

國立交通大學

電子工程學系電子研究所碩士班

碩士論文

橢圓曲線密碼系統之設計與實現

Design and Implementation for Elliptic Curve  
Cryptosystems



研究生：徐維均

指導教授：張錫嘉

中華民國九十四年十月

橢圓曲線密碼系統之設計與實現  
Design and Implementation for Elliptic Curve  
Cryptosystems

學生：徐維均

student :Wei-Chun Hsu

指導教授：張錫嘉

Advisor : Hsie-Chia Chang

國立交通大學

電子工程學系電子研究所碩士班

碩士論文



A Thesis

Submitted to Department of Electronics Engineering  
College of Electrical Engineering and Computer Science

National Chiao Tung University

In Partial Fulfillment of the Requirements

For the Degree of Master

In

Electronics Engineering

October 2005

Hsinchu, Taiwan, R.O.C.

# 橢圓曲線密碼系統之設計與實現

學生：徐維均

指導教授：張錫嘉

電子工程學系電子研究所碩士班

## 摘 要

橢圓曲線密碼系統用來對資料作加密，使得資料在傳輸中不會被竊取。它主要是根據在有限場中的橢圓曲線上之點的運算，加密與解密都是利用點的 scalar multiplication。本論文利用 point halving 演算法，來實現橢圓曲線密碼系統。此實現座落在有限場  $GF(2^{163})$  上，且利用 normal basis。所使用的橢圓曲線為 pseudo-random elliptic curve，其輸入之 base point 為  $\lambda$ -representation，輸入編碼過之 scalar，以為 halve-and-add 演算法所使用。再利用 add-and-subtract 演算法來進一步減少 1 的個數。所使用的 normal basis 乘法器為序列式乘法器，點之相加則利用 projective coordinates。此架構以  $0.18 \mu m$  的製程來實現，需 77K 個邏輯閘。根據模擬的結果，throughput 為 1.76Mb/s。也利用 Xilinx Virtex2 (2V8000) 之 FPGA 作驗證。其頻率為 90Mhz，LUT 數目為 8815。

# Design and Implementation for Elliptic Curve Cryptosystems

Student: Wei-Chun Hsu

Advisor: Hsie-Chia Chang

Institute of Electronics Engineering  
National Chiao Tung University

## ABSTRACT

Elliptic Curve Cryptosystems encrypts data so that the opponent eavesdropping over the channel can't get any information. Its operation is mainly based on the point operations on elliptic curve over finite field. The encryption and decryption utilize scalar multiplication. This thesis demonstrates the implementation of Elliptic Curve Cryptosystems using point halving. This implementation uses normal basis over  $GF(2^{163})$ . The chosen elliptic curve is pseudo-random elliptic curve and the input base point is in  $\lambda$ -representation. The input scalar encoded first for halve-and-add algorithm. We further use the add-and-sub algorithm to reduce the amount of 1's in the input scalar. Serial normal basis multiplier is used while the point addition is in projective coordinates. The architecture is synthesized using  $0.18 \mu m$  technology and requires 77K gates. The throughput is 1.76Mb/s. Verify the implementation with Xilinx Virtex2 (2V8000) FPGA. The frequency is 90Mhz and number of LUTS is 8815.

## 誌 謝

二年的研究所生活很快就過去了，在這兩年中學到了許多知識以及一些處世的道理。當然要感謝的人非常多，首先最要感謝的當然是我的指導教授張錫嘉博士，這兩年來他很有耐心的指導我，不但讓我學到許多 IC 設計的經驗，更亦師亦友的督促我給我鼓勵，真是有幸能在他的實驗室。再來感謝 oasis 實驗室的同學和學弟，很高興認識你們，你們在各方面都給予我許多幫助，帶給我充滿快樂與回憶的研究生生活，再一次跟每個人說一聲謝謝。



# CONTENTS

<b>Chapter 1</b> .....	<b>1</b>
Introduction .....	1
1.1 Motivation .....	2
1.2 Thesis Organization .....	3
<b>Chapter 2</b> .....	<b>4</b>
Mathematical Background.....	4
2.1 Finite Field Arithmetic.....	4
2.2 Elliptic Curve.....	12
<b>Chapter 3</b> .....	<b>25</b>
Scalar Multiplication Algorithms .....	25
3.1 Double-and-Add Algorithm.....	25
3.2 Halve-and-Add Algorithm.....	26
3.3 Add-and-Subtract Algorithm .....	34
<b>Chapter 4</b> .....	<b>37</b>
Implementation Results and Comparisons .....	37
<b>Chapter 5</b> .....	<b>49</b>
Conclusion .....	49
<b>Appendix</b> .....	<b>50</b>
Elliptic Curve Cryptosystems.....	50
a.1 Elliptic Curve ElGamal Cryptosystem .....	51
a.2 Elliptic Curve Diffie-Hellman Key Exchange.....	52
a.3 Elliptic Curve Digital Signature Algorithm.....	53
<b>BIBLIOGRAPHY</b> .....	<b>56</b>

# List of Figures

Figure 2.1: the elliptic curve $y^2=x^3+x+1$ .....	13
Figure 2.2: Point addition, $P+Q=R$ .....	14
Figure 2.3: Point doubling, $2P=R$ .....	14
Figure 2.4: Negative Point, $P+(-P)=O$ .....	15
Figure 4.1: Normal Basis Multiplier version 1.....	39
Figure 4.2: Multiplier element of $c_k$ .....	40
Figure 4.3: Normal basis multiplier version 2.....	40
Figure 4.4: Circuit for point halving.....	42
Figure 4.5: Point halving flow.....	43
Figure 4.6: Circuit for mix-coordinates addition.....	45
Figure 4.7: The control of point halving and projective addition.....	46



# List of Tables

Table 1.1: NIST guidelines for public key sizes for AES.....	2
Table 2.1: Normal Basis Table.....	9
Table 2.2: The multiplication table of type 1 normal basis in $GF(2^4)$ .....	11
Table 2.3: The number of required operations for point doubling .....	24
Table 2.4: The number of required operations for point addition.....	24
Table 2.5: The number of required operations for point addition when $Q= (X_2, Y_2, I)$ .....	24
Table 3.1: Comparison between halving and doubling in affine and projective coordinates...	32
Table 4.1: The data flow of mix-coordinates addition (5.10).....	44
Table 4.2: The synthesized results .....	47
Table 4.3: The performance comparison of Elliptic Curve Cryptosystems implementations on ASIC .....	48
Table 4.4: The performance comparison of Elliptic Curve Cryptosystems implementations on FPGA .....	48






# CHAPTER 1

## Introduction

The objective of cryptography is to enable two people to communicate with each other over an insecure channel such that an opponent can't steal the information. For private key system or symmetric key system, the two people share the same key and this key must be kept secret. Let Alice wants to send information to Bob, which this information is called plaintext. Alice uses the predetermined key to encrypt the information and then send it to Bob. Bob receive the resulting ciphertext and use the same key to decrypt the information back. Advanced Encryption Standard (AES) and Digital Encryption Standard (DES) are private key systems.

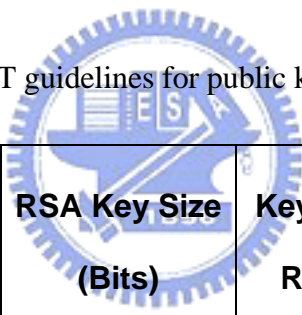


For public key systems or asymmetric key systems, user has two keys one is public key and one is private key which is kept secret. If Alice wants to send a message to Bob, she takes Bob's public key and encrypts the message. After Bob receive the encrypted data Alice sent. He decrypts the message with his own private key. There are two advantages for public key systems. One of them is the amount of keys. Given a group with many users, and users communication with each using private key systems. Then one is required to have the same amount of private keys as the amount of users, since private key can only be share by two people. In case of public key systems, the public key is broadcast to everyone. So given a large group users, only two keys need for each user one public and one private. The other advantage is that private key systems need to establish a secure channel first and send the private key to the user the other side, so that both sides have the same key. While a totally safe channel is not possible, private key system requires addition mechanism to exchange key. The Elliptic Curve Cryptosystems and RSA are public key systems.

## 1.1 Motivation

As the popularity of Internet and WLAN grows, the demand of information security rises. Thus, efficient and secure cryptosystems is of great importance. The Elliptic Curve Cryptosystems, one of the most advanced cryptosystems, is becoming the mainstream security system in all kinds of application. It is a part of the Digital Signature Standard (DSS) proposed by the National Institute of Standards and Technology. The Elliptic Curve Cryptosystems is based on the mathematical operations of elliptic curve. It can achieve high security level using shorter key respect to RSA cryptosystems. As shown bellow[1], the 163-bit ECC key offers the same level of security as 1024-bit RSA key and AES is compared with these two cryptosystems.

Table 1.1: NIST guidelines for public key sizes for AES



<b>ECC Key Size (Bits)</b>	<b>RSA Key Size (Bits)</b>	<b>Key Size Ratio</b>	<b>AES Key Size (Bits)</b>
163	1024	1:6	
256	3072	1:12	128
384	7680	1:20	192
512	15360	1:30	256

In order to speed up the Elliptic Curve Cryptosystem, we proposed an efficient hardware implement for the elliptic curve cryptosystems. The proposed architecture utilizes the point halving technique to achieve a better performance.

## ***1.2 Thesis Organization***

In chapter 2, the mathematical background of finite field, normal basis, polynomial basis, and elliptic curves in projective and affine coordinates are introduced. Chapter 3 shows several algorithms for calculating scalar multiplication, which includes double-and-add algorithm, halve-and-add algorithm, and add-and-subtract algorithm. The idea of Elliptic Curve Cryptosystems is introduced in Chapter 4. Chapter 5 contains the implementation results of the proposed architecture for Elliptic Curve Cryptosystems and comparisons between other implementations are made. Finally, chapter 6 is the conclusion.



# CHAPTER 2

## Mathematical Background

The Elliptic Curve Cryptosystems utilize elliptic curves over finite field, either binary field  $GF(2^n)$  or prime field  $GF(p)$ . In this chapter, I will give the mathematical background related to Elliptic Curve Cryptosystems. I will focus on finite field  $GF(2^n)$ , where two kinds of basis in this field and each of their basic arithmetic will be introduced.

In the second part of the chapter, elliptic curve will be introduced. Elliptic curve, the foundation of Elliptic Curve Cryptosystems, will be specified according to different fields and coordinates. Each different fields and coordinates yields different formulas for the operations of points on elliptic curve, while these point operations are the basis operations of Elliptic Curve Cryptosystems.



### 2.1 Finite Field Arithmetic

#### Polynomial Basis

For finite field  $GF(2^n)$ , the set of polynomial basis is  $\{\alpha^{n-1}, \alpha^{n-2}, \dots, \alpha, 1\}$ , where  $\alpha$  is the root of the field polynomial. Each element belongs to  $GF(2^n)$  could be represented as a linear combination of the basis. For instance, let  $B$  be an element of  $GF(2^n)$  with polynomial basis:

$$B = b_{n-1} \cdot \alpha^{n-1} + b_{n-2} \cdot \alpha^{n-2} + \dots + b_1 \cdot \alpha + b_0 \cdot 1 \quad (2.1)$$

, and we can give a binary notation to this element:

$$B = "b_{n-1}b_{n-2} \dots b_1b_0"$$

For example, given a  $GF(2^8)$  element

$$\alpha^6 + \alpha^4 + \alpha^2 + \alpha + 1$$

or

$$0 \cdot \alpha^7 + 1 \cdot \alpha^6 + 0 \cdot \alpha^5 + 1 \cdot \alpha^4 + 0 \cdot \alpha^3 + 1 \cdot \alpha^2 + 1 \cdot \alpha + 1 \cdot 1$$

, its binary representation is “01010111” where each bit match to the coefficient of each term.

The arithmetic for elements over finite field with polynomial basis will be introduced in the following paragraph.

The sum of two elements in the field is simply bitwise exclusive-or of the two elements. For example,  $(\alpha^6 + \alpha^4 + \alpha^2 + \alpha + 1)$  and  $(\alpha^7 + \alpha + 1)$  are elements of  $GF(2^n)$  and in binary notation we can find the bitwise exclusive-or of these two numbers:

$$“01010111” \oplus “10000011” = “11010100”$$

, which means

$$(\alpha^6 + \alpha^4 + \alpha^2 + \alpha + 1) + (\alpha^7 + \alpha + 1) = (\alpha^7 + \alpha^6 + \alpha^4 + \alpha^2)$$

In the case of multiplication, the two elements are treated as polynomials and multiplied first, then module the result by the field polynomial. An example is shown bellow. Let  $GF(2^8)$  and field polynomial  $f(\alpha) = \alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1$  find  $(\alpha^6 + \alpha^4 + \alpha^2 + \alpha + 1)$  multiply by  $(\alpha^7 + \alpha + 1)$ :

$$(\alpha^6 + \alpha^4 + \alpha^2 + \alpha + 1)(\alpha^7 + \alpha + 1) = \alpha^{13} + \alpha^{11} + \alpha^9 + \alpha^8 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + 1$$

$$\begin{aligned} & \alpha^{13} + \alpha^{11} + \alpha^9 + \alpha^8 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + 1 \bmod (\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1) \\ & = \alpha^7 + \alpha^6 + 1 \end{aligned}$$

So the multiplication result is  $\alpha^7 + \alpha^6 + 1$ .

