISO 15946-3 describes a variety of key establishment schemes including some based on ECMQV and ECDH.

The FSTC FSML standard in signing electronic checks and other electronic financial documents use ECDSA as specified in ANSI X9.62. The PKIX standard within the PKIX certification framework describes how ECDSA as specified in ANSI X9.62 can be used. The S/MIME standard specifies how to use ECDH, ECDSA and ECMQV as indicated in ANSI X9.62 and ANSI X9.63 to secure email. The SSL/TLS ECC standard indicate how to use ECDSA, ECDH, and ECMQV as indicated in ANSI X9.62, ANSI X9.63, and IEEE P1363 in SSL/TLS. The WAP WTLS standard allows

the use of ECDSA and ECDH as indicated in IEEE P1363. The WAP WMLScript Crypto API standard uses of ECDSA as indicated in IEEE P1363.

### 3.7. Elliptic Curve Diffie-Hellman

Symmetric-key cryptosystems are usually used for encryption/decryption, because it is faster than public-key cryptosystems. Before the cryptographic process starts, Symmetric-key cryptosystems needs the entities to agree beforehand on a secret key. This agreement can be presented by Diffie-Hellman key exchange. The Elliptic Curve Diffie-Hellman (ECDH) key exchange method is described by the following example (Tata, 2007) & (Huang 2007).

As shown in Figure 3.4, two communicating parties which are Alice and Bob, agree on an elliptic curve 'E' over finite field  $F_q$ . They also agree on a point  $P \in E(F_q)$ . Alice chooses a secret integer 'k', computes  $A = g^k \text{ mod } p$  and sends it to Bob. Bob does the same. He chooses a secret integer l, computes  $B = g^l \text{ mod } p$ , and sends it to Alice. She computes  $K_{ab} = B^k \text{ mod } p$ . At the other part Bob computes  $K_{ab} = A^l \text{ mod } p$ . After those computations, there is the same result for both parts ( $K_{ab} = K_{ba}$ ).

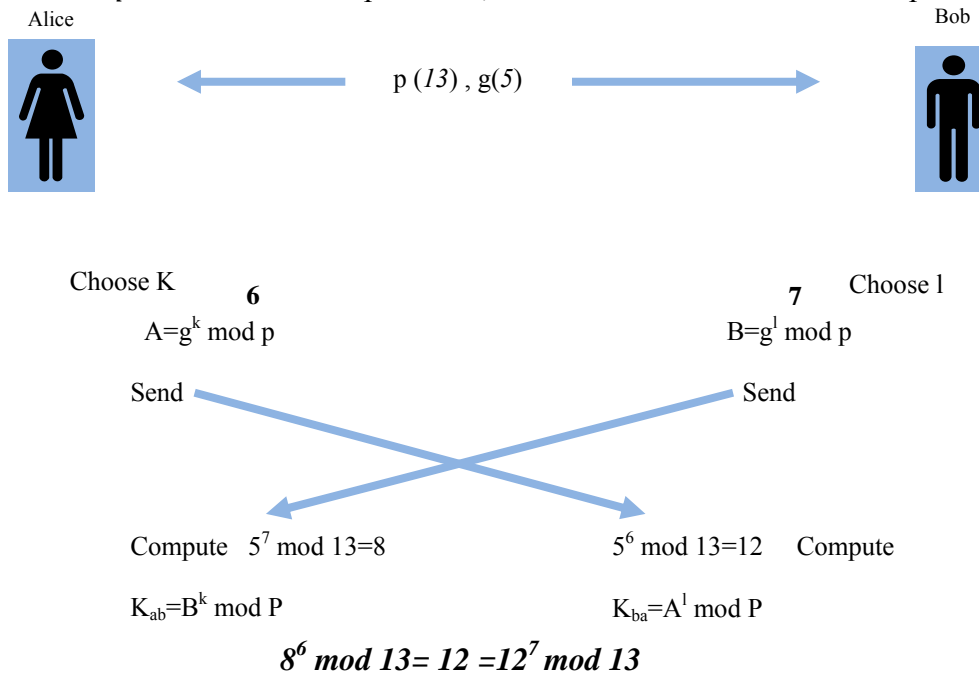


Figure 3.4- (ECDH) key exchange method.

### **3.8.Applications of ECC**

Pietiläinen illustrated that most of the standards are written in an algorithm independently (Pietiläinen 2000). Consequently any known public key algorithm can be implemented. It allows the implementing of ECC in applications in which other PKC is not practical.

#### **3.8.1. Radio Frequency Identification (RFID-Tags)**

Now, tags become essential applications. They are used to categorize goods or give an access for an individual to categorize books in libraries and in other areas. An RFID system consists of a wireless reader device to communicate with an RFID tag. These tags are important in helping to identify counterfeit products out of the original. The issue raised is the possibility of copying these tags. By eavesdropping on the channel between the chip and the reader, the counterfeiter can create another chip with the same information of the original hold. It means that there is no protection on data. Therefore the location privacy is not protected too.

Protection of data privacy means that the data transmission from RFID is encrypted in a way that unauthorized reader cannot understand it. Location privacy means the transmitted data by tags cannot be used to track the tag. Tags are constrained resources (power, area, memory). The challenge is the formation in making these communications between the chip and the reader secure.

“Unfortunately asymmetric cryptography is too heavy to be implemented on a tag" was stated in (Avoine, Dysli and Oechslin 2005). Where (Tuyls and Batina 2006) illustrated that the use of public key cryptography is possible on RFID-tags.

There is practical comparison in terms of chip area, power consumption, and clock cycles. The comparison is on the implementations of some standardized cryptographic algorithms (e.g., SHA-1, AES-128, and ECC-192) and passive RFID tags demonstrated by Feldhofer & Wolkerstorfer in 2007.

The systems that use symmetric protocol for authentication require the security of the secret key stored in the reader. The connections to the database also need to be secured in order, to prevent the system from begin compromised. This protection is very expensive to be provided. An additional  $GF(p)$  hardware is necessary to process  $GF(p)$  operations which is needed in RFID tags. L. Batina, J. Guajardo, T. Kerins, N. Mentens, Tuyls, and Verbauwhede, (2006) proposed a protocol to implement ECC over binary fields. Their design is secured against passive attacks, but it is vulnerable against active attacks.

Braun, et al. (2008) claimed that their scheme offers advantages in a large decentralized system over the previous solution using Elliptic Curve Cryptosystem.

Certicom collaborates with Texas Instruments for RFID Authentication and Encryption. Their collaboration is to prevent the counterfeiting of the medical supply. It is an important case in which the fake product may put the person's life in danger. They use centralized approach where the tag readers need to have online connection to the centralized database. The signature used is Elliptic Curve Pintsov-Vanstone Signature (ECPVS), which conceals the product class Id and form the unauthorized readers. ECPVS is six times smaller than RSA signature. It is three times faster and has half of the size of Elliptic curves Digital Signature Algorithm (ECDSA).

### **3.8.2. Elliptic Curve Digital Signature Algorithm**

Digital Signature Algorithm (DSA) is used for authenticating a message or an application in electronic communication. Any signature has its own property which is produced by only a single individual that has the private key. However, it can be verified by anyone who receives the message. If Alice (*sender*) sends a message to Bob (*receiver*), Alice should sign the message using her private key to authenticate the message. The signature can be verified using the Alice's public key. Therefore, if Bob



knows Alice's public key, he would be able to verify the signature (Tata, 2007) & (Huang 2007).

Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of the DSA that operates on elliptic curve. In the previous example, when Alice sends the signed message to Bob, both have to agree on EC domain parameters. Alice has key pair consisting of a private key ' $d_A$ ' and public key  $Q_A = d_A * G$ . ' $d_A$ ' is the random number that it is less than ' $n$ ' (' $n$ ' is the order of curve) and ' $G$ ' is the generator point (Tata, 2007) & (Huang 2007).

### 3.8.2.1. ECDSA key generation

Key generation is the first step of ECDSA:

1. Choice an elliptic curve ' $E$ ' defined over  $GF(2^m)$ . The number of points on ' $E$ ' should be divisible by a large prime number ' $n$ '.
2. Select a point  $P = (x,y) \in GF(2^m)$  of order ' $n$ '.
3. Select a random integer ' $d$ ' in interval  $[1, n-1]$ . This number acts as the private key.
4. Compute  $Q=dP$
5. The public key is  $(E, P, n, Q)$ .

### 3.8.2.2. ECDSA signature generation

Signing a message  $m$  by sender:

1. Compute a random integer number  $k$  in range  $[1, n-1]$ .
2. Compute  $(x_1, y_1) = kP = k(x,y)$ , and set  $r = x_1 \bmod n$ . If ' $r$ ' equals to zero then go back to step 1. In other words,  $r=0$  is the private key. ' $d$ ' will not involve in the signing equation  $[s = k^{-1}(h(m) + dr) \bmod n]$ .
3. Compute  $k^{-1} \bmod n$ .
4. Compute  $s = k^{-1}(h(m) + dr) \bmod n$ , where  $h(m)$  is the hash value acquired from a appropriate hash algorithm (such as the Secure Hash Algorithm, SHA-1).

5. If  $s=0$  go to step 1. This is because  $s^{-1} \bmod n$  does not exist and  $s^{-1} \bmod n$  is essential in the signature verification.
6. The signature that is included in the message  $m$  is the pair of calculated integers  $(r, s)$ .

### 3.8.2.3. ECDSA signature verification

The following steps should be done to verifying the signature  $(r, s)$  on the message  $m$ :

1. Acquire a valid copy of the sender public key  $(E, P, n, Q)$ .
2. Verify  $r$  and  $s$  are integers in the interval  $[1, n-1]$ .
3. Compute  $w = s^{-1} \bmod n$  and  $h(m)$ .
4. Compute  $u_1 = h(m).w \bmod n$  and  $u_2 = r.w \bmod n$ .
5. Compute  $u_1P + u_2Q = (x_0, y_0)$  and  $v = x_0 \bmod n$ .
6. The signature is accepted only if  $v=r$ .

According to A. Khaled and M. AL-Kayali work, Elliptic Curve Discrete Logarithm Problem (ECDLP) algorithm leads to a very strong security with small keys. If the key is small, the required memory to store the key is also small. Thus, the data which have to be transferred between the card and the reader will be small. It is resulted in short transmission time. Smart card applications need stronger security that can be achieved with long keys. Using ECC smart cards can keep their cost and provide a higher level of security with small key size simultaneously compared to RSA.

ECC reduces the processing time very much especially in using the binary field  $GF(2^k)$ . To do the cryptographic computations, ECC does not need special crypto coprocessor like other PKC systems do. It can be implemented in the available CPU with no additional hardware.

### 3.8.3. WIRELESS SENSOR NETWORK (WSN)

Wireless sensor networks applications include wildlife, earthquake monitoring, and numerous of military applications. A major benefit of these applications is they execute in

network processing. Therefore, it decreases large streams of raw data into useful aggregated information (Perrig, et al. 2001).

Wireless sensors are measured as constrained devices. It is due to the limitation of the number of gates, power and bandwidth etc. Like traditional networks, wireless sensors applications require protection against eavesdropping, alteration, and packet injection. In order to achieve this protection, data cryptography can prevent these security issues. Presently, sensor networks are supplied exclusively through symmetric key cryptography. The entire network is under risk if only one of its nodes has to be compromised by using symmetric cryptography. It means that the shared secret among those nodes is exposed. Another approach is to use a shared key between two nodes in the whole network. Then, it removes the network wide key. The disadvantage is additional nodes cannot be added after the deployment process. In a sensor network with  $n$  nodes, each node needs to store  $n - 1$  keys.  $n * (n - 1) / 2$  keys need to be established in the network. Eventually, a secure key distribution mechanism needs to be achieved as it allows a simple key establishment for large scale sensor networks. The current sensor devices have limited computational power. It makes the implementation of public-key cryptographic primitives and too expensive in terms of system overhead. In order to achieve 80 bit of security in ECC, it needs 160bit parameters size, and gives the same security level offered by 1024 bit RSA. Langendorfer and Piotrowski mentioned some works are addressed as PKC feasibility in WSN by evaluating different parameters (Langendorfer and Piotrowski 2008). Some evaluations consider the processing time as parameter and some evaluations evaluate the memory needs as a parameter. Others new discussed architectures try to utilize the different efforts which are needed to run PKC.

### **3.9.Related work**

A few works found talks about the use of scalar multiplication. Previous research concentrated on both, the computational algorithms and the architectures which those algorithms are executed. Hankerson, Hernandez and Menezes (Hankerson, et al.

2000) provided a very good survey with the aim of discussing software algorithms to compute elliptic curve point multiplication. Many of these algorithms can be adapted to be used in hardware implementation. Großschädl and Kamendje (Großschädl and Kamendje 2003) proposed an architectural change to the multiplier. It is within a RISC processor and software algorithms using the modified multiplier for point multiplication. These implementations commonly use a polynomial basis over binary fields.

Another work by Mohammad R. H. Fatemi, Iqbal J. and S. Rosli provided a description of an ECC co-processor design for constrained devices. Their structure supports all the necessary operations over  $GF(2^{163})$  using the projective representation.

An FPGA implementation proposed by Leung, Ma, Wong and Leong (Leung, et al. 2000) uses a microcoded elliptic curve processor for different key sizes. Several other hardware implementations also can be found. These implementations use microcode instructions to control special-purpose arithmetic units and store transitional results in standard registers. Some reported works allow some flexibility in choosing the ECC parameters. The development of such architecture is hard because of the variety of parameters and the complexity of the underlying algorithms.

M. M. Sandoval (Sandoval 2008) work dealt with the interoperability problems of ECC. It presents the results of a hardware design which allow dynamic adaptation to work with different parameters.

In  $GF(p)$  A multiplication is performed in a serial /parallel approach . In this approach the multiplicand is programmed fully parallel and the multiplier is programmed sequentially (bit by bit).

The reduction modulo is divided with the multiplication by subtractions of once or twice the modulus. Therefore, the arithmetic unit has to perform simply three operations:

1. Addition of partial-products
2. left-shift of the intermediate result
3. And subtraction of once or twice the modulus.

Compared to other bit-serial multipliers, this architecture is benefiting from an efficient subtrahend assessment circuit which does not cause a significant critical path. The modulo multiplier performs multiplications in  $GF(2^m)$ . The introduced MSB-first algorithm requires neither operand transformation into Montgomery domain or pre-computed constants.

This design is scalable in size, and a multiplier that operates over a large range of finite fields. For instance, a multiplier designed for 200 bits as well can be used for fields of smaller order, like 192bit or 163bit, by left-aligning all operands in the registers.

The unified multiplier can be implemented for an area-cost only a little higher than that of the multiplier for the prime field  $GF(p)$ , offering significant area savings. Another advantage of the bit-serial architecture is its high degree of regularity. The presented multiplier gives a reasonable area/ performance trade-off, which makes it attractive for the implementation of a crypto-coprocessor for low end 8bit smartcards.

### **3.10. Conclusion**

In this chapter, an introduction of ECC operations over binary field, prime field and their mathematical operations is explained. With clear examples, how the field operation level work over both fields (Binary, Prime) are shown. Then, higher level operations (ECC operations) are discussed. The process of adding point to another point and point doubling in order to produce a new point is explained. The standard issued by many standard bodies such IEEE ANSI and ISO is discussed to give a clear view of the current ECC issued standards. This chapter also gives an overview of some protocols

such as ECDSA and ECDH. Next chapter is continued with ECC Algorithms and its applications.