Squaring is a special case of multiplication when two inputs are the same. For example, let  $B = (\alpha^7 + \alpha + 1)$  be an element of  $GF(2^8)$  and its square

$$\begin{aligned} B^2 &= (\alpha^7 + \alpha + 1)^2 = (\alpha^{14} + \alpha^2 + 1) \\ & (\alpha^{14} + \alpha^2 + 1) \bmod (\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1) \\ & = (\alpha^4 + \alpha^2 + \alpha) \end{aligned}$$

Observing the above example, note that squaring a function has the same result as squaring each and every term. As a result, given a element  $B = B(\alpha) \in GF(2^n)$  and the field polynomial  $f(\alpha)$ .

$$B(\alpha)^2 = B(\alpha^2) \bmod f(\alpha) \quad (2.2)$$

## Normal Basis

In the normal basis case, the set of the basis is  $\{\beta^{2^{n-1}}, \beta^{2^{n-2}}, \dots, \beta^2, \beta\}$  over  $GF(2^n)$  where  $\beta$  is the root of the field polynomial. Each element in the field could be expressed as the linear combination of the basis. Let  $A$  be an element of  $GF(2^n)$  with normal basis:

$$A = a_{n-1}\beta^{2^{n-1}} + a_{n-2}\beta^{2^{n-2}} + \dots + a_1\beta^2 + a_0\beta \quad (2.3)$$

Similarly, we can express each element of the field as a binary number " $a_{n-1}a_{n-2}\dots a_1a_0$ ". Addition in normal basis is the same as polynomial basis. It's still bitwise exclusive-or of the

two elements. For example, let  $(\beta^8 + \beta^2)$  where its binary notation is “1010” and  $(\beta^4 + \beta^2 + \beta)$  “0111” are two elements of  $GF(2^4)$ , then

$$\text{“1010”} \oplus \text{“0111”} = \text{“1101”}$$

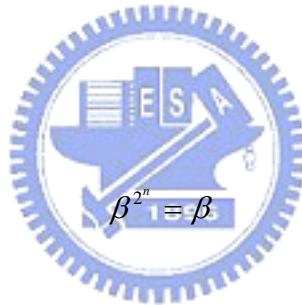
$$(\beta^8 + \beta^2) + (\beta^4 + \beta^2 + \beta) = (\beta^8 + \beta^4 + \beta)$$

In the case of squaring, let  $A$  be an elements of  $GF(2^n)$  equation (2.3): then from equation (2.2)

$$A^2 = a_{n-1}\beta^{2^n} + a_{n-2}\beta^{2^{n-1}} + \dots + a_1\beta^{2^2} + a_0\beta^2 \quad (2.4)$$

and from Fermat’s little theorem:

Given  $\beta \in GF(2^n)$



$$\beta^{2^n} = \beta \quad (2.5)$$

We can derive from equation (2.4)

$$\begin{aligned} A^2 &= a_{n-1}\beta + a_{n-2}\beta^{2^{n-1}} + \dots + a_1\beta^{2^2} + a_0\beta^2 \\ &= a_{n-2}\beta^{2^{n-1}} + \dots + a_1\beta^{2^2} + a_0\beta^2 + a_{n-1}\beta \end{aligned} \quad (2.6)$$

In normal basis, squaring operation is simply one bit cyclic shift of the original data. Let  $A = “a_{n-1}a_{n-2} \dots a_1a_0” \in GF(2^n)$ ,  $A^2 = “a_{n-2} \dots a_1a_0 a_{n-1}”$ . This squaring characteristic gives normal basis an advantage over polynomial basis, because the implement of the normal basis squaring requires no extra hardware only wiring.

Next we will derive the multiplication of normal basis. Suppose  $A, B$  are elements in the field  $GF(2^n)$ :

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i}, B = \sum_{i=0}^{n-1} b_i \beta^{2^i} \quad (2.5)$$

Multiplying  $A$  by  $B$  is defined as bellow:

$$C = A \times B = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \beta^{2^i} \beta^{2^j} = \sum_{i=0}^{n-1} c_i \beta^{2^i} \quad (2.6)$$

Let the product of multiplying  $\beta^{2^i}$  by  $\beta^{2^j}$  be:

$$\beta^{2^i} \beta^{2^j} = \sum_{k=0}^{n-1} \lambda_{ijk} \beta^{2^k} \mid \lambda \in \{0,1\} \quad (2.7)$$

Substitute equation (2.7) into equation (2.6). We can get:

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ijk} a_i b_j \mid 0 \leq k \leq n-1 \quad (2.8)$$

If  $GF(2^n)$  and the number of the nonzero terms or  $\lambda_{ijk}=1$  terms in equation (2.8) equals to  $2n-1$ , then this normal basis is call the optimal normal basis. Optimal normal basis leads to minimum multiplication complexity and thus efficient hardware implement. There are many types of normal basis. [2] gives a chart of existing normal basis type with different field length  $n$  of  $GF(2^n)$ .



Table 2.1: Normal Basis Table

n	Normal basis type	n	Normal basis type
2	1,2	155	2
3	2	156	13
4	1	157	10
5	2	158	2
6	2	159	22
7	4	160	-
8	-	161	6
9	2	162	1
10	1	163	4
⋮	⋮	⋮	⋮

Of all types, only type 1 and type 2 are optimal normal basis.

According to equation (2.7), we raise both side to the power of  $2^{-l}$

$$\left(\beta^{2^i} \beta^{2^j}\right)^{2^{-l}} = \beta^{2^{i-l}} \beta^{2^{j-l}} = \sum_{k=0}^{n-1} \lambda_{i-l, j-l, k} \beta^{2^k} = \sum_{k=0}^{n-1} \lambda_{ijk} \beta^{2^k} \quad (2.9)$$

Comparing the coefficient of the  $\beta^{2^0}$  term, we will get:

$$\lambda_{ijl} = \lambda_{i-l, j-l, 0} \mid \forall 0 \leq i, j, l \leq n-1 \quad (2.10)$$

This implies we can find the value of every  $\lambda_{ijk}$  by means of  $\lambda_{i-k, j-k, 0}$ . And from equation (2.8) utilizing equation (2.10):

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{i-k, j-k, 0} a_i b_j \mid 0 \leq k \leq n-1 \quad (2.10)$$

, and by changing the subscripts

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij0} a_{i+k} b_{j+k} \quad | \quad 0 \leq k \leq n-1 \quad (2.11)$$

The above the equation shows the property of normal basis multiplication. By cycle shifting the subscripts of the formula for  $c_0$ , we can obtain other coordinates of the production. We need to construct a table of  $\lambda_{ijk}$  first before performing normal basis multiplication. For type 1 normal basis and  $GF(2^n)$ , if  $i$  and  $j$  satisfy one of the following congruence then  $\lambda_{ijk}=1$ :

$$2^i + 2^j \equiv 1 \pmod{n+1} \quad (2.12)$$

$$2^i + 2^j \equiv 0 \pmod{n+1}$$

Given type 1 normal basis and  $GF(2^4)$ , the table of  $\lambda_{ijk}$  is constructed bellow following to the rules above. Note that only the  $k=0$  column is needed to be evaluated and the rest of the columns could be easily derived from this column by utilizing equation (2.10). For example:  $\lambda_{001}=\lambda_{330}=1$ ,  $\lambda_{011}=\lambda_{300}=0$ ,  $\lambda_{021}=\lambda_{310}=1$ , and so on.

Table 2.2: The multiplication table of type 1 normal basis in  $GF(2^4)$

i	j	k			
		0	1	2	3
0	0	0	1	0	0
0	1	0	0	0	1
0	2	1	1	1	1
0	3	0	0	1	0
1	0	0	0	0	1
1	1	0	0	1	0
1	2	1	0	0	0
1	3	1	1	1	1
2	0	1	1	1	1
2	1	1	0	0	0
2	2	0	0	0	1
2	3	0	1	0	0
3	0	0	0	1	0
3	1	1	1	1	1
3	2	0	1	0	0
3	3	1	0	0	0

Now we can write the product of the type 1 normal basis multiplication in  $GF(2^4)$  from the above table.

$$c_0 = a_0b_2 + a_1b_2 + a_1b_3 + a_2b_0 + a_2b_1 + a_3b_1 + a_3b_3$$

Since type 1 normal basis is optimal normal basis, the number of terms in above equation equals to  $2*4-1=7$ .

And from equation (2.11)

$$c_k = a_k b_{2+k} + a_{1+k} b_{2+k} + a_{1+k} b_{3+k} + a_{2+k} b_k + a_{2+k} b_{1+k} + a_{3+k} b_{1+k} + a_{3+k} b_{3+k}$$

The formula of other coordinates can be derived by cyclic shifting the subscripts of the  $c_0$  formula:

$$c_1 = a_1b_3 + a_2b_3 + a_2b_0 + a_3b_1 + a_3b_2 + a_0b_2 + a_0b_0$$

$$c_2 = a_2b_0 + a_3b_0 + a_3b_1 + a_0b_2 + a_0b_3 + a_1b_3 + a_1b_1$$

$$c_3 = a_3b_1 + a_0b_1 + a_0b_2 + a_1b_3 + a_1b_0 + a_2b_0 + a_2b_2$$

As for other types of normal basis, [2] provides an efficient algorithm for evaluating the multiplication product the normal basis. Where the type of normal basis and the field length  $n$  of finite field  $GF(2^n)$  is given as the input data of the algorithm.

## 2.2 Elliptic Curve

A *non-singular* elliptic curve over real numbers is described by the following equation:

$$y^2 = x^3 + ax + b \tag{2.13}$$

Where  $a, b$  are real numbers such that

$$4a^3 + 27b^2 \neq 0 \tag{2.14}$$

The elliptic curve is *singular*, if equation (2.14) fails[3]. The following diagram shows an example of an elliptic curve where  $a=b=1$ . Note that the diagram is symmetric with respect to  $x$ -axis.

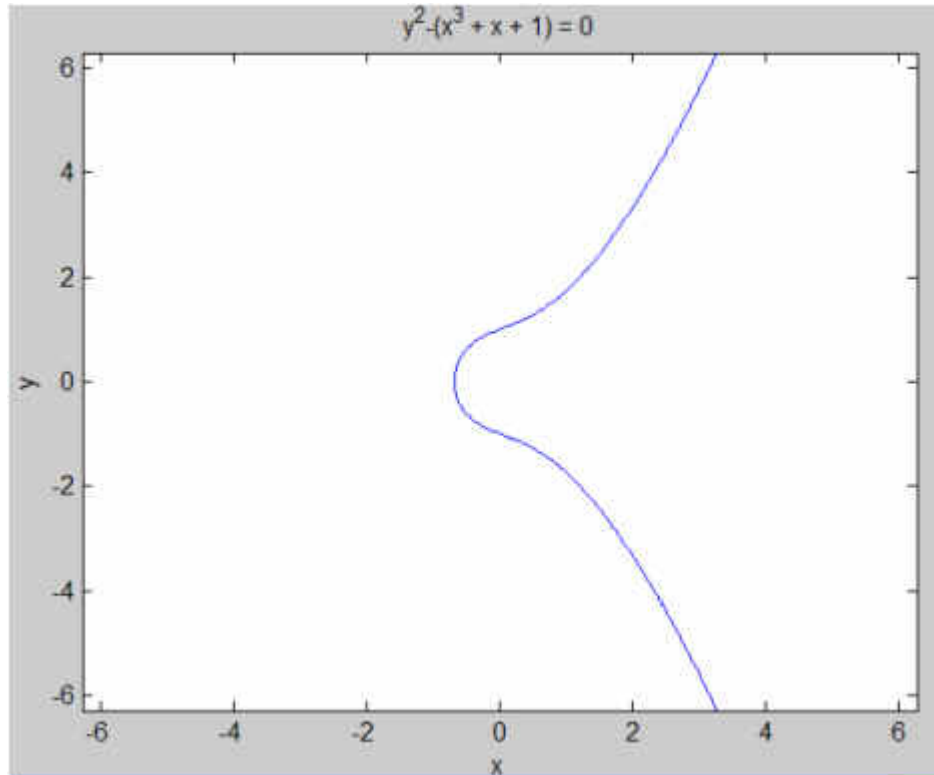


Figure 2.1: the elliptic curve  $y^2 = x^3 + x + 1$

For finite field  $GF(p)$ , the elliptic curve satisfies the congruence, where  $a, b \in GF(p)$ :

$$y^2 \equiv x^3 + ax^2 + b \pmod{p} \quad (2.15)$$

For finite field  $GF(2^n)$ , the elliptic curve is in a slightly different form as shown below, where  $a, b \in GF(2^n)$ :

$$y^2 + xy = x^3 + ax^2 + b \quad (2.16)$$

An abelian group can be defined on the set  $E$  of solutions  $(x, y)$  to the elliptic curve equation plus a point  $O$  at infinity. Now consider the addition law of elliptic curve:

Given two points  $P$  and  $Q$  on elliptic curve  $E$ , consider the result of  $P+Q$ . First, we define  $L$  to be the line through  $P$  and  $Q$ . The  $L$  intersects  $E$  at point  $R'$ , then we reflect  $R'$  in

the  $x$ -axis to get  $R$ . We define  $R$  to be the result of  $P+Q$ , that is,  $P+Q=R$ . An example is given below:

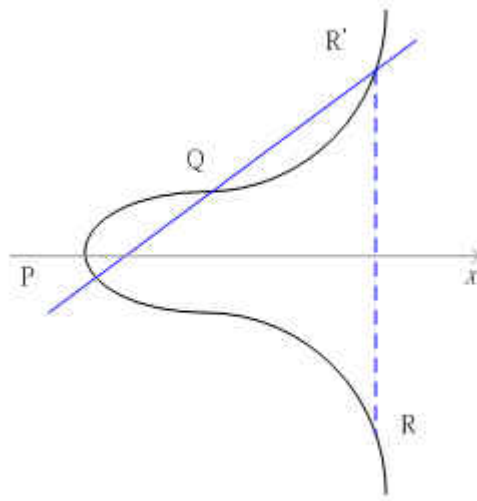


Figure 2.2: Point addition,  $P+Q=R$

Now consider the situation when  $P=Q$ , namely, consider the result of  $2P$ . Since  $P=Q$ , line  $L$  now become a tangent line passing through  $P$ . Similarly, the line  $L$  intersects  $E$  at point  $R'$ , then we reflect the  $x$ -axis to obtain the result  $R$ . The following diagram shows this condition:

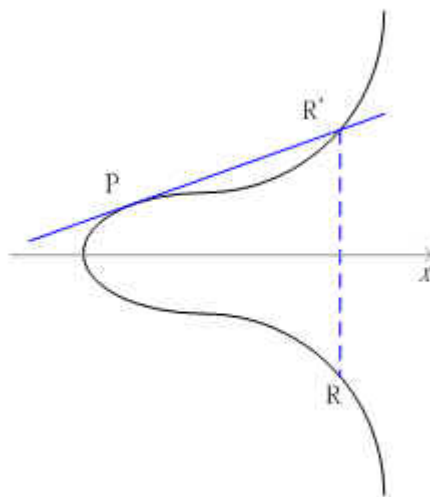


Figure 2.3: Point doubling,  $2P=R$

The point at infinity  $O$  is considered as the identity element:

$$P+O=O+P=P \tag{2.17}$$

We consider the case when  $Q$  is the reflection of  $P$  in the  $x$ -axis. So if we draw a line  $L$  through  $P$  and  $Q$ , then line  $L$  will be a vertical line through  $P$  and intersect  $E$  at infinity  $O$  and we can get  $P+Q=O$ . Since  $O$  is the identity element, we can consider that  $Q$  as the negative of  $P$ , that is  $Q=-P$ . We can conclude that the negative point of a given point is the reflection of the point in the  $x$ -axis.

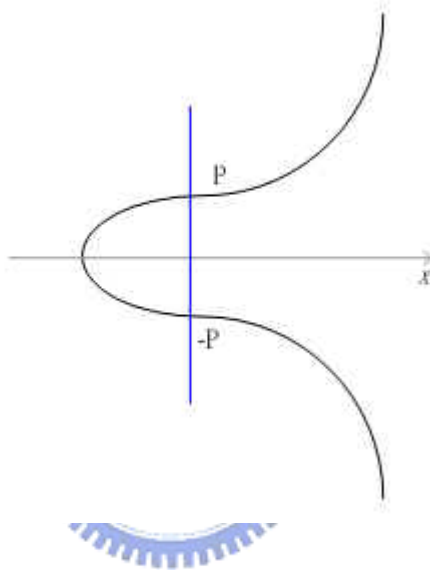


Figure 2.4: Negative Point,  $P+(-P)=O$

Given a point  $P \in E$  over finite field, then  $E$  is a finite abelian group. We can find an integer  $r$  such that  $rP = \overbrace{P + P + \dots + P}^r = O$ . The integer  $r$  is called the order of point  $P$ .

Next, I will derive the addition and doubling formula for points on elliptic curve according to the addition law mentioned above. Moreover, a different kind of representation called the projective coordinates representation will be introduced.

### Affine Coordinates Representation

Affine coordinate representation is respect to projective coordinates representation. Given an elliptic curve  $E: y^2=x^3+ax+b$ , let's derive the negative of a point first. Let  $P=(x_1,$

$y_1$ ), the negative of  $P$  is simply the corresponding point of the reflected  $P$  in the  $x$ -axis which is  $(x_1, -y_1)$ .

$$-(x_1, y_1) = (x_1, -y_1) \quad (2.18)$$

We next derive the formula for point addition  $P+Q=R$ . Let  $P, Q \in E$ , where  $P=(x_1, y_1)$ ,  $Q=(x_2, y_2)$ ,  $R=(x_3, y_3)$  and  $L$  is the line passing through  $P$  and  $Q$  represented as

$$y = \lambda x + \nu \quad (2.19)$$

, where the slope of  $L$  is:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \quad (2.20)$$

, and

$$\nu = y_1 - \lambda x_1 = y_2 - \lambda x_2 \quad (2.21)$$


$L$  will intersect  $E$  at point  $R'$ . Substitute equation (2.19) into the equation for  $E$  to find the solution of the coordinates, we can get

$$(\lambda x + \nu)^2 = x^3 + ax + b \quad (2.22)$$

, we can derive from above

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\nu)x + b - \nu^2 = 0 \quad (2.23)$$

We have to solve equation (2.23) for the  $x$ -coordinates. Since  $x_1$  and  $x_2$  are two roots of equation (2.23), the sum of the three roots will equal to



$$x_1 + x_2 + x_3 = \lambda^2$$

$$x_3 = \lambda^2 - x_1 - x_2 \quad (2.24)$$

Since  $R'$  equals to  $(x_3, -y_3)$ . We can derive

$$\lambda = \frac{-y_3 - y_1}{x_3 - x_1} \quad (2.25)$$

, or

$$y_3 = \lambda(x_1 - x_3) - y_1 \quad (2.26)$$

For the case when doubling a point, we have to find the slope of the tangent line  $L$  to point  $P=(x_1, y_1)$ . Let  $2P=(x_3, y_3)$ , using the implicit differentiation of the equation of  $E$

$$2y \frac{dy}{dx} = 3x^2 + a \quad (2.27)$$

So the slope of the tangent line  $L$  with equation (2.22) to point  $P$  is

$$\lambda = \frac{3x_1^2 + a}{2y_1} \quad (2.28)$$

and

$$v = y_1 - \lambda x_1 \quad (2.29)$$

The line will intersect with  $E$  at  $R'=(x_3, -y_3)$  and substitute the line equation into  $E$ . Regarding equation (2.23), the cubic equation has two roots at  $x_1$ , and one root at  $x_3$ . So  $x_3$  equals:

$$x_3 = \lambda^2 - 2x_1 \tag{2.30}$$

With the same procedure, we can find  $y_3$  by equation (2.26).

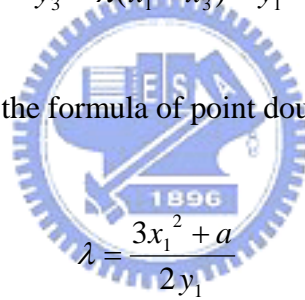
Finally, the formula for point addition and point doubling can be summarized as bellow. Suppose  $P=(x_1, y_1)$ ,  $Q=(x_2, y_2)$ ,  $P+Q=(x_3, y_3)$ , elliptic curve with equation (2.13) or (2.15), then the formula of point addition:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \tag{2.31}$$

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

Let  $P=(x_1, y_1)$ ,  $2Q=(x_3, y_3)$ , the formula of point doubling



$$\lambda = \frac{3x_1^2 + a}{2y_1} \tag{2.32}$$

$$x_3 = \lambda^2 - 2x_1$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

When used over finite field  $GF(2^n)$ , the elliptic curve is in the form (2.16). We can derive the formulas for point addition and point addition over finite field  $GF(2^n)$  in a similar method. As in the previous context, we will derive the negation of a point first. Given a point  $P=(x_1, y_1)$ , we try to find the representation of  $-P=(x_2, y_2)$ . As mentioned above that  $P+-P=O$ , we draw a vertical line  $L$  through  $P$  and the line will intersect  $E$  at point  $-P$ . The equation of this line  $L$  is simply

$$x+x_1=0 \tag{2.33}$$

, which implies that  $x_2+x_1=0$  and the  $x$ -coordinate of  $-P$  is  $x_1$ . Substitute equation (2.33) into equation (2.16) in order to find the solution of the  $y$ -coordinate of  $-P$ . We will get:

$$y^2+x_1y=x_1^3+ax_1^2+b \tag{2.34}$$

This square equation has two solutions and one of them is  $y_1$ . The sum of the two solutions will equal to the coefficient of the term  $y$ . As the result,

$$y_1+y_2=x_1$$

, or



$$y_2=x_1+y_1 \tag{2.35}$$

So for  $P=(x_1, y_1)$ , the negation of  $P$  over finite field  $GF(2^n)$

$$-(x_1, y_1)=(x_1, x_1+y_1) \tag{2.36}$$

Again, let  $P, Q \in E$ , where  $P=(x_1, y_1)$ ,  $Q=(x_2, y_2)$ ,  $P+Q=R=(x_3, y_3)$  and  $L$  is the line passing through  $P$  and  $Q$ .  $L$  has the equation (2.19), where

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1} \tag{2.25}$$

and

$$v = y_1 + \lambda x_1 = y_2 + \lambda x_2 \tag{2.26}$$

Substitute the equation of  $L$  (2.19) into the elliptic curve equation (2.16)

$$(\lambda x + \nu)^2 + x(\lambda x + \nu) = x^3 + ax^2 + b \quad (2.27)$$

, it is the same as

$$x^3 + (\lambda^2 + \lambda + a)x^2 + \nu x + b = 0 \quad (2.28)$$

$x_1$  and  $x_2$  are two solutions of the cubic equation, we can find  $x_3$  from the coefficient of  $x^2$  term

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \quad (2.29)$$

Then same as before, we use  $R'=(x_3, x_3+y_3)$  and  $P=(x_1, y_1)$  to compute the slope

$$\lambda = \frac{x_3 + y_3 + y_1}{x_3 + x_1} \quad (2.30)$$

Derived from above,

$$y_3 = \lambda(x_3 + x_1) + x_3 + y_1 \quad (2.31)$$

Let's move on to the formulas of doubling a point over  $GF(2^n)$ , using the implicit differentiation of the elliptic curve equation (2.16):

$$2y \frac{dy}{dx} + y + x \frac{dy}{dx} = 3x^2 + 2ax \quad (2.32)$$

Applying the property of  $GF(2^n)$ , the equation is reduced to:

$$y + x \frac{dy}{dx} = x^2 \quad (2.33)$$

Note that if not the  $xy$  term in the elliptic curve equation (2.16), the implicit differentiation would be meaningless. This gives one reason why the elliptic curve equation is slightly

differently over finite field  $GF(2^n)$ . Let  $P=(x_1, y_1)$ ,  $2P=(x_3, y_3)$  and line  $L$  is the tangent line to  $P$  described by equation (2.19). The slope of the tangent line  $L$  would be:

$$\lambda = x_1 + \frac{y_1}{x_1} \quad (2.34)$$

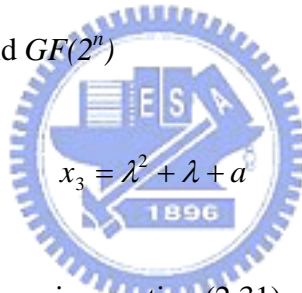
while

$$v = y_1 + \lambda x_1 \quad (2.35)$$

Following the same procedure,  $x_1$  is the two roots of equation (2.29),  $x_3$  is the other. So,

$$2x_1 + x_3 = \lambda^2 + \lambda + a$$

which  $2x_1 = 0$  over finite field  $GF(2^n)$



$$x_3 = \lambda^2 + \lambda + a \quad (2.36)$$

Finally,  $y_3$  is the same as shown in equation (2.31)

The formulas for point addition and point doubling over finite field  $GF(2^n)$  are given bellow:

Let  $P=(x_1, y_1)$ ,  $Q=(x_2, y_2)$ ,  $P+Q=(x_3, y_3)$ , elliptic curve with equation (2.16), then the point addition formula:

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1} \quad (2.37)$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$

$$y_3 = \lambda(x_3 + x_1) + x_3 + y_1$$

And the formula of point doubling, where  $P=(x_1, y_1)$ ,  $2P=(x_3, y_3)$

$$\lambda = x_1 + \frac{y_1}{x_1} \quad (2.38)$$

$$x_3 = \lambda^2 + \lambda + a$$

$$y_3 = \lambda(x_3 + x_1) + x_3 + y_1$$

## Projective Coordinates Representation

Finite field  $GF(2^n)$  inversion is relatively expensive. If inversion could be avoided while performing point addition or point doubling, then the performance of the elliptic curve cryptosystems would be improved. This is done by using projective coordinates.

Points with projective coordinates have three coordinates, for example, a projective point  $P=(X, Y, Z)$ . An affine point  $(x, y)$  corresponds to the projective coordinate point  $(x, y, 1)$ , while a projective point  $(X, Y, Z)$  could be converted into an affine point  $(X/Z, Y/Z^2)$ . Replacing  $x= X/Z$ ,  $y= Y/Z^2$  into equation (2.4), the resulting projective elliptic curve equation would be:

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \quad (2.39)$$

The formulas for adding and doubling points on elliptic will be presented here. Let  $P=(X_1, Y_1, Z_1)$ ,  $Q=(X_2, Y_2, Z_2)$ , and  $P+Q=R(X_3, Y_3, Z_3)$  are points with projective coordinates, then the formula for adding points is [4]:

$$\begin{array}{lll}
A_1 = Y_2 \cdot Z_1^2, & D = B_1 + B_2, & H = C \cdot F, \\
A_2 = Y_1 \cdot Z_2^2, & E = Z_1 \cdot Z_2, & X_3 = C^2 + H + G, \\
B_1 = X_2 \cdot Z_1, & F = D \cdot E, & I = D^2 \cdot B_1 \cdot E + X_3, \\
B_2 = X_1 \cdot Z_2, & Z_3 = F^2, & J = D^2 \cdot A_1 + X_3, \\
C = A_1 + A_2, & G = D^2 \cdot (F + aE^2), & Y_3 = H \cdot I + Z_3 \cdot J.
\end{array} \tag{2.40}$$

When  $Z_2=1$ , the formula becomes

$$\begin{array}{ll}
A = Y_2 \cdot Z_1^2 + Y_1, & Z_3 = C^2, \\
B = X_2 \cdot Z_1 + X_1, & X_3 = A^2 + D + E, \\
C = Z_1 \cdot B, & F = X_3 + X_2 \cdot Z_3, \\
D = B^2 \cdot (C + aZ_1^2), & G = (X_2 + Y_2) \cdot Z_3^2, \\
E = A \cdot C, & Y_3 = (E + Z_3) \cdot F + G.
\end{array} \tag{2.41}$$

Suppose  $P=(X_1, Y_1, Z_1)$ ,  $2P=Q=(X_2, Y_2, Z_2)$ , the doubling formula is:

$$\begin{array}{l}
Z_2 = Z_1^2 \cdot X_1^2, \\
X_2 = X_1^4 + b \cdot Z_1^4, \\
Y_2 = bZ_1^4 \cdot Z_2 + X_2 \cdot (aZ_2 + Y_1^2 + bZ_1^4).
\end{array} \tag{2.42}$$

Comparing with affine coordinates, projective coordinates doubling and adding requires more multiplications but no inversion. The performance analysis with affine coordinates doubling and adding is given below:

Table 2.3: The number of required operations for point doubling

Operations	Affine coordinates	Projective coordinates
<b>Multiplication</b>	2	4
<b>Squaring</b>	1	5
<b>Inversion</b>	1	0

Table 2.4: The number of required operations for point addition

Operations	Affine coordinates	Projective coordinates
<b>Multiplication</b>	2	13
<b>Squaring</b>	1	6
<b>Inversion</b>	1	0

Table 2.5: The number of required operations for point addition when  $Q = (X_2, Y_2, 1)$

Operations	Affine coordinates	Projective coordinates
<b>Multiplication</b>	2	8
<b>Squaring</b>	1	5
<b>Inversion</b>	1	0

The performance comparison between the two coordinates is determined by the computational complexity of the finite field inversion in affine coordinates. For example, given the table 2.3 condition and neglecting the squaring operation, the affine coordinates will outperform projective coordinates if the computational complexity of the inversion is less than 6 multiplications.



# CHAPTER 3

## Scalar Multiplication Algorithms

Scalar multiplication, given a point  $P$  on elliptic curve and a scalar  $k$  find  $kP$ , is the mainly the Elliptic Curve Cryptosystems all about. In order to compute scalar multiplication efficiently, many algorithms are proposal. The basic one is the double-and-add algorithm and halve-and-add algorithm gives an efficiently way to compute scalar multiplication by acquiring point halving. These two algorithms will be introduced in this chapter. Besides, we can apply add-and-subtract algorithm to these two algorithms to achieve a better performance.

### 3.1 Double-and-Add Algorithm

The double-and-add algorithm is the basic algorithm for calculating scalar multiplication. This algorithm is composed of point doubling and point addition. Given  $GF(2^n)$  a base point  $P$  and a scalar  $k$ , the double-and-add algorithm is:

$$\begin{aligned} k &= \sum_{i=0}^{n-1} b_i 2^i, \quad b_i \in \{0,1\} & (3.1) \\ Q &= O \\ \text{for } i &\text{ from } n-1 \text{ down to } 0 \\ \{ & \\ & Q = 2Q \\ & \text{if } b_i = 1 \text{ then} \\ & \quad Q = Q + P \\ & \} \end{aligned}$$

For example, given  $P$  and a scalar  $k=10=1010$ :

$$k = \quad \quad \quad "1 \quad 0 \quad 1 \quad 0"$$

$$Q = \quad O \quad \rightarrow P \quad \rightarrow 2P \quad \rightarrow 4P+P=5P \quad \rightarrow 10P$$

The formulas required for adding points and doubling points in the algorithms is explained in chapter 2.

### 3.2 Halve-and-Add Algorithm

The halve-and-add algorithm[5] is similar to double-and-add algorithm but the point doubling step is replaced by point halving. Next, the procedure of point halving is given.

#### Point Halving

For  $P=(x_1, y_1)$ ,  $2P=(x_3, y_3)$ , the formula of point doubling is given in equation (2.38) which is the same as:



$$\lambda = x_1 + \frac{y_1}{x_1} \tag{3.2}$$

$$x_3 = \lambda^2 + \lambda + a$$

$$y_3 = x_1^2 + x_3(\lambda + 1)$$

Point halving is the reverse of point doubling. Given an input point  $2P=(x_3, y_3)$  find  $P=(x_1, y_1)$ . In order to compute  $x_1$ , and  $y_1$ , first we have to solve  $\lambda$  from:

$$\lambda^2 + \lambda = a + x_3 \tag{3.3}$$

Where this square equation has two solutions  $\lambda$  and  $\lambda + 1$ .

Solve

$$x_1^2 = y_3 + x_3(\lambda + 1) \quad (3.4)$$

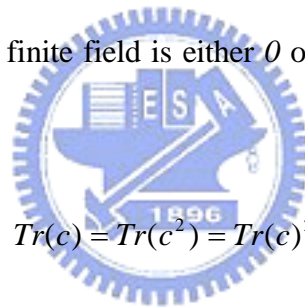
for  $x_1$ . And finally, calculate  $y_1$ :

$$y_1 = x_1(x_1 + \lambda) \quad (3.5)$$

The idea of *trace* plays an important role in deriving the algorithm for point having. Let  $c \in GF(2^n)$ , *trace* is defined as:

$$Tr(c) = c + c^2 + c^{2^2} + \dots + c^{2^{n-1}} \quad (3.6)$$

The *trace* of an element in finite field is either 0 or 1. Following are some properties of *trace*: let  $c, d \in GF(2^n)$ ,



$$Tr(c) = Tr(c^2) = Tr(c)^2 \quad (3.7)$$

*Trace* is linear:

$$Tr(c + d) = Tr(c) + Tr(d) \quad (3.8)$$

My implement uses pseudo-random curve over  $GF(2^{163})$  which has the form

$$E : y^2 + xy = x^3 + x^2 + b \quad (3.9)$$

The coefficient  $a$  in equation (2.16) is always equal to 1. So:

$$Tr(a) = 1 \quad (3.10)$$

If  $(x, y)$  is a point on elliptic curve (3.9), then:

$$Tr(x)=Tr(a) \tag{3.11}$$

The following theorem finds the correct solution of equation (3.3) while halving a point:

$$\text{Let } P=(x_1, y_1) \text{ and } 2P=(x_3, y_3). \tag{3.12}$$

Let  $\hat{\lambda}$  be a solution to (3.3) and  $t = y_3 + x_3\hat{\lambda}$ .

Suppose that  $Tr(a)=1$ . Then  $\hat{\lambda}$  is the correct solution if and only if

$$Tr(t)=0$$

We will prove the theorem. If  $\hat{\lambda}$  is a correct solution then it will satisfy equation (4.4), that is,



$$x_1^2 = y_3 + x_3(\hat{\lambda} + 1) \tag{3.13}$$

From equation (4.10) and equation (4.11)

$$Tr(y_3 + x_3(\hat{\lambda} + 1)) = Tr(x_1^2) = Tr(x_1) = Tr(a) = 1 \tag{3.14}$$

and

$$Tr(y_3 + x_3(\hat{\lambda} + 1)) = Tr((y_3 + x_3\hat{\lambda}) + x_3) = Tr(y_3 + x_3\hat{\lambda}) + Tr(x_3) = Tr(t) + 1 \tag{3.15}$$

Finally, we can get

$$Tr(t) + 1 = 1,$$

$$Tr(t) = 0$$

Else if  $\hat{\lambda}$  is not a correct solution then the correct solution must be  $\hat{\lambda}+1$ . Now  $\hat{\lambda}+1$  will satisfy equation (3.4), substitute  $\hat{\lambda}+1$  into equation (3.4)

$$x_1^2 = y_3 + x_3(\hat{\lambda} + 1 + 1) = y_3 + x_3(\hat{\lambda}) \quad (3.16)$$

Similarly,

$$Tr(t) = Tr(y_3 + x_3(\hat{\lambda})) = Tr(x_1^2) = Tr(x_1) = Tr(a) = I \quad (3.17)$$

That is, if  $Tr(t) = I$  then the correct solution is  $\hat{\lambda}+1$ .

Let the  $\lambda$ -representation of a point  $2P = (x_3, y_3)$  be  $(x_3, \lambda_3)$ , where  $\lambda_3 = x_3 + y_3/x_3$ . Let the  $\lambda$ -representation of  $2P$  as the input to point halving, then  $t$  in equation (3.12) can be computed directly from this  $\lambda$ -representation

$$t = x_3(x_3 + \lambda_3 + \hat{\lambda}) = x_3(x_3 + x_3 + \frac{y_3}{x_3} + \hat{\lambda}) = x_3(\frac{y_3}{x_3} + \hat{\lambda}) = y_3 + x_3\hat{\lambda} \quad (3.18)$$

If  $Tr(t) = 0$ ,  $\hat{\lambda}$  is the correct answer, from equation (3.13)

$$\begin{aligned} x_1^2 &= y_3 + x_3\hat{\lambda} + x_3 = t + x_3 \\ x_1 &= \sqrt{t + x_3} \end{aligned} \quad (3.19)$$

If  $Tr(t) = I$ ,  $\hat{\lambda}+1$  is the right solution, from equation (3.16)

$$\begin{aligned} x_1^2 &= y_3 + x_3\hat{\lambda} = t \\ x_1 &= \sqrt{t} \end{aligned} \quad (3.20)$$

Next is the full algorithm of point halving. The input of the algorithm is  $\lambda$ -representation  $2P = (x_3, \lambda_3)$ . The output is the  $\lambda$ -representation of  $P = (x_1, \lambda_1)$

1. Find a solution  $\hat{\lambda}$  of  $\lambda^2 + \lambda = a + x_3$  (3.21)

2. Compute  $t = x_3(x_3 + \lambda_3 + \hat{\lambda})$

3. If  $Tr(t)=0$ , then  $\lambda_1 = \hat{\lambda}$ ,  $x_1 = \sqrt{t + x_3}$

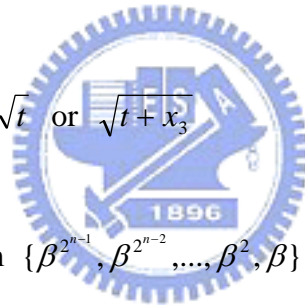
else  $\lambda_1 = \hat{\lambda} + 1$ ,  $x_1 = \sqrt{t}$

Point halving requires a multiplication and three major operations:

Solving  $\lambda^2 + \lambda = a + x_3$

Computing the trace of  $t$

Calculating a square root  $\sqrt{t}$  or  $\sqrt{t + x_3}$



Normal basis is of the form  $\{\beta^{2^{n-1}}, \beta^{2^{n-2}}, \dots, \beta^2, \beta\}$ . Let  $c$  be an element in field  $GF(2^n)$ .