## Chapter 4

# ECC Algorithms and its Applications

### 4.1. Introduction

The previous chapter discussed ECC mathematics, it consists a brief description of the Galois field (finite field). According to the hierarchy of Elliptic curve, three operations are needed by the ECC operations which are (Addition, Multiplication and Inversion). Addition operation in binary field is an XOR operation. This chapter describes the basic arithmetic modular multiplication and inversion which are used in elliptic curve over Galois Fields. We will go through many algorithms that are used for computing them and briefly explain about some related algorithms.

### 4.2. Methodology

One of the most important pieces in designing ECC co-processor is the scalar multiplication. Scalar multiplication sets the speed of the designed processor. This can be done by choosing the appropriate algorithms that can be implemented in the hardware environment. To understand Elliptic curves (EC) operations, we will write a program using Java language to draw the EC points with size of 23 over prime field. This program should be able to add two points or doubling a point. After this level of understanding we will be able to go GF operations which concerning three important operations (Inversion, Multiplication, addition). In this phase we will take our first program and modify it to wok over the binary field.

There are different algorithms to calculate each of the three operations (Inversion, Multiplication, addition) but some of these algorithms difficult to be implemented in a hardware environment. As the prime field is not suitable for hardware design, we will focus on the binary field and its algorithms. We will use these algorithms to calculate

the points of the EC over the binary field. Chapter three gives a clear distinction between these algorithms.

After completing the program over the binary field we will move to the scalar multiplication. Mainly scalar multiplication uses two operations (point addition, point doubling) and these two were comprehended at the first phase of our thesis using two different fields (Prime, Binary). The next phase will be designing the scalar multiplication in hardware model using Xilinx ISE.

### 4.3. Multiplication

#### 4.3.1. Integer multiplication

Field multiplication of  $a, b \in F_p$  can be calculated by multiplying 'a' and 'b' as integers and then reduce the result using modulo  $p$ . Algorithms 4.1 shows the multiplication of two integers.

**Algorithm 4.1-** *Integer multiplication.*

**INPUT:** *Integers  $a, b \in [0, p - 1]$*

**OUTPUT:**  *$c = a \cdot b$ .*

1. *Set  $C[i] \leftarrow 0$  for  $0 \leq i \leq t - 1$ .*

2. *For  $i$  from 0 to  $t - 1$  do*

    2.1  *$U \leftarrow 0$ .*

    2.2 *For  $j$  from 0 to  $t - 1$  do:*

*$(UV) \leftarrow C[i + j] + A[i] \cdot B[j] + U$ .*

*$C[i + j] \leftarrow V$*

    2.3  *$C[i + t] \leftarrow U$ .*

3. *Return( $c$ ).*

#### 4.3.2. Right-to-left shift-and-add field multiplication in $F_{2^m}$

The shift-and-add algorithm for field multiplication is based on the:



$$x(z) \cdot y(z) = x_{m-1} z^{m-1} y(z) + \dots x_2 z^2 y(z) + x_1 z y(z) + x_0 y(z)$$

Repetition 'i' in the algorithm 4.1 compute  $z^i y(z) \bmod f(z)$  and if  $x_i = 1$  the result will be added to the accumulator 'c'. If  $y(z) = y_{m-1} z^{m-1} + \dots y_2 z^2 + y_1 z + y_0$  then

$$y(z).z = y_{m-1} z^{m-1} + y_{m-2} z^{m-1} + \dots y_2 z^3 + y_1 z^2 + y_0 z$$

$$y(z).z = y_{m-1} r(z) + (y_{m-2} z^{m-1} + \dots y_2 z^3 + y_1 z^2 + y_0 z) \pmod{f(z)}$$

So  $y(z).z \bmod f(z)$  can be calculated by a left-shift of the vector representation of  $y(z)$ , followed by addition of  $r(z)$  to  $y(z)$  if the high order bit  $y_{m-1}$  is 1.

**Algorithm 4.2-** Right-to-left shift-and-add field multiplication in  $F_{2^m}$ .

**INPUT:** Binary polynomials  $x(z)$  and  $y(z)$  of degree at most  $m-1$

**OUTPUT:**  $c(z) = x(z).y(z) \bmod f(z)$

1. If  $x_0 = 1$  then  $c \leftarrow y$  else  $c \leftarrow 0$

2. For  $i$  from 1 to  $m-1$  do

2.1  $y \leftarrow y.z \bmod f(z)$

2.2 if  $a_i = 1$  then  $c \leftarrow c + y$

3. Return( $c$ )

'x' vector shift in hardware can be performed in one clock cycle, by making Right-to-left shift-and-add field multiplication algorithm that is suitable for the hardware implementations.

### 4.3.3. Montgomery Multiplication Algorithm

P. L. Montgomery in 1985 proposed the Montgomery multiplication algorithm (Montgomery 1985). Trial division does not calculate the modular multiplication. This algorithm computes the modular multiplication with repetition of addition and shift operations. If there is modulus 'p' with m-bit integer ( $2^{m-1} \leq p < 2^m$ ) and 'q'= $2^m$  where Greatest Common Divisor  $\gcd(p, q) = \gcd(p, 2^m) = 1$  then the p-remainder of

an integer 'x' is shown as  $X \equiv x \cdot q \pmod{p}$  where  $0 \leq x < p$ . Let 'X' and 'Y' be two p-remainders, thus the Montgomery product will be:

$$\begin{aligned} Z &\equiv X \cdot Y \cdot q^{-1} \pmod{p} \\ &\equiv (x \cdot q) \cdot (y \cdot q) \cdot q^{-1} \pmod{p} \\ &\equiv (x \cdot y) \cdot q \pmod{p} \\ &\equiv z \cdot q \pmod{p}, \text{ where } 0 \leq z < p \end{aligned}$$

$q \cdot q^{-1} = 1 \pmod{p}$  So it means  $q^{-1}$  is the inverse of  $q \pmod{p}$ . Montgomery multiplication can be calculated from a variety of methods (Koc, Acar and Kaliski 1996). Algorithm 4.3 is the radix-2 Montgomery multiplication algorithm over GF(p) (Walter 1999). It simply can be used to calculate  $GF(2^m)$ . Algorithm 4.4 offers the radix-2 Montgomery multiplication based on  $GF(2^m)$  (Koc and Acar 1997) .

**Algorithm 4.3- Montgomery Multiplication over  $GF(p)$**

**Input:**  $X, Y$  and  $P$ , where  $X, Y \in GF(p)$  and  $P$  is the modulus  $p$ .

**OUTPUT:**  $Z$ , where  $Z \equiv X \times Y \times 2^{-m} \pmod{P}$ .

1. Let  $x_{m-1}x_{m-2} \dots x_1x_0$  be the binary representation of  $X$ .
2. Set  $Z = 0$ .
3. For  $j$  from 0 to  $m - 1$  do.
  - 3.1. Set  $Q = (Z + x_j Y)$ .
  - 3.2. Set  $Z = (Q + q_0 P) / 2$ .
4. If  $Z \geq P$ , then set  $Z = Z - P$ .
5. Output  $Z$ .

**Algorithm 4.4- Montgomery Multiplication over  $GF(2^m)$**

**Input:**  $X(n), Y(n)$  and  $P(n)$ , where  $X(n), Y(n)$  and  $P(n)$

$\in GF(2^m)$ , and  $GF(2^m)$  is generated by  $P(n)$ .

**OUTPUT:**  $Z(n)$ , where  $Z(n) \equiv X(n) \cdot Y(n) \cdot n^{-m} \pmod{P(n)}$ .

1. Let  $X(n) = \sum_{j=0}^{m-1} x_j n^j, \forall x_j$

$\in GF(2)$ , be the polynomial representation of  $X(n)$ .

2. Set  $Z(n) = 0$ .

3. For  $j$  from 0 to  $m - 1$  do.

3.1. Set  $Q(n) = (Z(n) + x_j Y(n))$ .

3.2. Set  $Z(n) = (Q(n) + q_0 P)/n$ .

5. Output  $Z(n)$ .

The suffix  $j$  of the variable points the  $j^{th}$  bit in the polynomial representation of the variable, i.e.  $q_0$  indicates the least significant bit (LSB) of 'Q'. The polynomial representation coefficients of  $X(n)$ , i.e.  $x_{m-1}x_{m-2} \dots x_1x_0$ , also represent the binary of the integer  $X(2)$  while  $X(2) = \sum_{j=0}^{m-1} x_j 2^j$ . By adding two elements in the field  $(2^m)$ , it can be achieved by using the following equation,

$$X(n) + Y(n) = \sum_{j=0}^{m-1} x_j n^j + \sum_{j=0}^{m-1} y_j n^j = \sum_{j=0}^{m-1} (x_j \oplus y_j) n^j$$

The subtraction in  $GF(2^m)$  is similar to addition in  $GF(2^m)$ . Moreover, to apply shift operations a division by number two can shift right the number in  $GF(p)$  and division by 'n' in  $GF(2^m)$ .

#### 4.4. Inversion

Modular inversion is one of the most important modular which is used in cryptographic applications such as the private and public key pair generations. It is used in RSA, the Diffie-Hellman key exchange (Diffie and Hellman 1976), and point addition and point doubling operations in ECC.

#### 4.4.1. Extended Euclidean algorithm for polynomials

The greatest common divisor (GCD) of 'a' and 'b' ('a' and 'b' are binary polynomials and they are not zero), are denoted by  $d = \gcd(a, b)$ . 'd' is the largest common divisor. In the classical Euclidean algorithm,  $\deg(b) \geq \deg(a)$  computes the gcd of binary polynomials. 'b' is divided by 'a' to obtain a quotient 'q'. A remainder 'r' satisfying  $b = qa + r$  and  $\deg(r) < \deg(a)$ .

In such state, the problem in determining  $\gcd(a, b)$  reduces the computation of  $\gcd(r, a)$ , where  $(r, a)$  have lower degrees than  $(a, b)$ . The process should be repeated until one of the arguments reaches to zero. Then the result is immediately obtained since  $\gcd(0, d) = d$ . The algorithm is reached and ended since the degrees of the remainders decrease. The Euclidean algorithm can be extended to find binary polynomials 'x' and 'y' satisfying  $ax + by = d$  where  $d = \gcd(a, b)$ .

$$ax_1 + by_1 = g_1$$

$$ax_2 + by_2 = g_2$$

The algorithm ends when  $u$  value reaches zero, in the case of  $g_2 = \gcd(a, b)$  and  $ax_2 + by_2 = d$ . The next algorithm is used to compute gcd(a,b) (Hankerson, et al. 2004) (Liu 2007).

**Algorithm 4.5** *Extended Euclidean algorithm (EEA) for binary polynomials.*

**INPUT:** Nonzero binary polynomials 'a' and 'b' with  $\deg(a) \leq \deg(b)$

**OUTPUT:**  $d = \gcd(a, b)$  and binary polynomials  $x, y$  satisfying  $ax + by = d$ .

1.  $g_1 \leftarrow a, g_2 \leftarrow b$

2.  $x_1 \leftarrow 1, x_2 \leftarrow 0, y_1 \leftarrow 0, y_2 \leftarrow 1$

3. while  $g_1 \neq 0$  do

    3.1  $j \leftarrow \deg(g_1) - \deg(g_2)$

    3.2 If  $j < 0$  then  $g_1 \leftrightarrow g_2, x_1 \leftrightarrow x_2, y_1 \leftrightarrow y_2, j \leftarrow -j$

```

3.3  $g_1 \leftarrow g_1 + z^j \cdot g_2$ 

3.4  $x_1 \leftarrow x_1 + z^j \cdot x_2$  ,  $y_1 \leftarrow y_1 + z^j y_2$ 

4.  $d \leftarrow g_2$ ,  $g \leftarrow g_2$ ,  $y \leftarrow y_2$ 

5. Return  $(d, x, y)$ 

```

Assume that  $f$  is an irreducible binary polynomial of degree  $m$  and  $a$  has degree at most  $m - 1$  so  $\gcd(a, f) = 1$ . If Algorithm 4.5 is executed with inputs  $a$  and  $f$ , the last nonzero  $g_1$  face in step 3.3 is  $g_1 = 1$ . After this event, the polynomials  $x_1$  and  $y_1$ , as updated in step 3.4, satisfies  $ax_1 + fy_1 = 1$ . Hence  $ax_1 \equiv 1 \pmod{f}$  and  $a^{-1} = x_1$ . Note that  $y_1$  and  $y_2$  are not needed for  $x_1$  determination. These annotations guide to Algorithm 4.6 for inversion in  $F_{2^m}$ .

**Algorithm 4.6-** *Inversion in  $F_2^m$  using the extended Euclidean algorithm.*

**INPUT:** *A nonzero binary polynomial  $a$  of degree at most  $m - 1$ .*

**OUTPUT:**  $a^{-1} \pmod{f}$ .

```

1.  $g_1 \leftarrow a$ ,  $g_2 \leftarrow f$ .

2.  $x_1 \leftarrow 1$ ,  $x_2 \leftarrow 0$ .

3. while  $g_1 \neq 1$  do

    3.1  $j \leftarrow \deg(g_1) - \deg(g_2)$ .

    3.2 if  $j < 0$  then  $g_1 \leftrightarrow g_2$ ,  $x_1 \leftrightarrow x_2$ ,  $j \leftarrow -j$ .

    3.3  $g_1 \leftarrow g_1 + z^j g_2$ .

    3.4  $x_1 \leftarrow x_1 + z^j x_2$ .

4. Return  $(x_1)$ 

```

#### 4.4.2. Binary inversion algorithm in $F_2^m$

Algorithm 4.7 is the polynomial of the binary inversion algorithm. In contrast to Algorithm 4.6, the bits of  $g_1$  and  $g_2$  are cleared from left to right. The bits of  $g_1$  and  $g_2$  in Algorithm 4.6 are cleared from right to left (Hankerson, et al. 2004).

**Algorithm 4.7- Binary algorithm for inversion in  $F_2^m$ .****INPUT:** A nonzero binary polynomial  $a$  of degree at most  $m - 1$ .**OUTPUT:**  $a^{-1} \bmod f$ .1.  $g_1 \leftarrow a, g_2 \leftarrow f$ .2.  $x_1 \leftarrow 1, x_2 \leftarrow 0$ .3. *while*  $g_1 = 1$  and  $g_2 = 1$  *do*    3.1 *while*  $z$  divides  $g_1$  *do*         $g_1 \leftarrow g_1/z$ .        *If*  $z$  divides  $x_1$  *then*  $x_1 \leftarrow x_1/z$ ; *else*  $x_1 \leftarrow (x_1 + f)/z$ .    3.2 *while*  $z$  divides  $g_2$  *do*         $g_2 \leftarrow g_2/z$ .        *If*  $z$  divides  $x_1$  *then*  $x_2 \leftarrow x_2/z$ ; *else*  $x_1 \leftarrow (x_1 + f)/z$ .    3.3 *If*  $\deg(g_1) > \deg(g_2)$  *then*  $g_1 \leftarrow g_1 + g_2, x_1 \leftarrow x_1 + x_2$ ;        *Else*  $g_2 \leftarrow g_2 + g_1, x_2 \leftarrow x_2 + x_1$ .4. *If*  $g_1 = 1$  *then return*  $(x_1)$ ; *else return*  $(x_2)$ .**4.4.3. Montgomery Modular Inverse Algorithm**

Kaliski in 1995 introduced the Montgomery modular inverse (Kaliski 1995). It is defined as the Montgomery demonstration of the modular inverse,  $A^{-1}(\bmod P)$  in which ' $m$ ' is the bit-length of ' $P$ '. Montgomery Modular Inverse Algorithm is based on the extended binary GCD algorithm. This algorithm contains two phases as shown in algorithm 4.8. The result of the second phase can be obtained either by iterative half modulo ' $P$ ' or multiplication modulo ' $P$ '. At the end of the loop, the values of  $g_1 = 1$  and  $g_2 = 0$  allow to check  $G = -A^{-1}2^i(\bmod P)$  which then bring the result back in the range  $[1, P - 1]$ . The following algorithm rewrites the Kaliski Montgomery inverse algorithm with combination of the two phases in one algorithm.