By equation (2.3):

$$c = c_{n-1}\beta^{2^{n-1}} + c_{n-2}\beta^{2^{n-2}} + \dots + c_1\beta^2 + c_0\beta \tag{3.22}$$

The trace of  $c$  is

$$c = c_{n-1} + c_{n-2} + \dots + c_1 + c_0 \tag{3.23}$$

The square root equals a cyclic shift right one bit, an inverse of squaring.

$$\sqrt{c} = c_0\beta^{2^{n-1}} + c_{n-1}\beta^{2^{n-2}} + \dots + c_2\beta^2 + c_1\beta \tag{3.24}$$

### Solving the Second Degree Equation

Now deal with the solutions of the second degree equation in (3.21). Let  $c$  equation (3.22), there are two ways to solve a second degree equation as given bellow.

$$\lambda^2 + \lambda = c \quad (3.25)$$

Let

$$\lambda = \lambda_{n-1}\beta^{2^{n-1}} + \lambda_{n-2}\beta^{2^{n-2}}, \dots, \lambda_1\beta^2 + \lambda_0\beta \quad (3.26)$$

A solution is given by:

$$\lambda_0 = 0, \quad \lambda_i = \sum_{k=1}^i c_k \quad \text{for all } 1 \leq i \leq n-1 \quad (3.27)$$

These operations are expected to be inexpensive relative to normal basis multiplication.

Or we can solve equation (3.25) by half-trace

$$H(c) = c + c^{2^2} + c^{2^4} \dots + c^{2^{n-1}} \quad (3.28)$$

Substitute equation (3.28) into (3.25) and from equation (2.2) (2.5)

$$\begin{aligned} H(c)^2 + H(c) &= (c^2 + c^{2^3} + c^{2^5} + \dots + c^{2^n}) + (c + c^{2^2} + c^{2^4} \dots + c^{2^{n-1}}) \\ &= c + c^2 + c^3 + \dots + c^{2^{n-1}} + c^{2^n} = \text{tr}(c) + c \end{aligned} \quad (3.29)$$

Utilizing the above equation, we can prove that  $H(a + x_3)$  is a root of equation (3.3).

Since

$$\text{tr}(a + x_3) = \text{tr}(a) + \text{tr}(x_3) = 1 + 1 = 0 \quad (3.30)$$

As the result,

$$H(a + x_3)^2 + H(a + x_3) = tr(a + x_3) + a + x_3 = a + x_3 \quad (3.31)$$

Compare the operations of point halving and point doubling in affine and projective coordinates.

Table 3.1: Comparison between halving and doubling in affine and projective coordinates

Operations	Affine coordinates	Projective coordinates	Halving
Multiplication	2	4	1
Squaring	1	5	0
Inversion	1	0	0
Solving Second Degree Equation	0	0	1
Square Root	0	0	1
Check	0	0	1

If computation time of 1 second degree equation solving + 1 square root + 1 check is less than 3 multiplications + 5 squaring, then halving a better performance than point doubling in projective coordinates.

### Halve-and-Add Algorithm

Now we have gone through point halving. We want to employ it into scalar multiplication. Let  $GF(2^n)$ , given a point  $P$  on elliptic curve of odd order  $r$  and a scalar  $k$ . In order to compute  $kP$ , we will prove that[6]:

For every scalar  $k$ , we can find  $k'$  such that (3.32)

$$k \equiv \sum_{i=0}^{n-1} \frac{k_i'}{2^{n-1-i}} \pmod{r}$$

We will prove this by first calculating  $2^{n-1}$  multiplied by  $k$  modulo  $r$ .



$$2^{n-1}k(\text{mod } r) = \sum_{i=0}^{n-1} k'_i 2^i \quad k'_i \in \{0,1\} \quad (3.33)$$

Divide both side by  $2^{n-1}$  gives the result:

$$k(\text{mod } r) = \sum_{i=0}^{n-1} \frac{k'_i}{2^{n-1-i}} \quad k'_i \in \{0,1\} \quad (3.34)$$

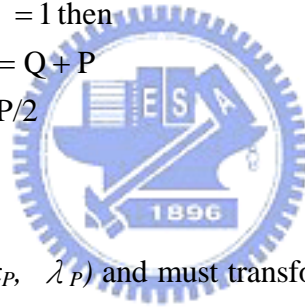
Next is a left-to-right version of the halve-and-add algorithm, where  $k$  is converted to  $k'$  by equation (3.33) first. Given  $GF(2^n)$ , the input is  $k'$  and  $P$  while the output is  $kP$ .

$$2^{n-1}k(\text{mod } r) = \sum_{i=0}^{n-1} k'_i 2^i, \quad k'_i \in \{0,1\} \quad (3.35)$$

```

Q = 0
for i from n - 1 down to 0
{
  if k'_i = 1 then
    Q = Q + P
    P = P/2
}

```



$P$  is in  $\lambda$ -representation  $(x_P, \lambda_P)$  and must transformed into affine representation  $(x_P, y_P)$  before added to  $Q$ .  $Q$  could have projective coordinates and  $Q+P$  is done by (2.41).

For example, let  $GF(2^4)$  and  $r=11="1011"$ . Given  $P$  and a scalar  $k=10$ , compute  $kP$ . First we will convert  $k$  using equation (3.33):

$$k' = 2^3 \cdot 10(\text{mod } 11) = 80(\text{mod } 11) = 3 = "0011"$$

One is required to compute the value of  $P/2(\text{mod } 11)$  in this example. We have  $2^{-1}(\text{mod } 11)=6$ , since  $6*2(\text{mod } 11)=12(\text{mod } 11)=1$ . Given any integer  $x$ ,  $x/2(\text{mod } 11)=x*6(\text{mod } 11)$ . From (3.35)

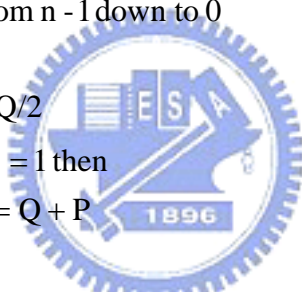
$$\begin{array}{l}
k' = \quad \text{"0} \quad 0 \quad 1 \quad 1\text{"} \\
P = \quad P \quad \rightarrow P/2=6P \quad \rightarrow 6P/2=3P \quad \rightarrow 14P/2=7P \\
Q = \quad O \quad \rightarrow O \quad \rightarrow O \quad \rightarrow O+3P=3P \quad \rightarrow 3P+7P=10P
\end{array}$$

The result is the same as the one computed from (3.1).

Another version of the halve-and-add algorithm is a right-to-left method. Point halving occurs on the accumulator  $Q$ , hence the projective coordinates is not usable.

$$2^{n-1}k \pmod{r} = \sum_{i=0}^{n-1} k_i' 2^i, \quad k_i' \in \{0,1\} \tag{3.36}$$

$Q = O$   
for  $i$  from  $n - 1$  down to  $0$   
{  
 $Q = Q/2$   
if  $k_i' = 1$  then  
 $Q = Q + P$   
}



Use the same condition  $GF(2^4)$  and  $r=11="1011"$ . Given  $P$  and a scalar  $k=10$ , that is,  $k'="0011"$ . Start from right to left

$$\begin{array}{l}
k' = \quad \text{"0} \quad 0 \quad 1 \quad 1\text{"} \\
9P/2=10P \leftarrow \quad 7P/2=9P \leftarrow \quad P/2+P=6P+P=7P \leftarrow \quad O+P=P \leftarrow \quad O = Q
\end{array}$$

And the final answer is  $10P$ . Unlike algorithm (3.35), here only requires one register for  $Q$ .

### 3.3 Add-and-Subtract Algorithm

We can further encode the scalar  $k$  or  $k'$  of the halve-and-add algorithm when computing  $kP$  to reduce the Hamming weight of  $k$  or  $k'$ , hence reduce the amount of point additions. Since point addition is more expensive than point doubling or halving, the performance of scalar multiplication is improved. Add-and-subtract algorithm [2] eliminates the situation of continuous 1's by combinations of additions and subtractions. Given an  $n$ -bit scalar  $k$

$$k = \sum_{i=0}^n e_i 2^i \quad e_i \in \{-1, 0, 1\} \quad (3.37)$$

Using add-and-subtract algorithm, we find  $m$ :

Let  $k_{n-1}k_{n-2}\dots k_1k_0$  be the binary representation of  $k$ , (3.38)

Let  $h_n h_{n-1} \dots h_1 h_0$  be the sum of  $k_{n-1}k_{n-2}\dots k_1k_0 + k_{n-1}k_{n-2}\dots k_1$

Let  $g_n g_{n-1} \dots g_1 g_0$  equals to  $00k_{n-1}k_{n-2}\dots k_1$

for  $i$  from 0 to  $n$

{

if  $h_i=1$  and  $g_i=0$ , then  $e_i=1$

else if  $h_i=0$  and  $g_i=1$ , then  $e_i=-1$

else  $e_i=0$

}



Take  $k=29="11101"$  for example.  $h="11101"+"1110"="101011"$

$$h = \text{"1 0 1 0 1 1"}$$

$$g = \text{"0 0 1 1 1 0"}$$

$$e = \text{"1 0 0 -1 0 1"}$$

It's easy to verify that:

$$k = 1 \cdot 2^5 - 1 \cdot 2^2 + 1 = 32 - 4 + 1 = 29$$

Combine add-and-subtract algorithm with (3.35):

$$2^{n-1} k(\text{mod } r) = \sum_{i=0}^{n-1} k_i 2^i, \quad k_i \in \{0,1\} \tag{3.39}$$

$$= \sum_{i=0}^n e_i 2^i, \quad e_i \in \{-1,0,1\}$$

$Q = 0$   
 for i from n down to 0  
 {  
   if  $e_i = 1$  then  
      $Q = Q + P$   
   else if  $e_i = -1$  then  
      $Q = Q - P$   
    $P = P/2$   
 }

$-P$  is given by (2.36). Combining add-and-subtract algorithm with (3.1) or (3.36) will do too.

# CHAPTER 4

## Implementation Results and Comparisons

My implementation uses pseudo-random curve of the form in normal basis over  $GF(2^{163})$

$$y^2 + xy = x^3 + x^2 + b \quad (4.1)$$

The normal basis is of type 4 which is not optimal normal basis. The base point  $P=(P_x, P_y)$

$$P_x = x_{162}\beta^{2^{162}} + x_{161}\beta^{2^{161}}, \dots, x_1\beta^2, x_0\beta \quad (4.2)$$

$$P_y = y_{162}\beta^{2^{162}} + y_{161}\beta^{2^{161}}, \dots, y_1\beta^2, y_0\beta \quad (4.3)$$

Express  $P_x$  and  $P_y$  as 163bit numbers  $x_{162}x_{161}, \dots, x_1x_0$  and  $y_{162}y_{161}, \dots, y_1y_0$ . Their value in hexadecimal equals

$$P_x=0\_bb95\_2eb0\_8fc0\_b1c8\_699f\_739a\_9357\_3474\_1e04\_4460 \quad (4.4)$$

$$P_y=7\_f185\_6ef0\_98cf\_adc8\_077e\_e437\_33a7\_f113\_1e41\_ae66 \quad (4.5)$$

If  $P$  is in  $\lambda$ -representation, then

$$P_\lambda=3\_e6c0\_a681\_341a\_b0a3\_6cc5\_c338\_7bff\_ea7e\_014f\_a6a3 \quad (4.6)$$

The value of coefficient  $b$  in equation (4.1) is

$$b=6\_fcde\_3c9e\_f967\_437b\_e459\_b1ce\_438e\_3479\_a9e7\_d133 \quad (4.7)$$

The base point  $P$  has order  $r$ .  $r$  is a large prime number with value in decimal

$$r=5846006549323611672814742442876390689256843201587 \quad (4.8)$$

The number of points on elliptic curve is  $2r$ .