**Algorithm 4.8- Montgomery Modular Inverse Algorithm.**

**Input:**  $A$  and  $P$ , where  $A \in GF(2^m)$  and  $P$  is the modulus  $p$ .

**Output:**  $G$ , where  $G \equiv A^{-1}(\text{mod } P)$

1. Set  $U = P, V = A, G = 0$  and  $K = 1$ .

2. Set  $i = 0$ , where  $i$  is an integer with  $m \leq i < 2m$ .

3. While  $V > 0$  do

3.1 If  $U$  is even, then set  $U = \frac{U}{2}, K = 2K$ .

3.2 Else if  $V$  is even, then set  $V = \frac{V}{2}, G = 2G$ .

3.3 Else if  $U > V$ , then set  $U = \frac{U - V}{2}, G = G + K, K = 2K$ .

3.4 Else if  $V \geq U$ , then set  $V = \frac{U - V}{2}, K = G + K, G = 2G$ .

3.5 Set  $i = i + 1$ .

4. For  $j$  from 1 to  $i$  do

4.1 If  $G$  is even, then set  $G = \frac{G}{2}$ .

4.2 Else set  $G = \frac{G + P}{2}$ .

5. If  $G \geq P$ , then set  $G = 2P - G$ , else set  $G = P - G$ .

6. Output  $G$ .

Compared with the binary method in the algorithm 4.7 to produce regular inverse, Montgomery Algorithm uses a simpler updating for the variables  $K$  and  $G$ .

### 4.5. Double-and-Add Algorithm

Scalar multiplication is the principal back-bone procedure in elliptic curve cryptosystems. It is the most frequent method used in scalar multiplication. It is presented in the top level of Galois Field multiplication and inversion (Moon 2005). This method is challenged by Non-Adjacent Format (NAF) algorithm.

Double-and-add is similar to the square-and-multiply algorithm in the RSA, the modular exponentiation is applied with the algorithm. Double-and-add algorithm is shown in algorithm 4.9.

**Algorithm 4.9- Double-and-Add Algorithm.**

**Input:** A positive integer  $k < n$ , where  $n$  is the order of  $P$ ; and an elliptic curve point  $P$

**Output:** The elliptic curve point  $kP$ .

1. let  $k_n k_{n-1} \dots k_1 k_0$  be the binary representation of  $k$ , where the leftmost bit  $k_n$
2. Set  $G = P$ .
3. For  $j$  from  $n - 1$  to  $1$  do
  - 3.1 Set  $G = 2G$ .
  - 3.2 If  $k_j = 1$ , then set  $G = G + P$ .
4. Output  $G$  (which is equal  $kP$ ).

We decided to choose Right-to-left shift-and-add field multiplication for its suitability in Hardware design in  $F_{2^m}$  as a multiplication algorithm and we used bit-serial modular which is presented in our model in the Design and Implementation chapter. Also we used this algorithm in our Matlab code.

The algorithms that we used and tested for inversion algorithms are Extended Euclidean algorithm for polynomials, Binary inversion algorithm in  $F_2^m$  and Montgomery Modular Inverse Algorithm. Implementing both Extended Euclidean and Binary inversion algorithm in Matlab software is our first phase of understanding these algorithms. Comparing the resulted values from each function, Binary inversion is faster than Extended Euclidean algorithm and Montgomery Modular Inverse Algorithm. However, Montgomery Modular Inverse Algorithm is chosen as a hardware module to obtain inversion in scalar multiplication. Its efficiency is being implemented in hardware as bit-serial. Our bit-serial model is presented in chapter five. Additionally, Double-and-Add Algorithm is described in this chapter to achieve Scalar multiplication.



## **4.6. Conclusion**

This chapter consists of various algorithms used in calculating the Elliptic Curve. In this chapter, algorithms used in multiplication, inversion and MSB are shown. In the next chapter, the chosen algorithms are used in our design and implementation. In our design several algorithms is chosen for their suitability for hardware design and for their speed to achieve the results. In multiplication module we chose Right-to-left shift-and-add field multiplication over Montgomery algorithm for its suitability in hardware implementation. For the inversion module we chose Montgomery algorithm over the binary algorithm for it uses a simpler updating method for it is variables.

## Chapter 5

### Design and Implementation and Testing

#### 5.1. Introduction

The necessary materials for mathematical theorems over the binary field are pointed in the previous chapters. According to the hierarchy of Elliptic Curve, scalar multiplication consists of point addition and point doubling which work over the Galois Field operations. On the other hand GF operations consist of three operations (addition, Multiplication and Inversion). All of these main components used in the scalar multiplier are listed in the following sections. The designed hardware for elliptic curve operations can deal with different field parameters. The point operations on elliptic curves and the implementation of scalar multiplication on hardware play the main part in hardware design.

The Verilog HDL (hardware description language) code for this design is generated in Xilinx software using the parameterized module for different values of ' $m$ '. Result of Xilinx ISE is presented in this chapter to show the synthesis and simulation of each module. Verifying the results of the design is done by using a written Matlab code. This Matlab code uses Right-to-left shift algorithm for multiplication. Inversion is tested with two different algorithms (Euclidean algorithm and Binary algorithm). As a result of not supporting Big Integer in Matlab, a code is written to calculate XOR operation to work with Big Integers. By using this code, the elliptic curve points over prime field ( $\mathbb{F}_{23}$ ) is generated depicted in the Figure 5.1 and 5.2. Later stages explain how an embedded EC in some application works. In order to achieve this, Matlab code is used to generate a digital signature using ECDSA. The domain parameters ( $a, b, Fq, n, G$ ) are obtained from one of the Certicom challenge curves. The code contains two phases which are generating the signature and verifying the signature.

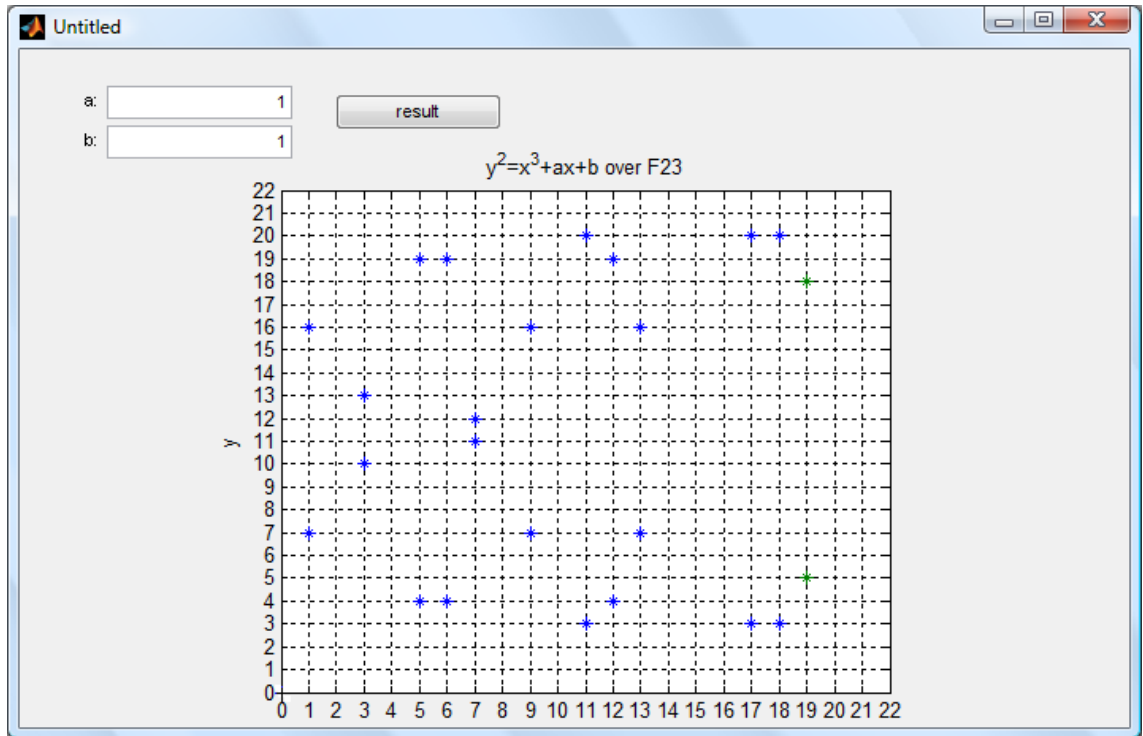


Figure 5.1- the generated points from the equation  $y^2 = x^3 + x + 1$ ,  $a=1$ ,  $b=1$ .

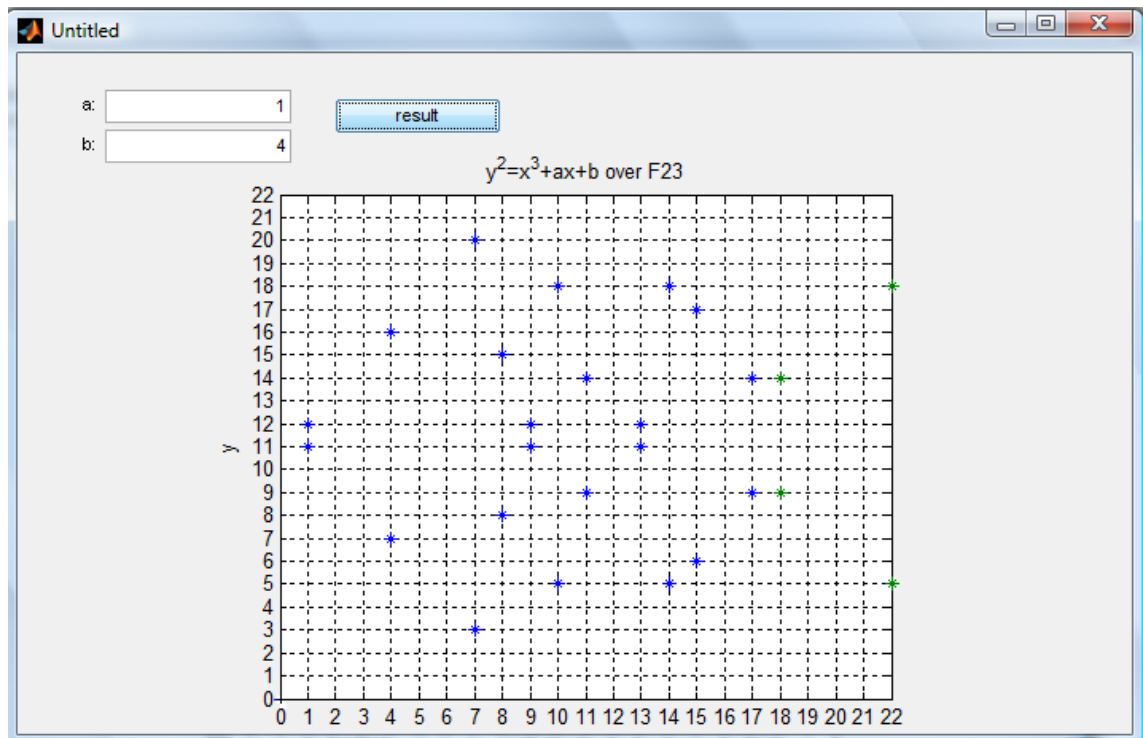


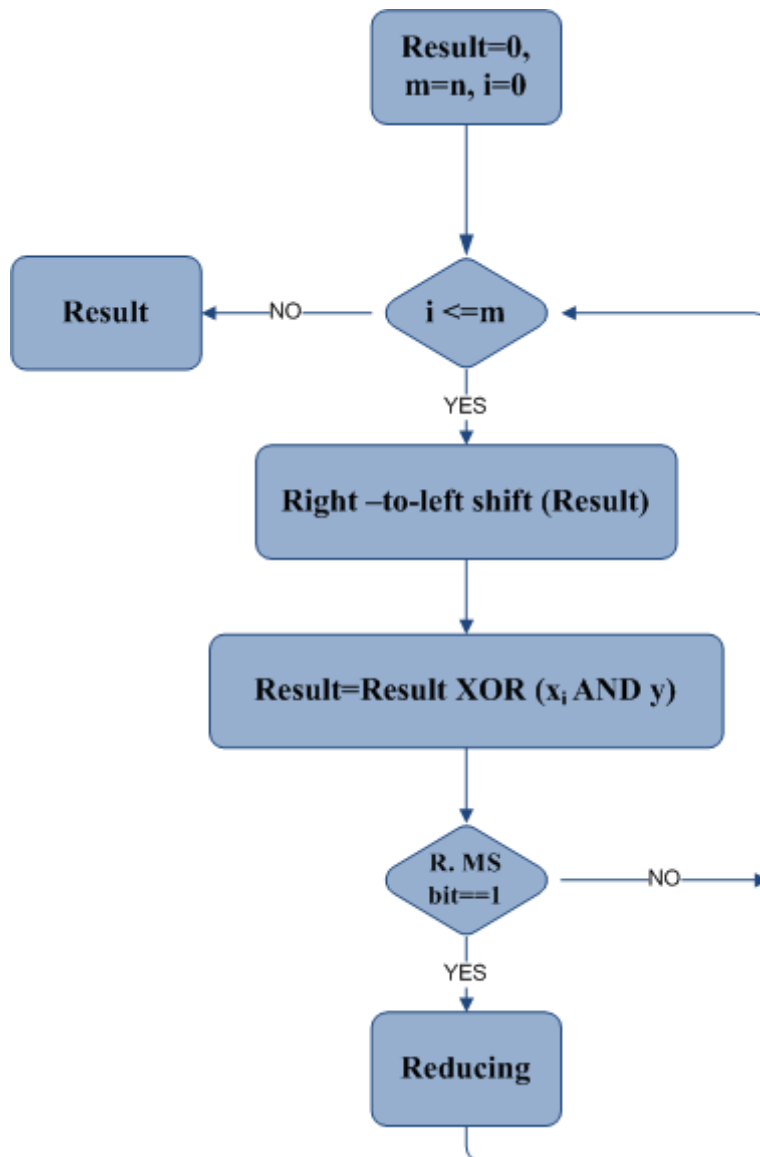
Figure 5.2- the generated points from the equation  $y^2 = x^3 + x + 4$ ,  $a=1$ ,  $b=4$ .