The fundamental element of the entire circuits is the  $GF(2^{163})$  normal basis serial multiplier. Let the inputs equal (2.5) and output equals (2.6). Using the algorithm in [2], derive the product.

$$c_0 = a_1(b_0 + b_{13} + b_{132} + b_{117}) + a_2(b_{117} + b_{92} + b_{111} + b_{145}) + \dots \quad (4.9)$$

The formulas for other coordinates can be derived from above:

$$c_1 = a_2(b_1 + b_{14} + b_{133} + b_{118}) + a_3(b_{118} + b_{93} + b_{112} + b_{146}) + \dots$$

$$c_2 = a_3(b_2 + b_{15} + b_{134} + b_{119}) + a_4(b_{119} + b_{94} + b_{113} + b_{147}) + \dots$$



We can implement this using three register to store input  $A$ ,  $B$ , and output  $C$ . Implement equation (4.9) and cyclic shift these three register by one bit at each cycle. The product is generated bit by bit. The circuit diagram is given bellow:

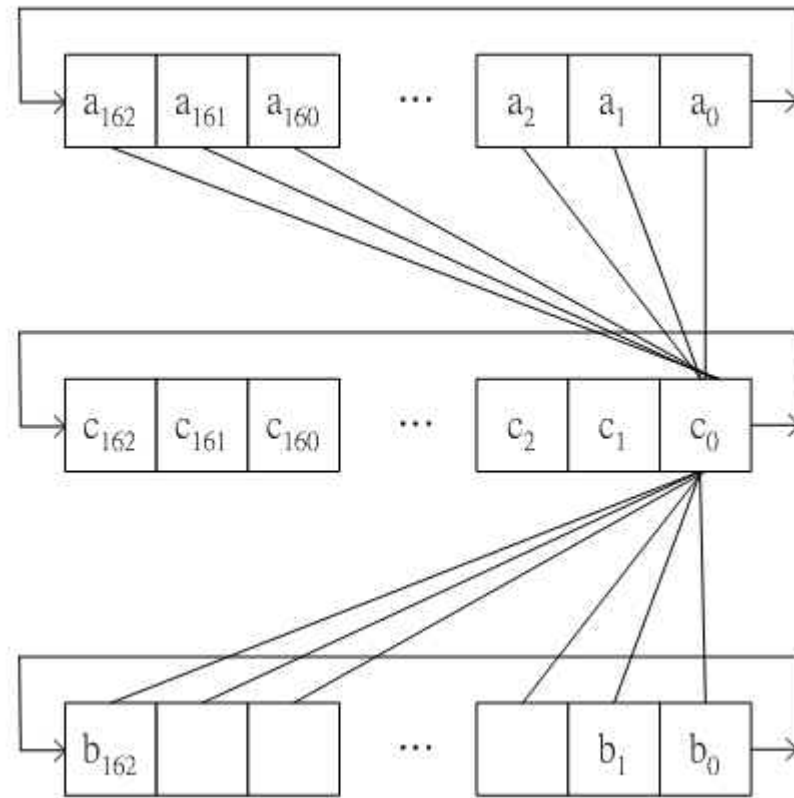


Figure 4.1: Normal Basis Multiplier version 1

The combinational circuit of the input of  $c_0$  is concealed. Only the idea of connection is given. The latency of this multiplier is 163 cycles and  $c_0$  has a larger fan-in. We can modify the above multiplier by adding one term at one cycle[9]. For example:

$$c_2 = c_1 + a_1(b_0 + b_{13} + b_{132} + b_{117})$$

$$c_3 = c_2 + a_4(b_{119} + b_{94} + b_{113} + b_{147})$$

⋮

The following is the multiplication cell for adding one term at each cycle:

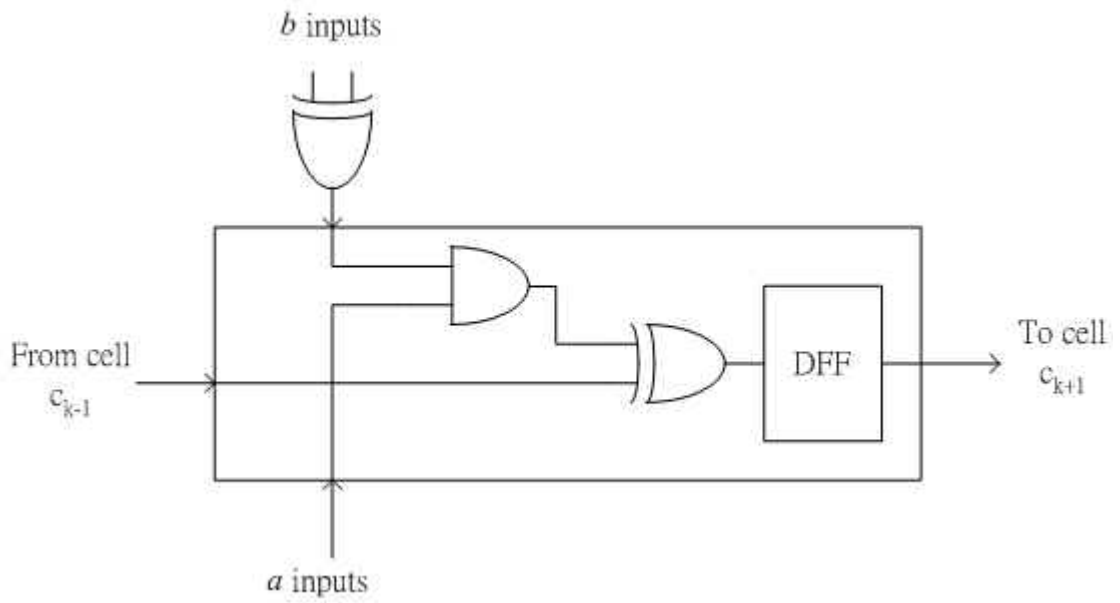


Figure 4.2: Multiplier element of  $c_k$

Modify the original multiplier we'll get:

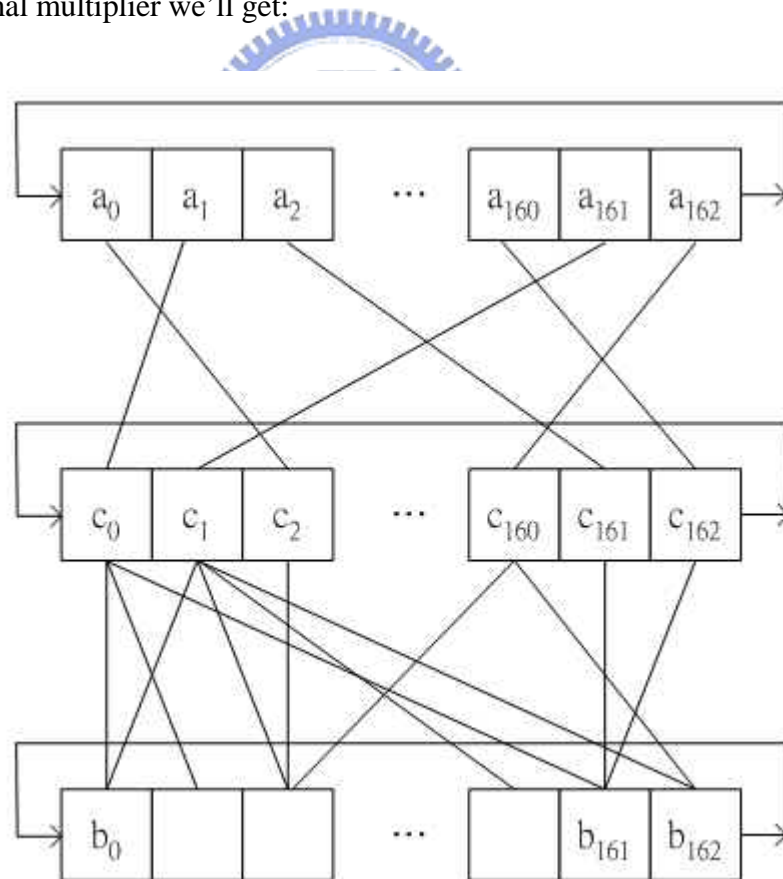


Figure 4.3: Normal basis multiplier version 2



This is a conceptual diagram showing the difference of wiring. The fan-in of the output register is reduced. Another benefit of this multiplier is that we could set the register of  $C$  to a value say  $D$  at beginning. Then the final output will equal  $A*B+D$  equivalent to the effect of a MAC, multiplication-and-accumulator.

The solution of the second degree equation is given by equation (3.27). This can be easily implemented using a one bit register and an exclusive-or. Since the solution is given out serially, we can modify the above multiplier by adding each  $a_i$  term of the product at each cycle. For example,

$$c_0 = c_1 + a(b_{13} + b_{117} + b_0 + b_{132})$$

$$c_1 = c_2 + a(b_{111} + b_{145} + b_{117} + b_{92})$$



Use similar cells in Figure 4.2, the new normal basis multiplier is

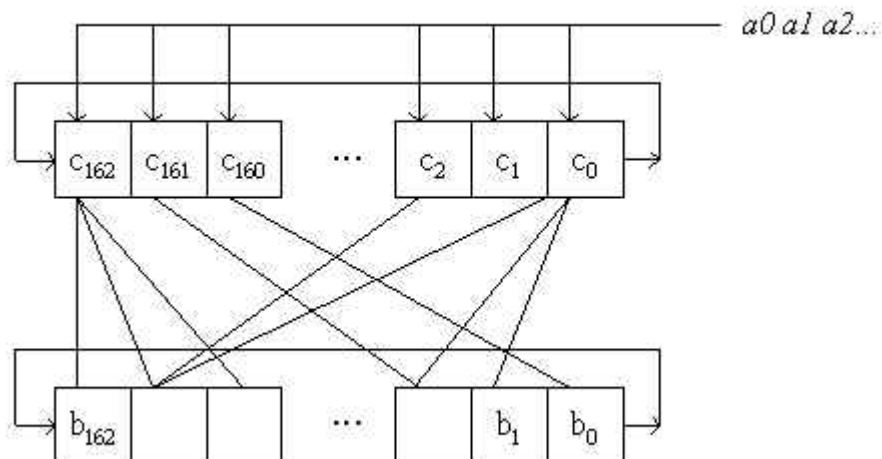


Figure 4.4 serial input normal basis multiplier

Combine the solution circuit with the serial input normal results an efficient implementation for point halving.

The input of point halving is in  $\lambda$ -representation. For the implementation of point halving, a normal basis multiplier is used. The second degree equation is solved by half-trace as given by equation (3.31). Trace  $t$  is given by exclusive-or every bit of  $t$ . Since only one multiplier is required, the over all latency is 163 cycles. The architecture of point halving is given bellow. Let  $2P=(x_3, \lambda_3)$ , the output is  $P=(x_1, \lambda_1)$

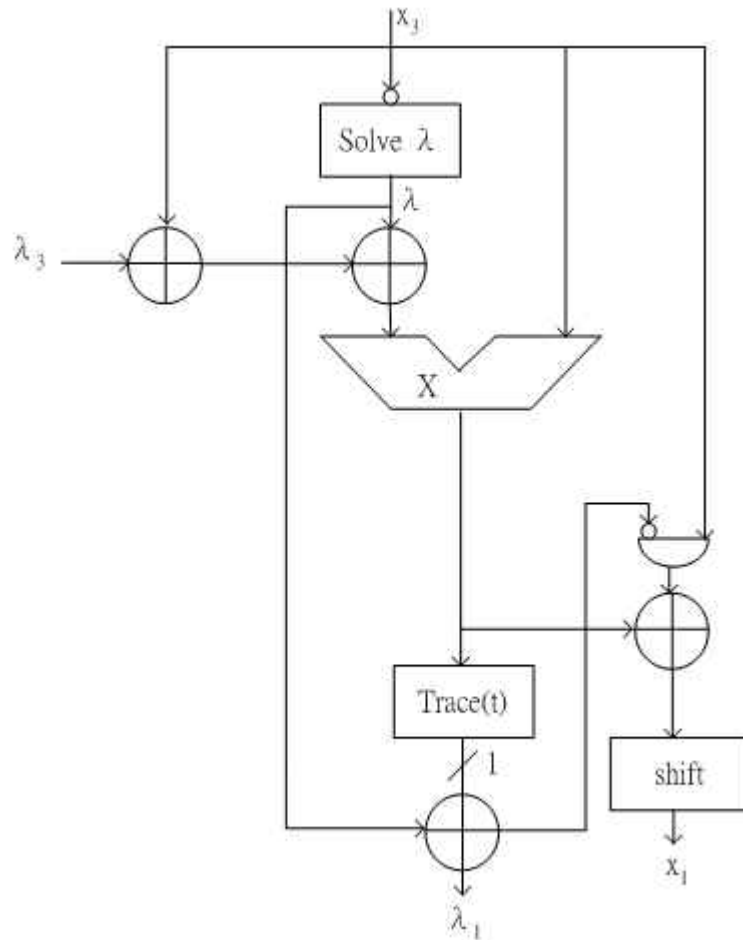


Figure 4.5: Circuit for point halving

The procedure of point halving is:

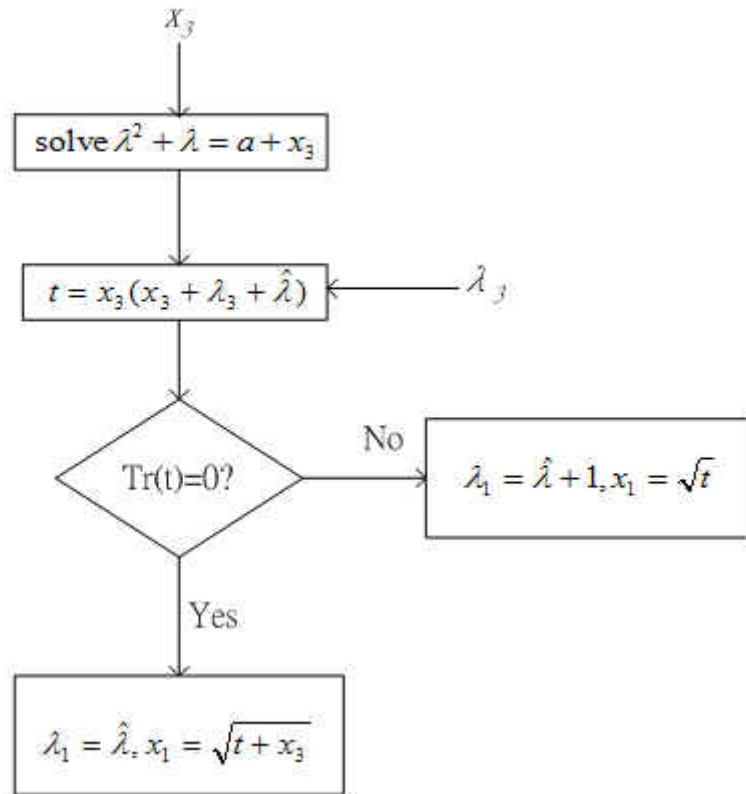


Figure 4.6: Point halving flow

The coefficient  $a$  of pseudo-random is always equal to one. One or the multiplication identity in normal basis is a number where every bit of it is 1. The right hand side of equation (3.3) equals:

$$a + x_3 = 1 + x_3 = \bar{x}_3$$

That is, exclusive-or each bit of  $x_3$  with 1 is the same as inverting each bit.

In order to implement scalar multiplication efficiently, algorithm (3.39) is chosen. Since the point addition in projective coordinates requires no inversion, we let the accumulator  $Q$  of (3.39) in projective coordinates. The point addition  $Q+P$  or  $Q-P$  has  $Q$  in projective coordinates and  $P$  in  $\lambda$ -representation  $P=(X_I, \lambda_I)$ . From (3.5) we modify formula (2.41) as:

$$\begin{aligned}
Y_2 &= X_2(X_2 + \lambda_2) & Z_3 &= C^2, \\
A &= Y_2 \cdot Z_1^2 + Y_1, & X_3 &= A^2 + D + E, \\
B &= X_2 \cdot Z_1 + X_1, & F &= X_3 + X_2 \cdot Z_3, \\
C &= Z_1 \cdot B, & G &= (X_2 + Y_2) \cdot Z_3^2, \\
D &= B^2 \cdot (C + aZ_1^2), & Y_3 &= (E + Z_3) \cdot F + G. \\
E &= A \cdot C,
\end{aligned} \tag{4.10}$$

My implementation of (2.41) contains three multipliers. Due to the data dependency, the data calculated at each multiplication is arranged as follow with minimum latency. The data dependency is indicated.

Table 4.1: The data flow of mix-coordinates addition (5.10)

Multiplications	Multiplier 1	Multiplier 2	Multiplier 3	Output
1		$B = X_2 \cdot Z_1 + X_1$	$Y_2 = X_2(X_2 + \lambda_2)$	
2	$C = Z_1 \cdot B$		$A = Y_2 \cdot Z_1^2 + Y_1$	$Z_3 = C^2$
3	$D = B^2 \cdot (C + aZ_1^2)$	$E = A \cdot C$	$F = X_3 + X_2 \cdot Z_3$	$X_3 = A^2 + D + E$
4	$Y_3 = (E + Z_3) \cdot F + G$	$G = (X_2 + Y_2) \cdot Z_3^2$		$Y_3$

As we can see from the above table, the timing of this mix-coordinates addition equals to 4 multiplications which is 4\*163 cycles.

The following is the circuit diagram of the mix-coordinates addition. The multiplier in the diagram has three inputs where two are from multiplication and one for accumulation. The *neg* signal is for adding  $-P$  to  $Q$ . The *ini* signal indicates the initial condition when  $O+P=P$ . That is,  $X_3=X_2$ ,  $Y_3=Y_2$  or  $X_2+Y_2$ , and  $Z_3=1$

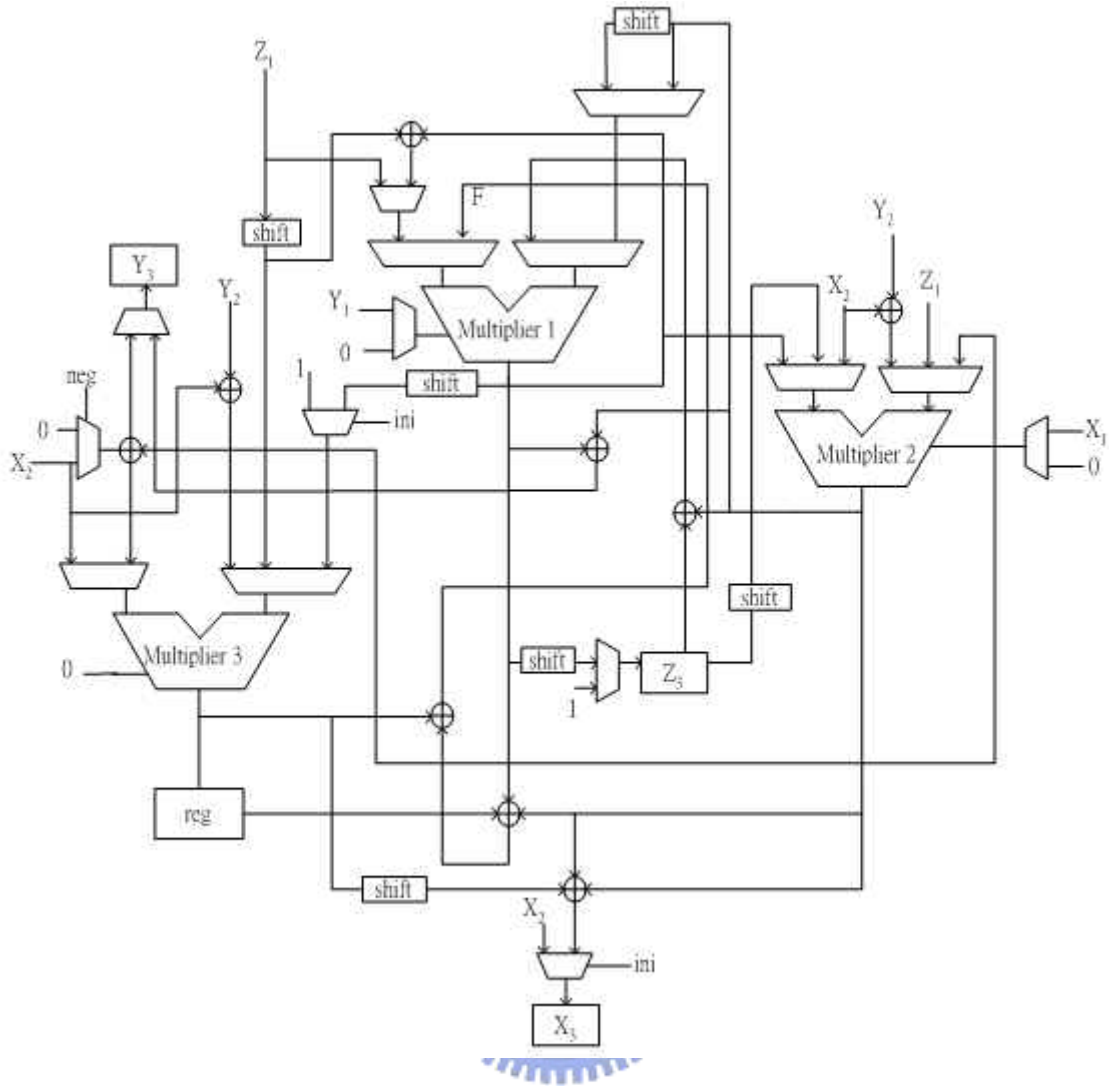


Figure 4.7: Circuit for mix-coordinates addition

My proposed design is a scalar multiplication circuit based on algorithm (3.39). It is composed of the point halving circuit and the point adding circuit plus some control signals. The inputs are  $k'$  which is derived from  $k$  as shown in (3.33) and base point  $P$ . The output is  $kP$ .  $k'$  is first encoded into  $e$  as in (3.39). From (3.38), the implementation of the encoding logic uses two shift register to store  $g$  and  $h$ . The shift registers shift one bit every one point halving complete. We observe the msb of the  $g$  and  $h$  registers to decide whether the input to the point addition circuits is  $P$  or  $-P$ . Since there are separate registers for the accumulator  $Q$  and  $P$ , the halving circuit and adding circuits can process at the same time. This makes computation more efficient. When  $e_i$  is nonzero or the MSB of the  $g$  and  $h$  registers are

different, the halving circuit must hold its output until the adding circuit reads the result. The point addition circuit adds  $P$  or  $-P$  to the accumulator when  $e_i$  is 1 or -1. The control flow of the whole circuit is:

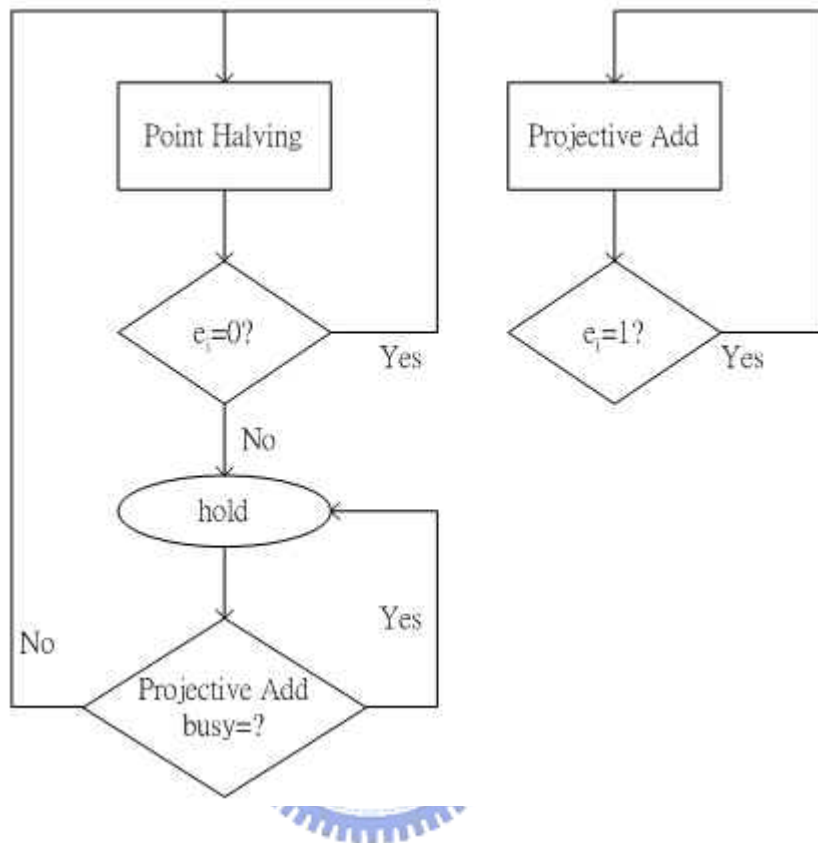


Figure 4.8: The control of point halving and projective addition

The synthesized result is given below. The cycle time is set to 5ns and the synthesis standard library is 0.18  $\mu m$  technology.

Table 4.2: The synthesized results

<b>Circuits</b>	<b>Gate Counts</b>
<b>Multiplier</b>	6961
<b>Halving</b>	14321
<b>Addition</b>	45723
<b>Scalar multiplication</b>	77100

The average latency of scalar multiplication is about 37000 cycles and frequent 200Mhz. So the throughput is  $2^{*}163^{*}200\text{Mhz}/37000=1.76\text{Mbit/s}$

The verification is given by an integrated FPGA system called iProve. This system allows displaying the outputs from FPGA on ModelSim directly. The FPGA chip is Xilinx Virtex2: XC2V8000. The synthesis frequency is set to 90Mhz and the total LUTs is 8815.

The table bellow lists a comparison of the Elliptic Curve Cryptosystems implementation. We can see that our design has about the same throughput as [12] while the area is smaller.

Table 4.3: The performance comparison of Elliptic Curve Cryptosystems implementations on ASIC

Authors	Huang [10]	Okada [11]	Bai [12]	Daneshbeh [13]	Sozzani [14]	Proposed
Technology	0.35 $\mu m$	0.25 $\mu m$	0.18 $\mu m$	0.18 $\mu m$	0.13 $\mu m$	0.18 $\mu m$
Field	GF(2 <sup>251</sup> )	GF(2 <sup>163</sup> )	GF(2 <sup>233</sup> )	GF(2 <sup>163</sup> )	GF(2 <sup>163</sup> )	GF(2 <sup>163</sup> )
Gate counts	56K	165K	120K	74K	?	77K
Clock rate	100Mhz	66Mhz	100Mhz	700Mhz	400Mhz	200Mhz
Latency for kP (cycles)	?	?	?	212,552	11,320	37,000
Processor	Y	Y	N	Y	Y	N
Algorithm for kP	Montgomery (affine)	?	Montgomery	Double -and -Add (serial)	Montgomery (parallel)	Halve -and- Add
Basis	Poly	Poly	Poly	Poly	Poly	Normal
Throughput	91Kb/s	501Kb/s	1.86Mb/s	1.1Mb/s	12Mb/s	1.76Mb/s

Table 4.4: The performance comparison of Elliptic Curve Cryptosystems implementations on FPGA

Authors	Orlando & Paar[15]	Gura[16]	Lutz[17]	Proposed
Platform	Xilinx XCV400E	Xilinx XCV2000E	Xilinx XCV2000E	Xilinx XC2V8000
Technology	0.18 $\mu m$	0.18 $\mu m$	0.18 $\mu m$	0.15/0.12 $\mu m$
Field	2 <sup>167</sup>	2 <sup>163</sup>	2 <sup>163</sup>	2 <sup>163</sup>
LUTs	3002	19508	10017	8815
FFs	1769	6442	1930	N/A
Processor	Y	Y	Y	N
Clock rate	76Mhz	66Mhz	66Mhz	90Mhz
Algorithm for kP	Montgomery	Montgomery	$\tau$ -NAF	Halve-and-Add
Basis	Poly	Poly	Poly	Normal
Throughput	1.5Mb/s	2.2Mb/s	4.3Mb/s	792kb/s



# CHAPTER 5

## Conclusion

In this paper, an implementation of Elliptic Curve Cryptosystems is shown. The architecture uses point halving to reduce the computation complexity. Point halving only requires one multiplier and some addition circuits. We can replace double-and-add algorithm by halve-and-add algorithms.