## 5.2. Modular Multiplication

Finite field multiplier over  $F_{2^m}$  always plays a major role in determining the performance of hardware accelerators of cryptography applications. It is necessary to design the multipliers with high efficiency. Bit-parallel and Bit-serial are two options to design a modular. Bit-parallel multiplier can get high operation speed by completing one multiplication in one clock cycle. On the other hand, it has maximum circuit complexity in which a large operand size makes it unsuitable. Bit-serial operators are noticeably smaller than those operators in bit-parallel. They are independent of word width. A multiplier is said to be bit-serial if it produces only one bit of the product at each clock cycle. Demanding only on an equal small amount of input and output pins is an advantage of bit-serial. An implemented multiplication in every bit-serial type has to be directly fitted to the data-width. As a result, the area of complexity is reduced to  $O(n)$  than in parallel multiplier  $O(n^2)$ . Based on the bit-serial advantages, bit-serial is chosen for our design. This section is continued with the efficient designs for polynomial multiplier operation. Right-to-left algorithm is introduced based on the bit-serial architecture.

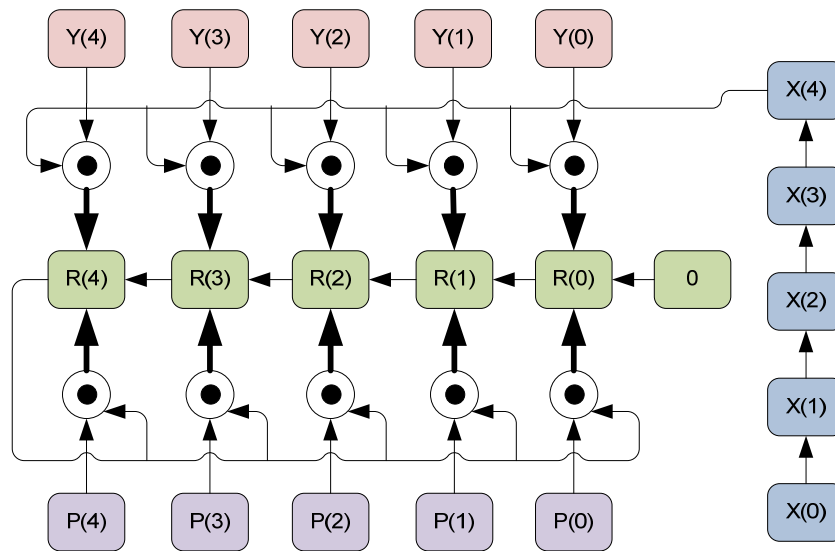
### 5.2.1. Bit-Serial Multiplier Structure

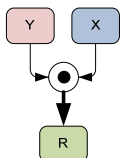
Back to the Algorithm 4.2 which is mentioned in chapter four, the following flowchart shows the bit-serial multiplier steps. When the multiplier bits shifted, the result is stored in 'R'. When  $R(n)$  is a 1, it indicates that the recent partial result overflows the n-bit register. It also reduces one copy of the irreducible polynomial. The reduction is XOR operation. It completes the overall "modulo an irreducible polynomial" correction operation.



**Figure 5.3- Flowchart for Right-to-left algorithm.**

Figure 5.4 depicts a multiplier 'X' and a multiplicand 'Y' when  $X, Y \in F_{2^m}$ . it processes the bits of 'x' from left to right. The multiplier is called a Most Significant Bit (MSB) multiplier. The MSB multiplier can present a multiplication in  $F_{2^m}$  in 'm' clock cycles.



Where  Is  $R = R \oplus (X \wedge Y)$

**Figure 5.4-Most significant bit first (MSB) multiplier for  $F_2^5$ .**

Table 5.1 via  $f_4(x) = x^4 + x + 1$  for two different 'x' and 'y' as input summarizes the behaviour where  $\wedge$  symbolize AND gate and  $\oplus$  symbolized an XOR gate.

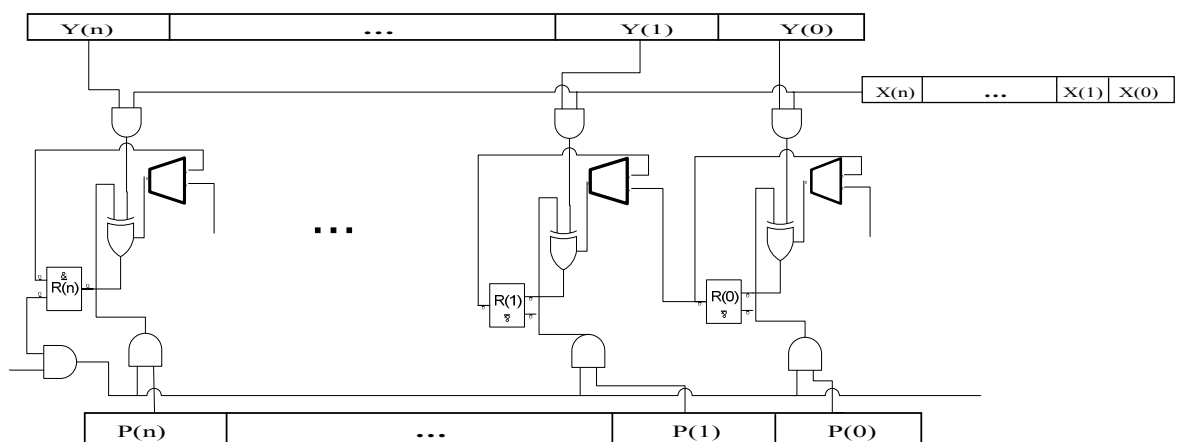
**Table 5.1- XOR ( $\oplus$ ), AND ( $\wedge$ ).**

Result XOR (xi AND y)	Initial	Result	Check MS Bit	Reduction	Shift Result
Result XOR (x1 AND y)	$0000 \oplus 1 \wedge (1101)$	1101	0	-	11010
Result XOR (x2 AND y)	$11010 \oplus 0 \wedge (1101)$	11010	1	$11010 \oplus 10011 = 1001$	10010
Result XOR (x3 AND y)	$10010 \oplus 0 \wedge (1101)$	10010	1	$10010 \oplus 10011 = 0001$	0010
Result XOR (x4 AND y)	$0010 \oplus 1 \wedge (1101)$	1111	0	-	1111

The first partial result is 1101, which shift left (zero fill), then is XORed with 1101. It returns the value 11010, which goes over the 4-bit register limit. It and needs reduction using XOR with the “irreducible polynomial” which is appeared by 10011 in binary for  $f_4(x) = x^4 + x + 1$ . So, in this stage the most significant bit is zeroed and 1001 is as an adjusted result. Yet, it needs to be shifted to left and it returns 10010 as a second line result which XORed to  $0 \wedge (1101)$  in third line. As shown in Table 5.1, most significant (MS) Bit is one. The Final Result is again 5 bits again. The partial result makes the same situation. In order to get the bit alignment, another reduction (XOR

with 10011) is needed. The result of shifting in the third step is 0010. It is used in the last step. As shown in the last step, there is no overflow and the exact 4-bit is a final result.

The following block diagram shows a basic multiplier structure which gets the multiplier data serially. 'X' participates as multiplier and resides in n-bit shift register 'Y' participates as multiplicand in n-bit register. 'R' is used to show the result and 'P' is put as an irreducible polynomial which is used when any overflow happens. External data input connections and Clocks are not shown in Figure 5.5.



**Figure 5.5- The n-bit Multiplier.**

Based on this structure, the diagram will never reach  $n+1$  bit length. It deals with the correction subtraction. By observing the 'n' bit result with the leftmost bit equals to 1, the multiplication mode is set. It can be done by reducing the current result registered by the irreducible polynomials. It is also noticed that the polynomial does not need to represent the leftmost bit of the polynomial.

The circuit shown in Figure 5.5 and its modules are modeled in Verilog HDL language. In order to perform functional finite field multiplication, testbenches are written by using Verilog language. The Verilog files for this section are listed in Appendix B. Figure 5.6 is an example of technology schematic of modular multiplication to test eight bits. Figure 5.7 and Figure 5.8 show testbench multiplications to test two different 'x' and 'y'.

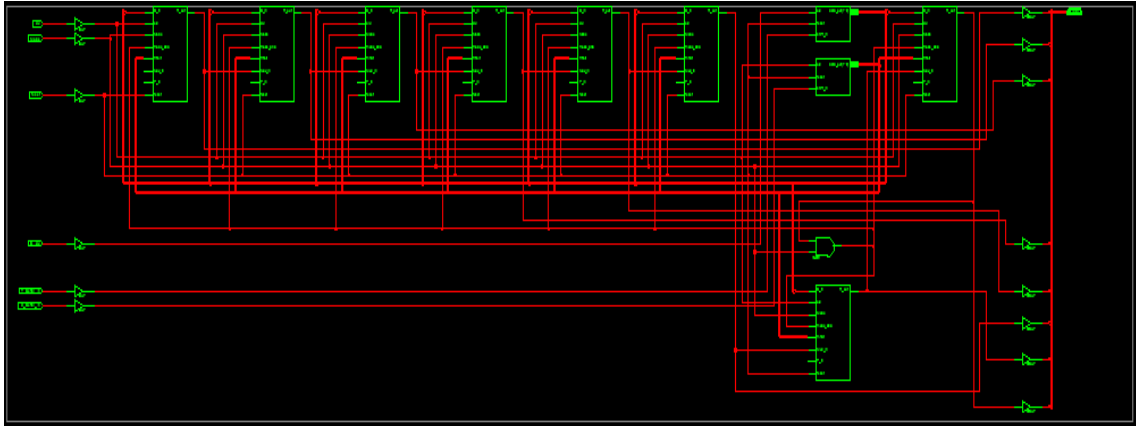


Figure 5.6 -Technology Schematic of Modular Multiplication to test 8 bits.

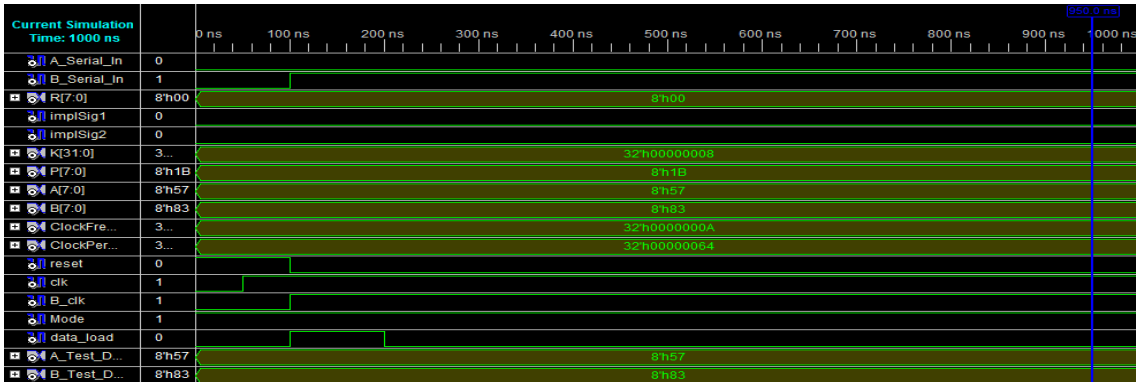


Figure 5.7 Test Bench Multiplications.  $K = 8$ ,  $P = 8'b\ 00011011$ ,  
 $X = 8'b\ 01010111$ ,  $Y = 8'b\ 10000011$ .

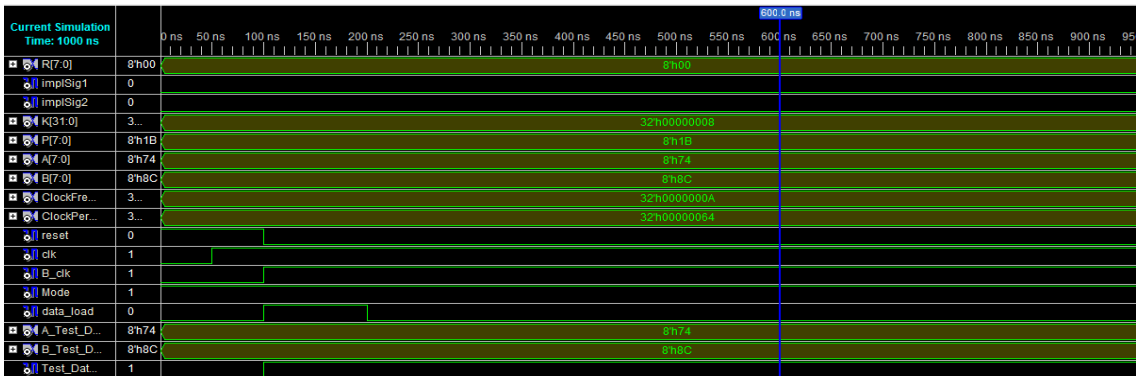


Figure 5.8 Test Bench Multiplications.  $K = 8$ ,  $P = 8'b\ 00011011$ ,  
 $X = 8'b\ 01110100$ ,  $Y = 8'b\ 10001100$ .



For 79 bits, our bit-serial multiplier needs 79 ANDs, 79 XORs and less than 400 FFs (Flip-Flops). A 79 bits multiplication is computed within 79 clocks, which is not including data input and output. In this implementation, control and memory access overheads go to a total time of execution less than 280 clocks. It starts from the processor by sending memory addresses of 'X', 'Y' and 'R' to the last result which is stored in 'R'. The multiplier is also used for squaring by loading  $X = Y$ .

### **5.3.Modular Inversion**

Inversion is the most difficult finite field operation to be implemented in hardware. The division in  $GF(2^m)$   $x/y$  is implemented as two sequential operations, which are the inversion  $y^{-1}$  and then the multiplication  $xy^{-1}$ . Based on the Algorithm 4.8 mentioned in chapter four Figure 5.9 summarizes the Montgomery inversion algorithm that has been chosen to compute inversion in polynomial representations.