The normal basis multiplier in the implementation is a serial multiplier. The projective addition circuit contains three multiplier and the timing equals to 4 times the timing of a multiplier and no inversion over finite field is required. The input is encoded as for the use of halve-and-add. We can further reduce the Hamming weight of the input, using add-and-subtract algorithm. The halving circuit and projective addition circuit can work in parallel under certain condition when the data have no dependency.

The implementation is synthesized using synthesis library of  $0.18\ \mu m$  technology. We use Xilinx Virtex2 (XC2V8000) to verify the implementation.

# APPENDIX

## Elliptic Curve Cryptosystems

In elliptic curve cryptosystems, we need to map a message onto a point on an elliptic curve. Then elliptic curve cryptosystems operate on that point to yield a new point that serves as the ciphertext. The idea of the mapping method is the following. Let equation (2.15) be the elliptic curve. The message  $m$  will be assign as the  $x$ -coordinates of a point first. However, there is only  $1/2$  chance that there exist a solution  $y$  such that

$$y^2 \equiv x^3 + am + b \pmod{p} \quad (\text{a.1})$$

Therefore, we append a few bits at the end of  $m$ , and try every pattern of these bits until there is a solution for equation (a.1). Namely, let  $K$  be a large integer so that when trying to map a message as a point on elliptic curve the failure rate of  $1/2^K$  is low. Suppose that

$$(m+1)K < p \quad (\text{a.2})$$

Represent the message  $m$  as

$$x = mK + j, \text{ where } 0 \leq j < K \quad (\text{a.3})$$

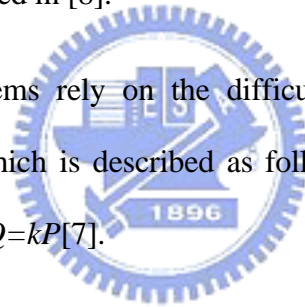
For  $j=0, 1, \dots, K-1$ , try to a solution  $y$  from (a.1). If a solution  $y$  exists, then message  $m$  is mapped to  $P_m=(x, y)$  and we can stop trying. Otherwise, increase  $j$  by one and use this new  $x$  to find a solution again. If we can't found any solution for  $j=0$  to  $K-1$ , then we failed to map message  $m$  to a point. Maybe we should pick a larger integer for  $K$  and start all over again. Since for each  $j$ , the probability of finding a solution is  $1/2$ , we have  $1/2^K$  chance of failure. Finally, the encoded message can be recovered from the point  $P_m=(x, y)$  by

$$m = \lfloor x / K \rfloor \quad (\text{a.4})$$

For example, let message  $m=5$ ,  $p=179$  and elliptic curve be  $y^2 = x^3 + 2x + 7$ . Pick  $K=10$ , so the failure rate is  $1/2^{10}$ , which is acceptable.  $x=mK+j=50+j$ ,  $x=50, 51, \dots, 59$ . For  $x=51$  we get  $x^3+2x+7=121(\text{mod } 179)$ , thus  $y=11$ . The message  $m$  is mapped to point  $(51, 11)$  and can be recover by  $m = \lfloor 51/10 \rfloor = 5$ .

For elliptic curve over  $GF(2^n)$  of the form (2.16). The steps of representing message  $m$  are the same. Let message  $m$  has  $t$ -bit, we append  $u$ -bit number  $j$  to the end of  $m$  and  $t+u \leq n$ . The message  $m$  will be represented as  $x=m2^u+j$ . For  $j=0, 1, \dots, 2^u-1$ , try to find a solution  $y$  from (2.16). If a solution is found we take  $P_m=(x, y)$ , else increase  $j$  and try again. Solving  $y$  from (2.16) given  $x$  is explained in [8].

Elliptic Curve Cryptosystems rely on the difficulty of solving the discrete logarithm problem for elliptic curves, which is described as follow. Suppose  $P, Q$  are two points on elliptic curve, find  $k$  such that  $Q=kP$ [7].



### ***a.1 Elliptic Curve ElGamal Cryptosystem***

An Elliptic Curve ElGamal Cryptosystem, a public key system, is one popular application of elliptic curve cryptography. One uses public key to encrypt plaintext and use private key to decrypt ciphertext. Let's take a look at this cryptosystem. Alice wants to send a message to Bob, so Bob chooses an elliptic curve (2.15), where  $p$  is a large prime. He also chooses a point  $P$  and a scalar  $k$ , which is the private key. He computes

$$Q = kP \quad (\text{a.5})$$

The point  $Q$  and  $P$  are public keys of Bob. Alice represents her message as a point  $x$  on elliptic curve (2.15). She also chooses a private integer  $a$ , and computes. The add and subtracts here are point operations.

$$y_1 = aP \quad \text{and} \quad y_2 = x + aQ \quad (\text{a.6})$$

She sends  $y_1$  and  $y_2$  to Bob. Bob can decrypt  $x$  by calculating

$$y_2 - ky_1 = (x + aQ) - kaP = x + akP - kaP = x \quad (\text{a.7})$$

Next is a example of Elliptic Curve ElGamal Cryptosystem. Let the point  $P=(4,11)$  and elliptic curve  $y^2 \equiv x^3 + 3x + 45(\text{mod } 8831)$ . The message of Alice is represented as point  $P_m=(5, 1743)$ . She wants to send the message to Bob.

Bob has a private key  $k=3$  and computes  $Q=kP=(413, 1808)$ .  $Q$  is made public. Alice takes Bob's public key  $Q$ . She chooses a random number  $a=8$ . She computes  $y_1=aP=(5415, 6321)$  and  $y_2=P_m+aQ=(6626,3576)$  and sends  $(y_1, y_2)$  to Bob. Bob wants to decrypt  $(y_1, y_2)$ . Bob first calculates  $ky_1=3(5415, 6321)=(673, 146)$  and subtracts this from  $y_2$

$$(6626, 3576)-(673,146)=(6626, 3576)+(673,-146)=(5,1743)$$

## ***a.2 Elliptic Curve Diffie-Hellman Key Exchange***

Another useful system is the Elliptic Curve Diffie-Hellman Key Exchange, which can be used for key exchange for private key system. Alice and Bob want to exchange a key. They choose a base point  $P=(3,5)$  on an elliptic curve  $E: y^2 \equiv x^3 + x + 7206(\text{mod } 7211)$ . Alice chooses a random integer  $a=12$  and bob choose  $b=23$ . The compute  $aP$  and  $bP$  and make them public.

$$aP=(1794,6375) \text{ and } bP=(3861, 1242)$$

Alice take  $bP$  and multiply by  $a$  to get the key

$$a(bP)=12(3861, 1242) =(1472,2098)$$

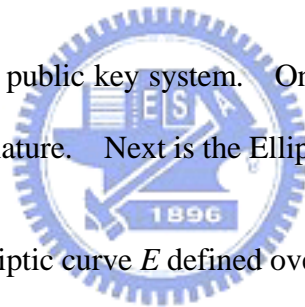
In the same way, Bob takes  $aP$  and compute  $b(aP)$

$$a(bP)=12(3861, 1242) =(1472,2098)$$

Now they have the same key.

### ***a.3 Elliptic Curve Digital Signature Algorithm***

Signature is the opposite of public key system. One use the private to sign and others use the public key to verify the signature. Next is the Elliptic Curve Digital Signature Algorithm:



Let  $p$  be a prime and let elliptic curve  $E$  defined over  $GF(p)$ . (a.8)

$A$  is a point on  $E$  having prime order  $q$  and define:

$$K=(p, q, E, P, m, Q), \text{ where } Q=mP$$

$p, q, E, P$  and  $Q$  are public key and  $m$  is the private key

$K=(p, q, E, P, m, Q)$  and  $k$  is a random number, define

$$sig_K(x, k)=(r, s),$$

where

$$kP=(u, v)$$

$$r=u \text{ mod } q, \text{ and}$$

$$s = k^{-1}(\text{SHA-1}(x) + mr) \bmod q$$

Verification is given bellow:

$$w = s^{-1} \bmod q$$

$$i = w \text{SHA-1}(x) \bmod q$$

$$j = wr \bmod q$$

$$(u, v) = iP + jQ$$

$\text{ver}_K(x, (r, s))$  is true if and only if

$$u \bmod q = r$$

Let  $E: y^2 \equiv x^3 + x + 6 \pmod{11}$  and  $p=11, q=13, P=(2,7), m=7$  and  $Q=(7,2)$ . Suppose message  $x$  and  $\text{SHA-1}(x)=4$ , Alice sign the message with random value  $k=3$ . She computes:

$$(u, v) = 3(2, 7) = (8, 3)$$

$$r = u \bmod 13 = 8, \text{ and}$$

$$s = 3^{-1}(4 + 7*8) \bmod 13 = 7$$

$(8, 7)$  is the signature.

Bob verifies the signature by

$$w = 7^{-1} \bmod 13 = 2$$

$$i = 2*4 \bmod 13 = 8$$

$$j = 2*8 \bmod 13 = 3$$

$$(u, v) = 8P + 3Q = (8, 3), \text{ and}$$

$$u \bmod 13 = 8 = r.$$

Then the signature is verified.



# BIBLIOGRAPHY

- [1] “Certicom ECC FAQ”, [http://www.certicom.com/index.php?action=ecc,ecc\\_faq](http://www.certicom.com/index.php?action=ecc,ecc_faq)
  
- [2] IEEE Std 1363-2000, *IEEE standard specifications for public-key cryptography*, IEEE Computer Society, August 29, 2000.
  
- [3] Douglas R. Stinson, *Cryptography: Theory and Practice - Second edition*, Chapman & Hall/CRC , 2002
  
- [4] J. Lopez and R. Dahab, “Improved algorithms for elliptic curve arithmetic in  $GF(2^n)$ ”, *Selected Areas in Cryptography - SAC '98*, LNCS **1556**, 1999, 201-212.
  
- [5] K. Fong, D. Hankerson, J. Lopez, and A. Menezes. “Field Inversion and Point Halving Revisited”. *IEEE Transactions on Computers*, 53(8):1047-1059, August 2004.
  
- [6] E. Knudsen, “Elliptic scalar multiplication using point halving”, *Advances in Cryptology - Asiacrypt '99*, LNCS **1716**, 1999, 135-149.
  
- [7] W. Trappe and L.C. Washington: *Introduction to Cryptography with Coding Theory*, Prentice Hall, 2001.
  
- [8] A. X9.62. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1998.
  
- [9] Philip H. W. Leong and Ivan K. H. Leung. “A microcoded elliptic curve processor using FPGA technology”. *IEEE Transactions on VLSI Systems*, 10(5), October 2002.
  
- [10] Chi Huang, Jimmei Lai, Junyan Ren, and Qianling Zhang, “Scalable Elliptic Curve Encryption Processor for Portable Application,” 5<sup>th</sup> Int. Conf. ASIC, pp. 1312-1316, Oct. 2003.



- [11] Souichi Okada, Naoya Torii, Kouichi Itoh, and Masahiko Takenaka. "Implementation of elliptic curve cryptographic coprocessor over  $GF(2^m)$  on an FPGA." In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 25–40. Springer-Verlag, 2000.
- [12] Guoqiang Bai, Zhun Huang, Hang Yuan, Hongyi Chen, Ming Liu, Gang Chen, Tao Zhou, and Zhihua Chen. "A high performance VLSI chip of the elliptic curve cryptosystems," 7<sup>th</sup> Int. Conf. SICT, pp. 2059-2062, Oct. 2004
- [13] A. Daneshbeh, M. Hasan, "Area Efficient High Speed Elliptic Curve Coprocessors for Random Curves," Proceedings of ITCC 04, Las Vegas, NE, USA, 2004
- [14] F. Sozzani, G. Bertoni, S. Turcato, L. Breveglieri, "A parallelized Design for an Elliptic Curve Cryptosystem Coprocessor" Proceedings of ITCC 05, 2005.
- [15] G. Orlando and C. Paar. "A high-performance reconfigurable elliptic curve processor for  $GF(2^m)$ ." In *Cryptographic Hardware and Embedded Systems (CHES)*, 2000.
- [16] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila. "An end-to-end systems approach to elliptic curve cryptography." In *Cryptographic Hardware and Embedded Systems (CHES)*, 2002.
- [17] J. Lutz, A. Hasan., "High Performance FPGA based Elliptic Curve Cryptographic Co-Processor". Proceedings of ITCC 04, Las Vegas, NE, USA, 2004