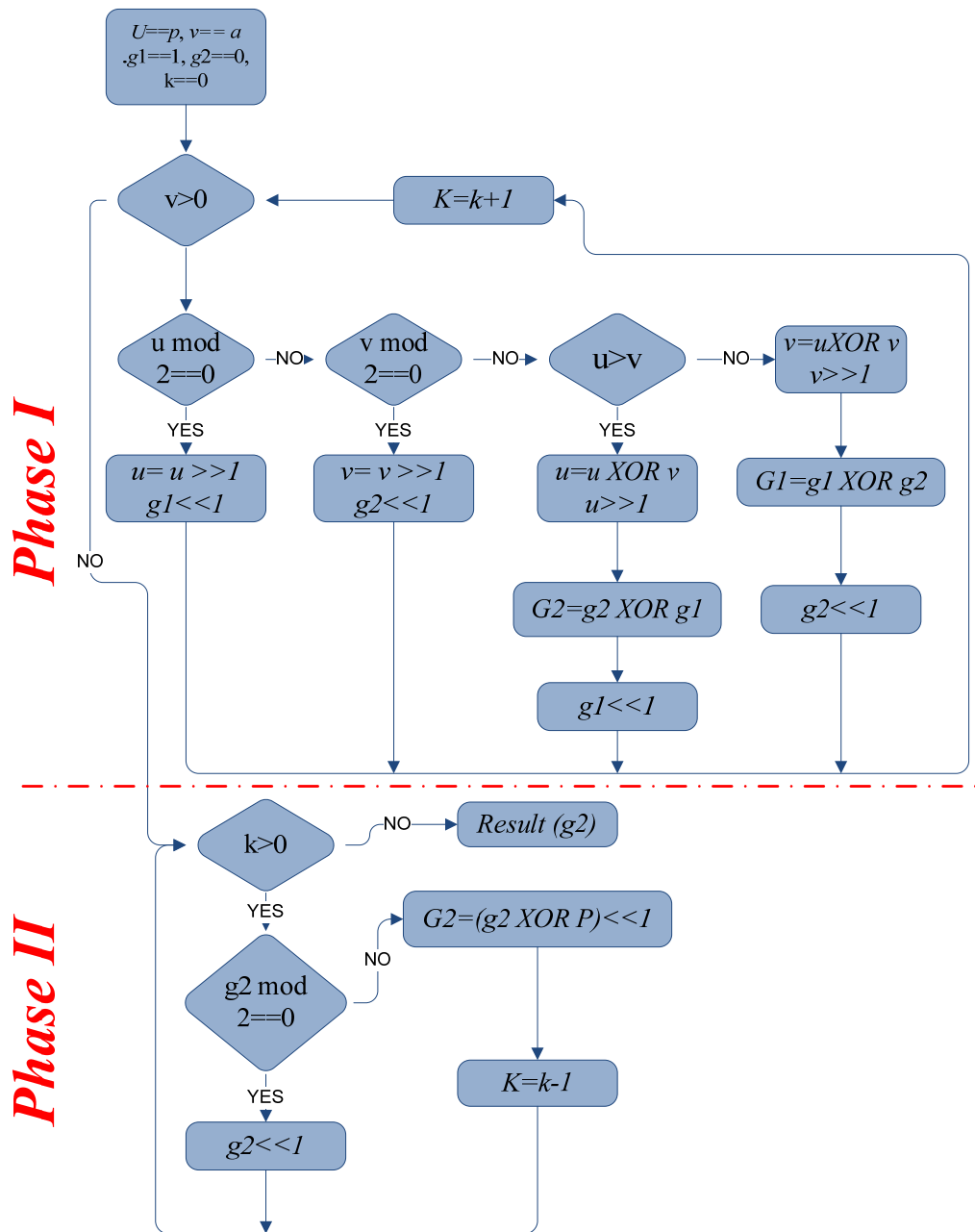
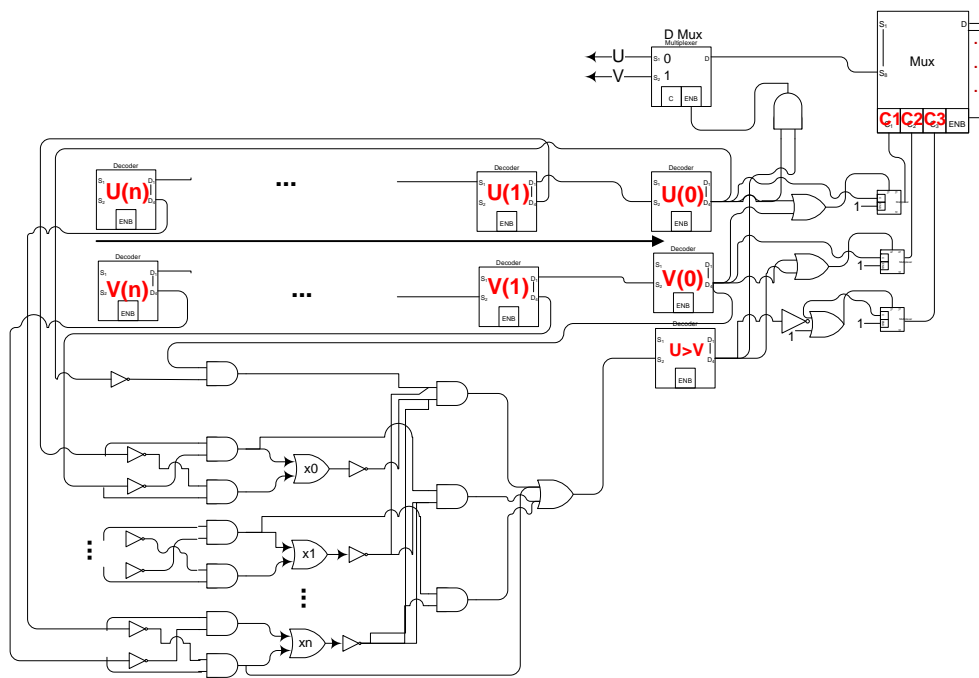


Figure 5.9 - Flowchart for Montgomery Inversion algorithm, Contain Phase I and II.

### 5.3.1. Bit-Serial Inversion Structure

In chapter four, the original Montgomery modular inverse is discussed. For the given advantages in section 5.2, the bit-parallel design is modified into a bit-serial structure. In this section, the modified structure is presented which is based on bit-serial architecture for the most computationally inversion algorithm over Galois field.



**Figure 5.10- Bit-Serial Inversion block diagram.**

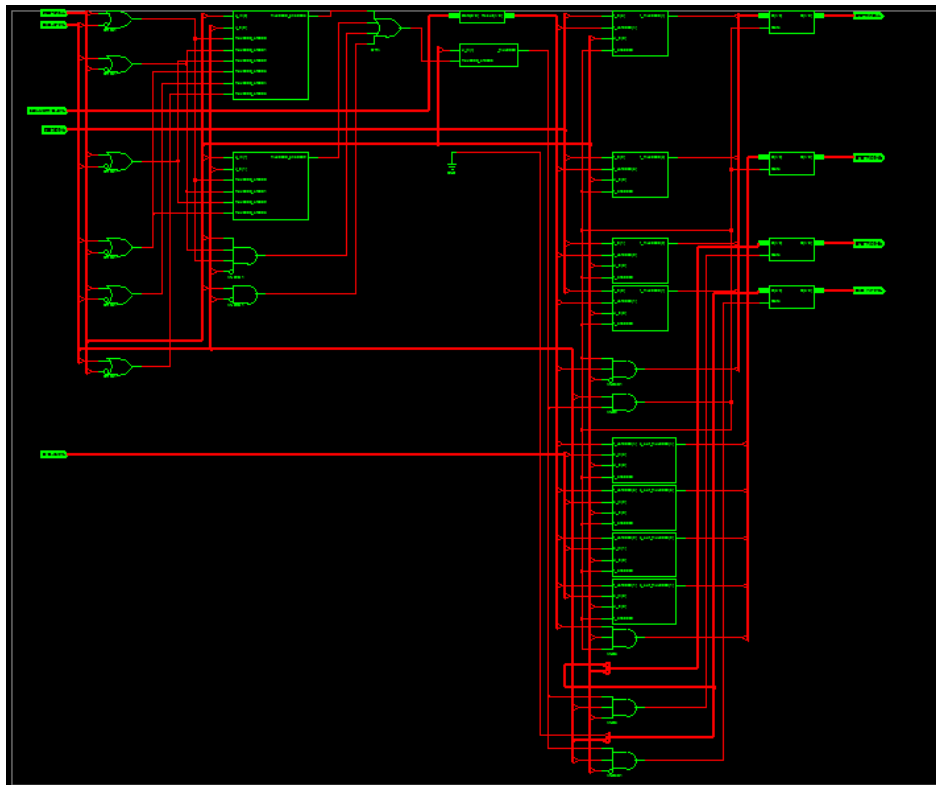
The Figure 5.10 shows the inversion architecture which loads the inputs serially. All the next steps are following algorithm 4.8 including the variables ( $U$ ,  $V$ ,  $K$ , and  $G$ ) to get the inversion result. If ' $U$ ' is even, then the new value of ' $U$ ' and ' $K$ ' is cleared. In order to use bit-serial, it depends on the less significant bit (LSB). Then ' $U$ ' will be known either even or not. If LSB is ' $0$ ', it means ' $U$ ' is even, then, it needs one shift to the right to find  $U=U/2$  and one shift to left ' $K$ ' to find  $K=2*K$ . There is the same row for find ' $V$ ' and ' $G$ '.

In the architecture block diagram, there are a Multiplexer and a De-multiplexer which are controlled by using selector. They are used to pass the correct result to the output. In the De-multiplexer, if ' $C0$ ' and ' $C3$ ' are equals to ' $0$ ', the output is placed in ' $U$ '. Otherwise the value is placed in ' $V$ ' as shown in Figure 5.10. If  $(U < V)$  then  $(U+V)/2$  is the exact result for ' $V$ '. Otherwise it appears as a result for ' $U$ '. The reminding steps are excluded from this dialog in order to not complicate the diagram.

But latter Register Transfer Level (RTL) schematic of this part will be presented. It comes out using Verilog HDL code in Xilinx ISE.

Caused by its architecture which is based on bit-serial, it has low power consumption, regular structure, low cost in term of area occupied and a reduced number of pins. Thus, it is appropriate for embedded applications.

According to Figure 5.9, it contains two Phases which are needed to be run in series to get the inversion. All the modules of both Phases are written in Verilog HDL language. To perform each Phase of inversion, the testbenchs are written in Verilog. The Verilog files for this section are listed in Appendix C. Similar to modular Multiplication, testbench's results are compared with the obtained results from Matlab.



**Figure 5.11- Inversion Architecture (5-bits).**

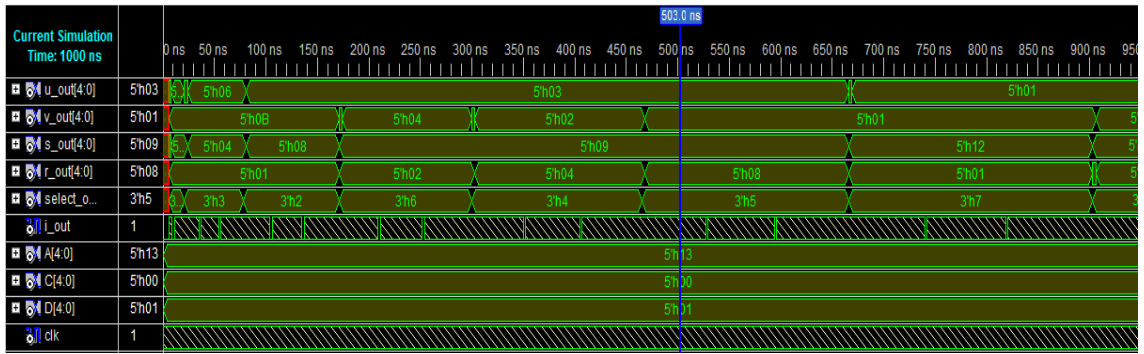


Figure 5.12 – Test Bench (Phase I) (5-bits)

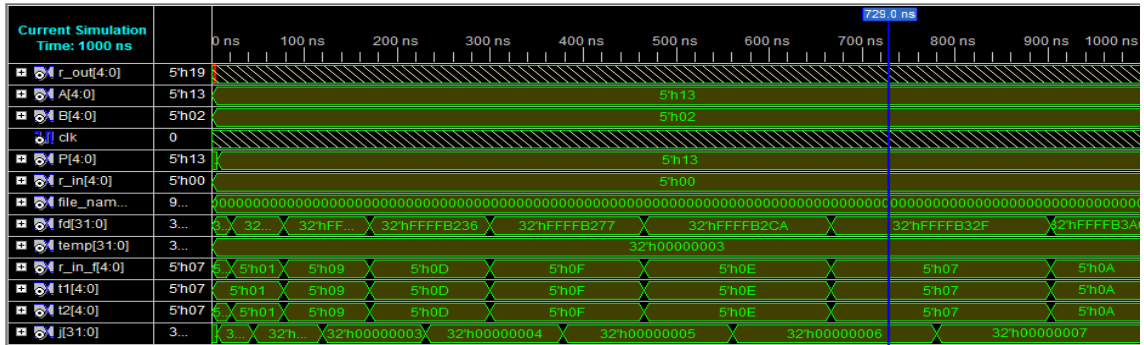


Figure 5.13- Test Bench (Phase II) (5-bits).

There are two phases depicted in Figure 5.9. The first phase is depicted in Figure 5.12 and the second phase is depicted in Figure 5.13.

Both phases require shift left and shift right registers. When shifting to right, the number needs to be divided by two. When shifting to the left, the number needs to be multiplied by two.

In this research, this aim is achieved by using shift-left and shift-right which work in serial-in parallel-out. The two following algorithms are used.

**Algorithm 5.1- Shift\_Right.**

**Input:**  $C, SI [0, m]$ .

**Output:**  $PO [0, m]$ .

Variable  $tmp[m:0], PO [m:0]$ .

1- If (posedge  $C$ )

Begin

2-  $tmp = \{1'b0, SI[m:1]\}$

End

3-  $PO = tmp$ ;

End

**Algorithm 5.2 - Shift\_Left.**

**Input:**  $C, SI [0, m]$ .

**Output:**  $PO [0, m]$ .

Variable  $tmp[m:0], PO [m:0]$ .

```
1- If (posedge C)
    Begin
2-   $tmp = \{SI[m:1, 1'b0]\}$ 
    End
3-  $PO = tmp;$ 
End
```

Appendix C shows Verilog code for both algorithms 5.1 and 5.2 which are used in our implementation.

## 5.4. Scalar Multiplication

The most important arithmetic on elliptic curve applications is the scalar multiplication that computes ' $dP$ '. ' $d$ ' is an arbitrary integer and ' $P$ ' is a point on elliptic curve. The scalar multiplication ' $dP$ ' can simply be clear by adding the  $(d - 1)$  copies of ' $P$ ' to itself.

If the bit check starts from left to right, it is called Most Significant Bit (MSB) or double-and-add. However, if it starts from right to left, it uses Less Significant Bit (LSB) to obtain scalar multiplication.

### 5.4.1. Double-and-Add

Double-and-add method is the common algorithm for computing scalar multiplication, which calculates scalar multiplication by a sequence of point addition and point doublings. Two types of double and add algorithms can be used, specifically,

the most significant bit (MSB) first and the least significant bit (LSB) first. To calculate MSB and LSB, two parameters are required  $(d,P)$  to get the resulted point 'Q'. Algorithm 5.3 is used to calculate MSB and Algorithm 5.3 is used to calculate LSB. LSB first double-and-add algorithm, doubling operations do not rely on the intermediate results of addition operations.

**Algorithm 5.3- Double-and-Add (Most Significant Bit)**

**Input:**  $d = (d_{i-1}, d_{i-2} \dots d_0)_2, d_i = 1$

**OUTPUT:** Compute  $Q = dP$ .

1.  $Q = P$ .
2. For  $j=i-2$  to 0
  - 2.1.  $Q = 2Q$  // it start with Doubling.
  - 2.2. If  $d_j = 1$  then
    - 2.2.1.  $Q = Q + P$  // Addition.
  - 2.3. End if
3. End for.
5. Output  $Q$ .

**Algorithm 5.4- Double-and-Add (Last Significant Bit)**

**Input:**  $d = (d_{i-1}, d_{i-2} \dots d_0)_2, d_i = 1$

**OUTPUT:** Compute  $Q = dP$ .

1.  $Q = 0, R = P$ .
2. For  $j=0$  to  $i-1$ 
  - 2.1. If  $d_j = 1$  then
    - 2.1.1.  $Q = Q + R$  // Addition.
  - 2.2. End if
3. End for.

## 5. Output Q.

There are many advanced researches on modular arithmetic operations over finite fields. For example, Right-to-left shift field multiplication in  $F_{2^m}$  and Montgomery inversion method. These algorithms are used in our bit-serial hardware architecture to calculate scalar multiplication.

Double-and-add algorithm is chosen as it is dictated in ANSI (ANSIX9.62 1999) for scalar multiplication. The double-and-add algorithm is a fundamental technique in calculating scalar multiplication. It performs by repeating point addition and point doubling operations which is discussed earlier. In this section, all equations for point addition and point doubling in  $GF(p)$  and  $GF(2^m)$  are summarised.

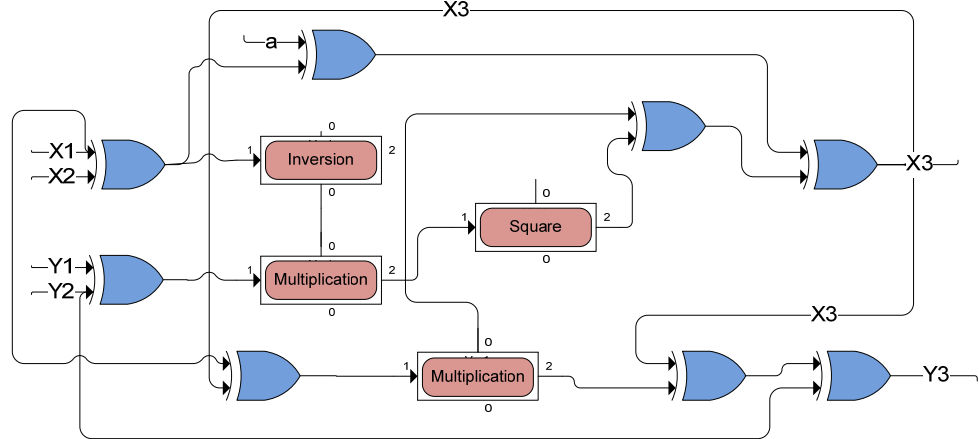
**Table 5.2- Scalar Multiplication.**

	Addition (where $x_1 \neq x_2$ ) $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$	Doubling (where $x_1 = x_2$ ) $(x_3, y_3) = (x_1, y_1) + (x_1, y_1)$
$GF(p)$	$S = (y_2 - y_1) / (x_2 - x_1)$ $X_3 = S^2 - x_1 - x_2$ $Y_3 = S(x_1 - x_3) - y_1$	$S = (3x_1^2 + a) / (2y_1)$ $X_3 = S^2 - 2x_1$ $Y_3 = S(x_1 - x_3) - y_1$
$GF(2^m)$	$S = (y_2 + y_1) / (x_2 + x_1)$ $X_3 = S^2 + S + x_1 + x_2 + a$ $Y_3 = S(x_1 + x_3) + y_1 + x_3$	$S = x_1 + (y_1) / (x_1)$ $X_3 = S^2 + S + a$ $Y_3 = (x_1)^2 + (S + 1)x_3$

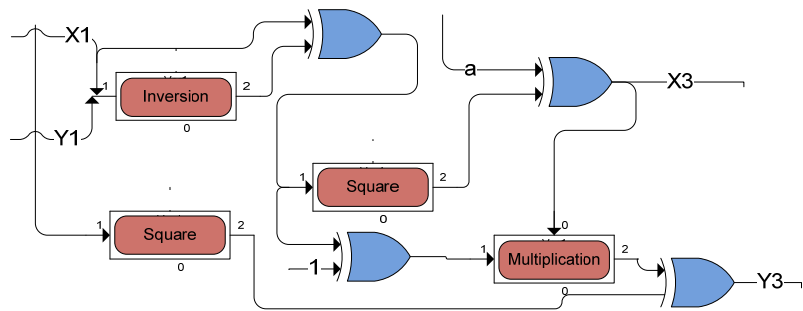
The numbers of ones in the binary representation of 'k' are expected to be  $m/2$ . 'm' represents the length of the integer number 'k'. The number of ones in 'k' shows how many times the point addition will be performed. The number of times for point doubling operation performed is approximately equal to 'm'. Therefore, double-and-add algorithm averagely takes 'm' times point doubling.  $m/2$  times point is added addition to perform m-bit elliptic curve scalar multiplication at one time.



Data flow for ECC point doubling and point addition is based on the Table 5.2 in  $GF(2^m)$  which is presented in Figure 5.14 and Figure 5.15.



**Figure 5.14 – Point Addition in  $GF(2^m)$ .**



**Figure 5.15 – Point Doubling in  $GF(2^m)$ .**

A Montgomery bit-serial modular inversion algorithm and Right-to-left shift multiplication bit-serial are developed in this thesis to reduce the number of Input/output pins for scalar multiplication on elliptic curves by using double-and-add algorithm.

Therefore, in this thesis the scalar multiplication on elliptic curves in  $GF(2^m)$  can be used to deal with various binary polynomials in  $GF(2^m)$ . The arithmetic which are introduced in section 5.4 in  $GF(2^m)$  fields are suitable to be implemented in hardware, since they are binary arithmetic.

**Table 5.3 - Comparison of Computation Steps.  
Double-and-add Scalar Multiplication Algorithm.**

Operation	Point Add	Point Double	Comment
add	$R_1 = x_1 + x_2$	$R_1 = 0 + x_1$	
add	$R_2 = y_1 + y_2$	$R_2 = 0 + y_1$	
Inversion	$R_1 = R_1^{-1}$	$R_1 = R_1^{-1}$	
multiply	$R_2 = R_2 \times R_1$	$R_2 = R_2 \times R_1$	
add	$R_2 = R_2 + 0$	$R_2 = R_2 + x_1$	$\lambda$ (saved)
square	$R_1 = R_2 \times R_2$	$R_1 = R_2 \times R_2$	$\lambda^2$
add	$R_1 = R_1 + R_2$	$R_1 = R_1 + R_2$	$\lambda^2 + \lambda$
add	$R_1 = R_1 + a$	$R_1 = R_1 + a$	$\lambda^2 + \lambda + a$
add	$R_1 = R_1 + x_1$	$R_1 = R_1 + x_1$	$\lambda^2 + \lambda + a + x_1$
add	$R_1 = R_1 + x_2$	$R_1 = R_1 + x_1$	$x_3$ (saved)
add	$R_3 = R_1 + x_2$	$R_1 = R_1 + x_1$	$x_2 + x_1$
multiply	$R_3 = R_3 \times R_2$	$R_1 = R_1 \times R_2$	$(x_2 + x_1) \lambda$
add	$R_3 = R_3 + R_1$	$R_2 = R_2 + R_1$	$(x_2 + x_1) \lambda + x_2$
add	$R_3 = R_3 + y_1$	$R_2 = R_2 + y_1$	$y_3$ (saved)

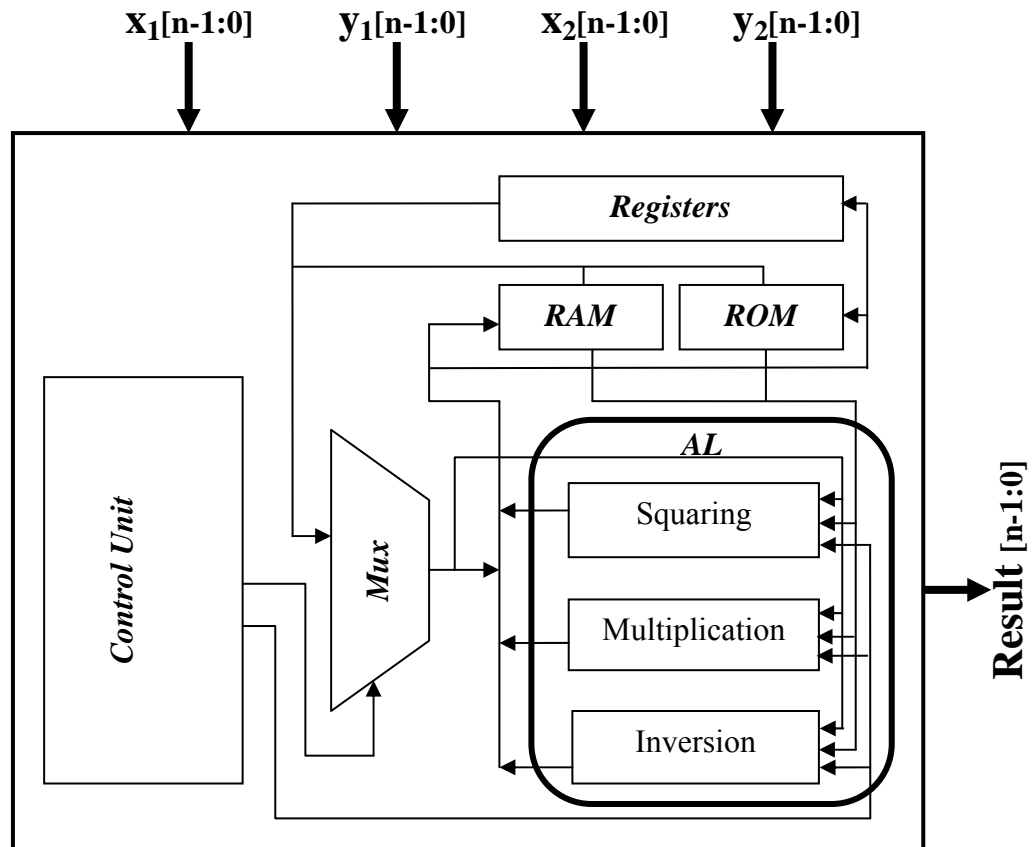
For a better understanding of each step of the Elliptic Curve, Table 5.3 is drawn based on all steps that have been gone through. An XOR operation is used because of the free cost in achieving the binary field addition. There is a free relative to multi-cycle inversion or multiplication based on the XOR operation for addition. So it can be extended inversion and multiplier functional unit in a bit-serial architecture.

### 5.5. ECC Co-Processor

The Elliptic Curve processor is a dedicated hardware implementation. It uses Control Unit (CU), Arithmetic Unit (ALU), RAM and registers which are available on small devices such as RFID. They have a non-volatile memory which is used to personalize the device. A special identifier is assigned to the device. The ECC processor uses memory to store EC definite data in it. This data realizes the secret private key, the public key and other associated data. In this section, our Elliptic Curve Cryptography Processor is presented which is shown in Figure 5.16. It consists of the following blocks:

- Control Unit (CU)

- Arithmetic Unit (ALU)
- ROM and RAM
- Registers



**Figure 5.16 – ECC Co-Processor.**

Data is inputted to this system via the ' $x_1$ ', ' $y_1$ ', ' $x_2$ ' and ' $y_2$ ' with  $[n-1:0]$  length. The data can be loaded through the multiplexer into the different registers. The multiplexer is also used to transfer data from one register to another.

Squaring, multiplication and inversion are performed by using the multiplexer. The result of each operation is loaded into the related register. Additionally, the data in registers can be passed to one of the other registers as required. As managed by the control unit (CU), the system can present different operations in each clock cycle.

### **5.5.1. Control Unit**

The Control Unit (CU) concerns on scalar multiplication, point operations and all switches. It is also in authority to control the ALU which carries out field squaring,

multiplication and inversion. CU is decided when particular data is located onto the wires. It controls the registers to be loaded with this data. Other responsibility of CU is to address the memories.

### 5.5.2. Arithmetic Unit

The largest part of the Arithmetic Unit (ALU) is finite field multiplier, which is the Right-to-left bit-serial multiplier (section 5.2.1.). The bit-serial Montgomery inversion operation is also discussed (section 5.3.1). A bit-serial comes up to minimize the number of gates which has a direct effect to circuits' power consumption.

Arithmetic unit can store intermediate results. Thus, it allows to scale the arithmetic component for arbitrary accuracy without affecting the maximum clock frequency. Table 5.3 shows the offered operations.

According to Figure 5.14, ROM blocks are used to store Elliptic Curve parameters. It has five curve parameter entries which are (a, b, n, G) and the generator-point. 'G' is represented by two elements (x,y) and RAM is used to store intermediate results.

In computing ECC, the series of arithmetic operations must be assured. The intermediate results cannot produce larger than the hardware size of the arithmetic component. Therefore, the hardware is larger than the word size of the EC parameters by some bits.

### 5.5.3. Testing

This section shows the operations of ECC of adding and doubling points. To test our modelled architectures for represented algorithms part, we used the Certicom challenge curve over  $GF(2^{79})$  polynomial reduction  $f(x) = x^{79} + x^9 + 1$ . Certicom issued challenges with different bit lengths. A 79bit is chosen to test our Matlab codes and

Verilog design based on our bit-serial multiplication and inversion modules. Both modules are used in the scalar multiplication design (Certicom n.d.).

**Step 1** initiate the domain parameters that is being found in Certicom challenge as follows

<i>Inputs</i>
Px="56747bc3ddbff399ef4b" Px="101011001110100011110111100001111011101101111111110011100110011110111101001011"
Py="1f8b59e8fddaf314d367" Py="11111100010110101100111101000111111011101101011110011000101001101001101100111"
Qx="30cb127b63e42792f10f" Qy ="110000110010110001001001111011011000111110010000100111100100101111000100001111"
Qy="64b03ef3458f97dd8034" Qy ="110010010110000001111101111001101000101100011111001011111011101100000000110100"
a="4a2e38a8f66d7f4c385f" a="1001010001011100011100010101000111101100110110101111111010011000011100001011111"

Point doubling is applied on ‘P’ to get the resulted point ‘R’. Referring to Table 5.2 the addition of a point to itself (doubling a point) on the Elliptic Curve is calculated as follow:

$$2P=R$$

**Step 2** the following steps shows the way of resolving point doubling

$$\lambda = Px \oplus \frac{Py}{Px}$$

$\lambda = 56747bc3ddbff399ef4b \oplus \left( \frac{1f8b59e8fddaf314d367}{56747bc3ddbff399ef4b} \right)$
$\lambda = 56747bc3ddbff399ef4b \oplus (1f8b59e8fddaf314d367$ $* 56747bc3ddbff399ef4b^{-1})$
$P_x^{-1} = 56747bc3ddbff399ef4b^{-1} = 44f12779ea958c97bd6f$ $P_x^{-1} = "1000100111100010010011101111001111010101001010110001100100101111011110101101111"$
$\lambda = 56747bc3ddbff399ef4b$ $\oplus (1f8b59e8fddaf314d367 * 44f12779ea958c97bd6f)$
$\lambda = 56747bc3ddbff399ef4b \oplus 63e3c23ac6b1548534ad$ $\lambda = 3597b9f91b0ea71cdbe6$ $\lambda = "110101100101111011100111110010001101100001110101001110001110011011011111001110"$
$\lambda^2 = 4f72932a8e87ffc212b6$ $\lambda^2 = "1001111011100101001001100101010100011101000011111111111100001000010010101101110"$

**Step 3** Rx can be calculated using the following Equation

$$Rx = \lambda^2 \oplus \lambda \oplus a$$

$Rx = 4f72932a8e87ffc212b6 \oplus 3597b9f91b0ea71cdbe6$ $\oplus 4a2e38a8f66d7f4c385f$
$Rx = 30cb127b63e42792f10f$ $Rx = "110000110010110001001001111011011000111110010000100111100100101111000100001111"$

**Step 4** Ry can be calculated using the following Equation

$$Ry = x_p^2 + (\lambda + 1) * Rx$$

$Ry = 37293f707e8be11be66f \oplus ((3597b9f91b0ea71cdbe6 \oplus 1)$ $* 30cb127b63e42792f10f)$
$Ry = 37293f707e8be11be66f \oplus$ $(3597b9f91b0ea71cdbe7 * 30cb127b63e42792f10f)$

$Ry = 37293f707e8be11be66f \oplus 539901833b0476c6665b$
$Ry = 64b03ef3458f97dd8034$
$Ry = "11001001011000000111110111100110100010110001111100101111011101100000000110100"$

Point addition is applied on two points 'P' and 'Q' to get 'R'. The following computation shows the addition of two distinctive points on the Elliptic Curve:

$$P+Q=R$$

**Step 2** the following steps shows the way of resolving point Addition

$$\lambda = \frac{(P_y \oplus Q_y)}{(P_x \oplus Q_x)}$$

$\lambda = \frac{(1f8b59e8fddaf314d367 \oplus 64b03ef3458f97dd8034)}{(56747bc3ddbff399ef4b \oplus 30cb127b63e42792f10f)}$
$\lambda = \frac{7b3b671bb85564c95353}{66bf69b8be5bd40b1e44}$
$= 7b3b671bb85564c95353 * 66bf69b8be5bd40b1e44^{-1}$
$66bf69b8be5bd40b1e44^{-1} = 6b4cf2eddfafe5f3e6b3$
$= "110101101001100111100101110110111011111101011111100101111001111100111010110011"$
$\lambda = 7b3b671bb85564c95353 * 6b4cf2eddfafe5f3e6b3$
$\lambda = 4c93fd7da37caac9688a$
$\lambda = "1001100100100111111101011110110100011011110010101010110010010110100010001010"$
$\lambda^2 = 74f4d31bbeb67bf3d8cc$
$\lambda^2 = "11101001111010011010011000110111011110101101100111101111100111101100011001100"$

**Step 3** Rx can be calculated using the following Equation

$$Rx = \lambda^2 \oplus \lambda \oplus P_x \oplus Q_x \oplus a$$

$Rx = 74f4d31bbeb67bf3d8cc \oplus 4c93fd7da37caac9688a \oplus$
$56747bc3ddbff399ef4b \oplus 30cb127b63e42792f10f \oplus 4a2e38a8f66d7f4c385f$

$$R_x = 14f67f7655fc7a7d965d$$

$$R_x = "101001111011001111111011101100101010111111000111101001111011001011001011101"$$

**Step 4 to calculate** Ry value we use the following Equation

$$R_y = \lambda(P_x \oplus R_x) \oplus R_x \oplus P_y$$

$$R_y = 4c93fd7da37caac9688a * (56747bc3ddbff399ef4b \oplus 14f67f7655fc7a7d965d) \oplus 14f67f7655fc7a7d965d \oplus 1f8b59e8fddaf314d367$$

$$R_y = 4c93fd7da37caac9688a * (428204b5884389e47916) \oplus 14f67f7655fc7a7d965d \oplus 1f8b59e8fddaf314d367$$

$$R_y = 5f400ca30340b22f99f4 \oplus 14f67f7655fc7a7d965d \oplus 1f8b59e8fddaf314d367$$

$$R_y = 543d2a3dab663b46dcce$$

$$R_y = "1010100001111010010101000111101101010110110011000111011010001101101110011001110"$$

$$R = (14f67f7655fc7a7d965d, 543d2a3dab663b46dcce)$$

Scalar multiplication uses both point addition and point doubling to compute the resulted point. The execution of the point addition in the scalar multiplication relies on the multiplied number as discussed this in section 5.4.

## 5.6. Conclusion

This chapter consists of the design of modular multiplication and modular inversion that are needed to calculate the scalar multiplication. In addition our bit-serial design for modular multiplication and inversion is presented. Moreover, there are two types of scalar multiplication (MSB, LSB) and MSB is selected as it is dictated by ANSI X9.62 standard. The ECC co-processor is also presented. This chapter ended with the testing part. It presents a testing of our bit-serial designed modules to calculate the scalar multiplication of a point.



## Chapter 6

### 6.1. Conclusion

Nowadays, RSA generally uses public key cryptosystem in most applications that use PKC. However, recently ECC has a trend which makes it become the convenient cryptography system. ECC is also becomes substitute for RSA in efficacious applications caused by its efficiency in software as well as in hardware realizations. ECC provides a better security with shorter bit sizes than in RSA. Shorter key length saves bandwidth, power, and it enhances the performance. In contrast with the past, pairing in ECC attracts more attention of experts because it can be used to build a number of cryptographic schemes that cannot be constructed in any other way. The research starts with survey of cryptography, Elliptic Curve arithmetic and Elliptic Curve operations hierarchy algorithms. A Matlab code has been written to draw the elliptic curve over prime field ( $F_{23}$ ) to understand how an embedded EC in some applications works. A code is written to generate and verify digital signature ECDSA using domain parameters from one of Certicom challenge curves. SHA-1 is chosen as a hashing algorithm as it is recommended by ANSI. The code is written in two phases, which are generating the signature and verifying the signature.

This work focuses on EC operations hierarchy algorithms. Right-to-Left shift Algorithm for multiplication and Binary inversion algorithm for inversion in Matlab code are used as explained in chapter 4.

After completing the finite field design, a model double-and-add algorithm method for Scalar multiplication is modelled. Our approach is begun with competent design for finite field arithmetic, mostly focusing on inversion and multipliers. The design of efficient arithmetic algorithm in bit-serial structure for Right-to-left shift multiplication and Montgomery inversion is shown. Montgomery inversion plays a

consequential task in elliptic curve scalar multiplication. The base algorithm is discussed in chapter four and its implementation is presented in chapter five.

All the field arithmetic units and the scalar multiplier are designed using hardware description Verilog HDL. We use Pentium IV Intel(R) Core(TM) 2 Duo CPU 2 GHZ and 2GB RAM computer. Xilinx XC2C32 is chosen as target device to do the synthesis part and ISE Simulation is used for simulation. As discussed in chapter five, the largest part of the Arithmetic Unit (ALU) is finite field multiplier. Our Right-to-left bit-serial multiplier design is presented and the bit-serial Montgomery inversion operation is discussed. Our model is done based on bit-serial to load the operands serially and gets the result as a serial output in multiplier. A bit-serial approach minimises the number of Input/outputs which has a direct effect on power consumption. Three macrocells per bit are exploited for the multiplicand, multiplier and the product. Eventually, Area saving can be achieved because it does not need to store reduction polynomial in a register. The result of proposed bit-serial architecture for the multiplication and inversion on finite field arithmetic appears to be an important consumption of area in comparison with others.

The implementation of point addition and point doubling using bit-serial architecture shows a great trade-off on area. Therefore, it is suitable for hardware design. By using our design, polynomial finite field for ECC over  $GF(2^m)$  seems to be attractive for their gate count and memory necessity. It proposes the use of our design in ECC Co-processor of embedded devices. Testbanche Xilinx ISE is used for simulation.

## **6.2. Contribution**

- 1) An implementation of scalar multiplication calculates Elliptic Curve Cryptography operations using Matlab and Maple codes.
- 2) Hardware architecture for scalar multiplication uses a bit-serial architecture over binary field operation in Galois field.

3) Elliptic Curve Cryptography Co-processor architecture is designed to work with our scalar multiplication design.

### **6.3. Recommendations for Further Research**

Further work can be done in view of more implementation matters for elliptic curve cryptosystem. In this research, the scalar multiplication over polynomial Galois Field is offered which is based on the bit-serial for finite field arithmetic. Further work can focus on prime fields  $GF(p)$  and unified hardware architectures for finite field arithmetic.

## Bibliography

- Al-Kayali, Ahmad Khaled M. "Elliptic Curve Cryptography and Smart Cards." *SANS Institute*, 17 February 2004: 2004.
- ANSIX9.62. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. New York, USA: American National Standards Institute, 1999.
- Avoine, G., E. Dysli, and P. Oechslin. "Reducing time complexity in RFID systems." In *B. Preneel and S. Tavares, editors, Selected Areas in Cryptography – SAC 2005*,. Lecture Notes in Computer Science. Springer-Verlag., 2005.
- Batina, L., J. Guajardo, T. Kerins, N. Mentens, , P. Tuyls, and I. Verbauwhede. *An elliptic curve processor suitable for RFID-tags*. Report 2006/227: Cryptology ePrint Archive, 2006.
- Bellovin, S., and E. Rescorla. "Deploying a New Hash Algorithm." *In a presentation delivered at the Rump Session of CRYPTO 2005*, 2005.
- Berson, Thomas A. "Differential Cryptanalysis Mod 232 with Applications to MD5." *EUROCRYPT: 71–80*, 1992.
- Borralleras, C., S. Lucas, R. Navarro-Marset, E. Carbonell, and A. Rubio. "Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic." *Springer Berlin / Heidelberg (Springer Berlin / Heidelberg) Volume 5663/2009*, no. 0302-9743 (Print) 1611-3349 (Online) (July 2009): 294-305.
- Braun, M., E. Hess, and B. Meyer. "Using Elliptic Curve on RFID Tags." *IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.2, February 2008*, 2008.
- Certicom. *Certicom*. [www.certicom.com](http://www.certicom.com) (accessed 2009-03-1).
- CIELNY, C. K. "A NEW APPROACH TO THE ELGAMAL ENCRYPTION SCHEME." *Int. J. Appl. Math. Comput. Sci., Vol. 14, No. 2*, 2004: 265–267.
- Daemen, joan, and Vincent Rijmen. "The Design of Rijndael: AES - The Advanced Encryption Standard." *Springer-Verlag*, 2002.
- Diffie, W., and M. E. Hellman. "New directions in cryptography." *IEEE Transactions on Information Theory*,, 1976: 644- 654.
- Dormale, G. M. de, P. Bulens, and J.J. Quisquater. "An improved Montgomery modular inversion targeted for efficient implementation on FPGA." *Field-Programmable Technology*,. 2004. 441- 444.
- Eberle, H., A. Wander, N. Gura, S. Shantz, and V. Gupta. "Architectural Extensions for Elliptic Curve Cryptography over  $GF(2^m)$  on 8-bit Microprocessors." *Architectures and Processors, Samos, Greece*,, 23-25 July 2005.

- ElGamal, T. "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." *IEEE Transactions on Information Theory*, n.4 v.IT-31 1985: pp469-472.
- Feistel, Horst. Block Cipher Cryptographic System. United States of America Patent US Patent 3,796,830. 2 Nov 1971.
- Feldhofer, M., and J. Wolkerstorfer. "Strong Crypto for RFID TagsCa Comparison of Low-Power Hardware Implementations." *In: IEEE International Symposium on Circuits and Systems (ISCAS 2007)*. New Orleans, USA, May 27-30., 2007. 1839-1842.
- FIPS, 186-2. *FIPS 186-2, DSS*. Gaithersburg: National Institute for Standards and Technology, 2000.
- Gaubatz, G., Jens-Peter K. Kaps, E. Ozturk, and B. Sunar. "State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks." *In 2nd IEEE International Workshop on Pervasive Computing and Communication Security (PerSec 2005)*. Kauai Island, Hawaii, March 2005., 2005.
- Großschädl, J., and G. A. Kamendje. "Instruction Set Extension for Fast Elliptic Curve Cryptography over Binary Finite Fields  $GF(2^m)$ ." *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors*. 2003. 455-468.
- Gura, N., A. Patel, A. Wander, H. Eberle, and S. C. Shantz. "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs." *6th International Workshop on Cryptographic Hardware and Embedded Systems*, August 2004.
- Halevi, S., and H. Krawczyk. *Update on Randomized Hashing*. IBM Research, 2006.
- Hankerson, D., A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. 2004.
- Hankerson, D., J. L. Hernandez, and A. Menezes. "Software Implementation of Elliptic Curve Cryptography Over Binary Fields." *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems*. Springer-Verlag, 2000. 1-24.
- Huang, J. "Fpga Implementations Of Elliptic Curve Cryptography And Tate Pairing Over Binary Field." *University Of North Texas*, August 2007.
- IEEE P1363A. "Standard Specifications for Public-Key Cryptography: Additional Techniques." May 2000.
- Ihde, T. *What is hybrid cryptography?* 2003.
- ISO/IEC14888-3. *Information technology – security techniques – digital signatures with appendix Part 3: Certificate based-mechanisms*. Geneva: International Organization for Standardization.

- ISO/IEC15946. *Information Technology - Security Techniques: Cryptographic Techniques based on Elliptic Curves*. Geneva, Switzerland: International Organization for Standardization, 2002.
- Jesper, J. *The Most Misunderstood Windows Security Setting of All Time*. TechNet Magazine. Retrieved on 2007-01-08., 2006.
- Kaliski, Jr., B. S. "The montgomery Inverse and its Applications." *IEEE transaction computers* 44, no. 8 (1995): 1064-1065.
- Knuth, D. E. *The Art of Computer Programming*. MA, USA: Addison-Wesley, 1973.
- Koblitz, N. "Elliptic Curve Cryptosystems." *Mathematics of Computation*, 1987: 203-209.
- Koc, Cetin Kaya, A. Acar, and B. S. Jr. Kaliski. "Analyzing and comparing montgomery multiplication algorithms." *IEEE Micro*, vol. 16, no. 3, June 1996, 1996: pp. 26–33.
- Koc, Cetin Kaya, and A. Acar. "Fast software exponentiation in GF(2k)." in *ARITH '97:Proceedings of the 13th Symposium on Computer Arithmetic (ARITH '97)*. Washington, DC: USA: IEEE Computer Society, 1997. p. 225.
- Koschuch, M., et al. "Hardware/Software Co-Design of Elliptic Curve Cryptography on an 8051 Microcontroller." *Cryptographic Hardware and Embedded Systems CHES 2006*. vol. 4249 ,Springer Verlag, 2006. 430–444.
- Kumar, S. S. *Elliptic Curve Cryptography for Constrained Devices*. June 2006: Ruhr-University Bochum, 2006.
- Kumar, Sandeep S. "Elliptic Curve Cryptography For Constrained Devices." 2006.
- Langendorfer, S. P., and K. Piotrowski. "Public key cryptography empowered smart dust is affordable." *International Journal of Sensor Networks* 4, no. 1/2 (2008): 130-143.
- Leong, P., and I. Leung. "A Microcoded Elliptic Curve Processor Using FPGA Technology." *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*. VOL.10, NO. 5, OCTOBER 2002, 2002. 550-559.
- Leung, K. H., K. W. Ma, W. K. Wong, and Philip H. W. Leong. "FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor." *Proceedings of Field-Programmable Custom Computing Machines*. 2000. 68-76.
- Liu, Yao-Jen. "An Implementation of Universal Dual-Field Scalar Multiplication on Elliptic Curve Cryptosystems." 2007.
- Mackenzie, P, A Chan, and Y Frankel. "Misrepresentation of identities in electronic cash schemes." In. *AsiaCrypt'96*, 1996: 276-285.
- Markoff, J. "A Tool to Verify Digital Records, Even as Technology Shifts." *New York Times*, 26 January 2009.

- Massey, L. James, Lai, and Xuejia. United States of America Patent 5,214,703. 7 January 1992.
- Menezes, A, P Van Oorschot, and S Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- Menezes, A.J. *Elliptic Curve Public Key Cryptosystems*. Boston: Kluwer Academic Publishers, 1993.
- Miller, V. S. "Use of elliptic curves in cryptography." *Advances in cryptology--CRYPTO '85*, 1986: 417-426.
- Montgomery, P. L. "Modular multiplication without trial division." *Mathematics of Computation*, vol. 44, no. 170, April 1985, 1985: pp. 519–521.
- Moon, S. "Elliptic Curve Scalar Point Multiplication Algorithm Using Radix-4 Booth's Algorithm." 2004.
- Mousa, A. "Security and Performance of ElGamal Encryption Parameters." *Journal of Applied Sciences* 5(5):883-886, 2005, ISSN 1812-5654, 2005.
- P1363A, IEEE. "Standard Specifications for Public-Key Cryptography: Additional Techniques." 2000.
- Perrig, A., R. Szewczyk, V. Wen, D. Culler, and J.D Tygar. "SPINS: Security Protocols for Sensor Networks." *Proceedings of ACM MobiCom'01*. Rome, 2001. 189–199.
- Pietiläinen, H. "Elliptic Curve Cryptography on Smart Cards." M.Sc Thesis, Univ. of Technology, Helsinki, 2000.
- Quisquater, J. J., and C. Couvreur. "Fast decipherment algorithm for RSA public-key cryptosystem." *Electronics Letters*, 1982: 18(21):905–907.
- Redmon, K. C. "C0D3 CR4CK3D: Means and Methods to Compromise Common Hash Algorithms." 17 July 2006.
- Riedel, I. "Security in Ad-hoc Networks: Protocols and Elliptic Curve Cryptography , on an Embedded Platform." *Ruhr-Universität at Bochum*. , March 2003.
- Rivest, R. L., A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Communications of the ACM*, 1978: 120-126.
- Sandoval, Miguel Morales. *A reconfigurable and interoperable hardware architecture for elliptic curve cryptography*. PhD Thesis, Instituto Nacional de Astrofísica, 2008.
- Sandoval, MSc. M. M. "A reconfigurable and interoperable hardware." 2008.
- Savas, E., and C. K. Koc. "The Montgomery Modular Inverse - Revisited." *IEEE Transactions on Computers*, 2000: 763-766.

- Schneier, B. "Applied Cryptography." *New York: Wiley Publishing*, 1996: pp.30-31, 428-459.
- Stalling, W. *Network and Internet work Security*. IEEE Press, 1995.
- Stoneburner, G. *Underlying technical models for information technology security*. Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-33, 2001.
- Tata, E. "Elliptic Curve Cryptography, An Implementation Guide." In *Anoop MS*. India: Anoop, MS, 2007.
- Tuyls, P., and L. Batina. "RFID-tags for Anti-Counterfeiting." In *D. Pointcheval, editor, Topics in Cryptology -CT-RSA 2006, Lecture Notes in Computer Science, San Jose, USA*. Springer Verlag., 13-17 February 2006.
- Walter, C. D. "Montgomery exponentiation needs no final subtractions." *Electronics Letters*, vol. 35, no. 21, October 1999, 1999: pp. 1831–1832.
- Wang, X., Y Lisa Yin, and H Yu. "Finding Collisions in the Full SHA-1."
- Wolkerstorfer, J. "Scaling ECC Hardware to a Minimum." *ECRYPT workshop - Cryptographic Advances in Secure Hardware*. 2005.
- Zhou, L., and Z. J. Haas. "Security ad hoc networks." *IEEE Network Magazine*, November 1999: 13(6).



## Appendix A- Elliptic Curve over Field $F_{23}$

### Matlab Code (Elliptic Curve over Field $F_{23}$ )

```
function varargout = uui(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @uui_OpeningFcn, ...
                  'gui_OutputFcn',  @uui_OutputFcn, ...
                  'gui_LayoutFcn',   [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function uui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
initialize_gui(hObject, handles, false);
function varargout = uui_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function density_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function density_Callback(hObject, eventdata, handles)
density = str2double(get(hObject, 'String'));
if isnan(density)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
handles.metricdata.density = density;
guidata(hObject,handles)
function volume_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function volume_Callback(hObject, eventdata, handles)
volume = str2double(get(hObject, 'String'));
if isnan(volume)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
handles.metricdata.volume = volume;
guidata(hObject,handles)
function calculate_Callback(hObject, eventdata, handles)
a1=4*handles.metricdata.density.^3;
b1=27*handles.metricdata.volume.^2;
ab=a1+b1;
if(rem(ab,23)~=0)
    x=zeros(22);
    y=zeros(22);
    x1=1;
    y1=1;
```

```

        for i=1:1:22
            for j=1:1:22

if (rem((i.^3)+(handles.metricdata.density*i)+handles.metricdata.volume
,23)==rem(j.^2,23))
                x(x1)=i;
                y(y1)=j;
                x1=x1+1;
                y1=y1+1;
            end
        end
    end
    x;
    y;
    plot(x,y, '*')
    set(gca, 'YLim', [0 22])
    set(gca, 'YTick', [0:1:22])
    set(gca, 'XLim', [0 22])
    set(gca, 'XTick', [0:1:22])
    xlabel('x')
    ylabel('y')
    title('y^2=x^3+ax+b over F23')
    grid
end
mass = handles.metricdata.density * handles.metricdata.volume;
function reset_Callback(hObject, eventdata, handles)
initialize_gui(gcf, handles, true);
function unitgroup_SelectionChangeFcn(hObject, eventdata, handles)
if (hObject == handles.english)
    set(handles.text4, 'String', 'lb/cu.in');
    set(handles.text5, 'String', 'cu.in');
    set(handles.text6, 'String', 'lb');
else
    set(handles.text4, 'String', 'kg/cu.m');
    set(handles.text5, 'String', 'cu.m');
    set(handles.text6, 'String', 'kg');
end
end
% -----
function initialize_gui(fig_handle, handles, isreset)
if isfield(handles, 'metricdata') && ~isreset
    return;
end
handles.metricdata.density = 0;
handles.metricdata.volume = 0;
set(handles.density, 'String', handles.metricdata.density);
set(handles.volume, 'String', handles.metricdata.volume);
guidata(handles.figure1, handles);

```

## Appendix B - Multiplication

```
// Verilog code for Shift in Multiplication.
`timescale 1 ns / 1 ns

module Shift_N (
    reset,
    clk,
    shiftin,
    dataout);

input  reset;
input  clk;
input  shiftin;
output [5 - 1:0] dataout;

wire [5 - 1:0] dataout;
parameter K = 5; // Example(input)
parameter P = 5'b 11011; // Example(input)
parameter A = 5'b 10111; // Example(input)
parameter B = 5'b 00011; // Example(input)
parameter resetactive = 1'b 1;
reg [K - 1:0] dataint;

always @(posedge clk or posedge reset)
begin : process_1
    if (reset === resetactive)
        begin
            dataint <= {K{1'b 0}};
        end
    else
        begin
            dataint <= {dataint[K - 2:0], shiftin};
        end
    end
assign dataout = dataint;
endmodule

// Verilog code for Each Cell
`timescale 1 ns / 1 ns
module Multiplier_Cell (
    reset,
    clk,
    Zout,
    Bin,
    mbit,
    modeand,
    Pin,
```

```

    Mux,
    Mode);

input    reset;
input    clk;
inout    Zout;
input    Bin;
input    mbit;
input    modeand;
input    Pin;
input    Mux;
input    Mode;

reg      ZZout;
wire     Zout;
wire     xoroutput;
wire     and1;
wire     and2;
reg      muxoutput;

assign and1 = Bin & mbit;
assign and2 = Pin & modeand;
assign xoroutput = and1 ^ muxoutput ^ and2;
always @(Mode or Rout or Mux)
    begin : process_1
        case (Mode)
            1'b 0:
                begin
                    muxoutput <= Zout;
                end
            1'b 1:
                begin
                    muxoutput <= Muxin;
                end
            default:
                begin
                    muxoutput <= Zout;
                end
        endcase
    end
always @(posedge clk or posedge reset)
    begin : process_2
        if (reset === 1'b 1)
            begin
                ZZout <= 1'b 0;
            end
        else
            begin
                ZZout <= xoroutput;
            end
    end
end

```

```

assign Zout = VHDL2V_R_out;

endmodule

// Verilog code for parallel Shift in Multiplication.
`define false 1'b 0
`define FALSE 1'b 0
`define true 1'b 1
`define TRUE 1'b 1
`timescale 1 ns / 1 ns
module shiftParallel (
    reset,
    clk,
    dataA,
    dataB,
    shiftB,
    dataoutput);
input  reset;
input  clk;
input  dataA;
input  [m - 1:0] dataB;
input  shiftB;
output dataoutput;

wire  dataoutput;
parameter m = 5; // Example(input)
parameter P = 5'b 00011011; // Example(input)
//parameter A = 5'b 01010111; // Example(input)
//parameter B = 5'b 10000011; // Example(input)
parameter resetactive = 1'b 1;
reg    [m - 1:0] dataB;
always @(posedge clk or posedge reset)
    begin : process_1
        if (reset === resetactive)
            begin
                dataB <= {m{1'b 0}};
            end
        else
            begin
                if (dataA === 1'b 1)
                    begin
                        dataB <= dataB;
                    end
                else
                    begin
                        dataB <= {dataB[m - 2:0], shiftB};
                    end
            end
    end
end

```

```

    end
    assign dataoutput = dataB[m - 1];
endmodule

```

## Appendix C - Inversion

```

// Verilog code for Shift Right.
`timescale 1ns / 1ps
module Shift_R (C, Serial_in, Parallel_Out);
input C;
input [n:0] Serial_in; // testing for n bits
output [n:0] Parallel_Out; // testing for n bits
reg [n:0] tmp; // testing for n bits
wire [n:0] Parallel_Out; // testing for n bits
    always @(posedge C)
    begin
        tmp = {1'b0, Serial_in [n:1]};
    end
    assign Parallel_Out = tmp;
endmodule

```

```

// Verilog code for Shift Left.
`timescale 1ns / 1ps
module Shift_L (C, Serial_in, Parallel_Out);
input C;
input [4:0] Serial_in; // testing for n bits
output [4:0] Parallel_Out; // testing for n bits
reg [4:0] tmp; // testing for n bits
wire [4:0] PO; // testing for n bits
    always @(posedge C)
    begin
        tmp = { Serial_in [n-1:0], 1'b0};
    end
    assign Parallel_Out = tmp;
endmodule

```

```

// Verilog code for MUX.
`timescale 1ns / 1ps
module Mux(
u_in,v_in,r_in,s_in,u_out,v_out,s_out,clk,reset,r_out,i_out
,select_out);

input [4:0] u_in;
input [4:0] v_in;
input [4:0] r_in;
input [4:0] s_in;

```

```

output [4:0] u_out;
output [4:0] v_out;
output [4:0] s_out;
output [4:0] r_out;
output [2:0] select_out;
output i_out;
input reset;
input clk;
reg [4:0] p;
reg [4:0] u_out;
reg [4:0] v_out;
reg [4:0] s_out;
reg [4:0] r_out;
wire i_out;
wire [2:0] select_out;
reg [4:0] r;

reg [2:0] select; //reg [2:0] select;
wire [4:0] ss_out;
wire [4:0] rr_out;
wire [4:0] uu_out;
wire [4:0] vv_out;
wire [4:0] uv_out;
wire [4:0] uv;
//+++++
reg [80*12:1] file_name = "t.txt";
integer fd;
//+++++
Shift_L sh_s(.C(clk), . Serial_in (s_in), . Parallel_Out
(ss_out));
Shift_L sh_r(.C(clk), . Serial_in (r_in), . Parallel_Out
(rr_out));
Shift_R sh_u(.C(clk), . Serial_in (u_in), . Parallel_Out
(uu_out));
Shift_R sh_v(.C(clk), . Serial_in (v_in), . Parallel_Out
(vv_out));
assign uv=v_in ^ u_in;
Shift_R sh_uv(.C(clk), . Serial_in (uv), . Parallel_Out
(uv_out));

integer i=0;
always @(u_in or v_in or r_in or s_in or reset or clk)
begin
//for(i=0;i<20;i=i+1)
//while(v_in)
// begin
if (u_in[4]==1)
select[0]=1;
else
select[0]=u_in[3]&~v_in[3]|(~(~u_in[3]&v_in[3]|u_in[3]
&~v_in[3])&u_in[2]&~v_in[2])|(~(~u_in[3]&v_in[3]|u_in[3]&~v
_in[3])&~(~u_in[2]&v_in[2]|u_in[2]&~v_in[2])&u_in[1]&~v_in[
1])|(~(~u_in[3]&v_in[3]|u_in[3]&~v_in[3])&~(~u_in[2]&v_in[2

```

```

]|u_in[2]&~v_in[2])&~(~u_in[1]&v_in[1]|u_in[1]&~v_in[1])&u_
in[0]&~v_in[0]);
    select [1]= v_in[0];
    select [2]= u_in[0];

    case (select)
        3'b000:
        begin
            u_out<=uu_out;
            s_out<=ss_out;
            v_out<=v_in;
            r_out<=r_in;
            i=i+1;

        end
        3'b001:
        begin
            u_out<=uu_out;
            s_out<=ss_out;
            v_out<=v_in;
            i=i+1;

        end
        3'b010:
        begin
            u_out<=uu_out;
            s_out<=ss_out;
            v_out<=v_in;
            i=i+1;

        end
        3'b011:
        begin
            u_out<=uu_out;
            s_out<=ss_out;
            v_out<=v_in;
            i=i+1;

        end
        3'b100:
        begin
            v_out<=vv_out;
            r_out<=rr_out;

            u_out<=u_in;
            s_out<=s_in;
            i=i+1;

        end
        3'b101:
        begin
            v_out<=vv_out;
            r_out<=rr_out;
            u_out<=u_in;
            s_out<=s_in;
            i=i+1;

        end
        3'b110:

```



```

        begin
            v_out<=uv_out;
            s_out<=r_in ^ s_in;
            r_out<=rr_out;
            u_out<=u_in;
            i=i+1;
        end
        3'b111:
        begin
            u_out<=uv_out;
            r_out<=r_in ^ s_in;
            s_out<=ss_out;
            v_out<=v_in;
            i=i+1;
        end
    endcase

//+++++
if(clk==1)
begin
    fd = $fopen( file_name, "w" );

    $fwrite(fd,"%b\n%b\n%b\n%b\n%b\n",u_out,v_out,r_out,s_out,s
    elect_out);
    $fclose( fd );
    end
//+++++
end
//end
assign i_out=i;
assign select_out=select;
endmodule

// Verilog code for Phase II.
module Phase_II(clk,k,P,r_in,r_out);
input clk;
input k;
input [4:0] P;
input [4:0] r_in;
output [4:0] r_out;
reg [4:0] r_out;
//+++++
reg [80*12:1] filename = "p.txt";
integer fd;
//+++++
wire [4:0] rr_out;
wire [4:0] rp_out;
wire [4:0] rp;
wire [4:0] p_out;

assign rp=r_in^P;
Shift_R r_Sh(clk,r_in,rr_out);

```

```

Shift_R rp_Sh(clk,rp,rp_out);
Shift_L p_Sh(clk,P,p_out);

always @ (clk or P or r_in )
begin
    if (r_in[0]==0)
        r_out=rr_out;
    else
        r_out=rp_out;

    if(r_out>= P)
        r_out<=p_out^r_out;
    else
        r_out<=P^r_out;
    //+++++
    if(clk==1)
    begin
        fd = $fopen( filename, "w" );
        $fwrite(fd,"%b",r_out);
        $fclose( fd );
    end
    //+++++
end
endmodule

```