SCALAR MULTIPLICATION ON ELLIPTIC CURVES

OĞUZ YAYLA

AUGUST 2006

SCALAR MULTIPLICATION ON ELLIPTIC CURVES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

OĞUZ YAYLA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF CRYPTOGRAPHY

AUGUST 2006

Approval of the Graduate School of Applied Mathematics

Prof. Dr. Ersan AKYILDIZ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. F. Rüyal ERGÜL
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Ersan AKYILDIZ
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Ali DOĞANAKSOY (METU, MATH) _____

Prof. Dr. Ersan AKYILDIZ (METU, MATH) _____

Prof. Dr. Hurşit ÖNSİPER (METU, MATH) _____

Assoc. Prof. Dr. Melek D. YÜCEL (METU, EEE) _____

Dr. Muhiddin UĞUZ (METU, MATH) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name, Last name : Oğuz YAYLA
Signature :

Abstract

SCALAR MULTIPLICATION ON ELLIPTIC CURVES

Oğuz YAYLA
M.Sc., Department of Cryptography
Supervisor: Prof. Dr. Ersan AKYILDIZ

August 2006, 63 **pages**

Elliptic curve cryptography has gained much popularity in the past decade and has been challenging the dominant RSA/DSA systems today. This is mainly due to elliptic curves offer cryptographic systems with higher speed, less memory and smaller key sizes than older ones. Among the various arithmetic operations required in implementing public key cryptographic algorithms based on elliptic curves, the elliptic curve scalar multiplication has probably received the maximum attention from the research community in the past a few years. Many methods for efficient and secure implementation of scalar multiplication have been proposed by many researchers. In this thesis, many scalar multiplication methods are studied in terms of their mathematical, computational and implementational points.

Keywords: Elliptic Curves, Group Law, Endomorphisms, NAF, Scalar Multiplication.

Öz

ELİPTİK EĞRİLER ÜZERİNDE KATSAYI ÇARPIMI

Oğuz YAYLA
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi: Prof. Dr. Ersan AKYILDIZ

Ağustos 2006, 63 **sayfa**

Geçen 10 yıllık sürede eliptik eğri kriptosistemleri önemli ölçüde yaygınlaştı ve artık RSA/DSA sistemlerine meydan okur hale geldi. Bunun nedeni, eliptik eğri kriptosistemlerinin diğerlerine oranla daha yüksek hızda, daha düşük hafıza isteyen ve daha düşük anahtar boyunda kriptosistemleri gerçekleştirebilme-sidir. Eliptik eğri kriptosistemlerinde gereksinim duyulan aritmetik işlemler ara-sında, araştırmacılar nezdinde son bir kaç yıldır en çok ilgi çeken katsayı çarpımı olmuştur. Etkili ve güvenli katsayı çarpım metodları araştırmacılar tarafından önerilmiştir. Bu tezde, bir çok katsayı çarpımı metodu matematiksel, sayısal ve uygulamaya dönük çalışılmıştır.

Anahtar Kelimeler: Eliptik Eğriler, Grup Kuralları, Endomorfizmalar, NAF, Katsayı Çarpımı.

To Saniye and to My Parents...

Acknowledgments

I acknowledge financial support from TÜBİTAK-BAYG through my undergraduate and graduate academic life up till now.

I would like to express my sincere gratitude to my supervisor, Prof. Dr. Ersan Akyıldız, for his guidance, encouragement and patience throughout of this thesis.

Thanks to my friends who have always been up not also for support but also for a good laugh.

A special thanks to Murat Yayla, Fikret Yayla and Mehmet Sarıer for the support of them throughout of my life.

And, a huge thanks to my family: to Mum and Dad - thank you for all your support over the years; to Selim and Gözde, for being there no matter what.

Last, but not least, Saniye, gave me lots of emotional support, without which I could not have succeeded.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a few decades ago, cryptography was known to be the study of providing secure communications over an insecure channel, so that only a set of intended recipients can understand the message. However, from the todays' point of view, the definition of cryptography needs to be extended as designing of algorithms, protocols and systems which are used to protect information against threats. Information protection mainly consists of privacy (confidentiality), data integrity, authentication, verifiability and non-repudiation, which are also known as main goals of cryptography.

In order to satisfy privacy, one needs to convert the information (message, plaintext) to some unintelligible language (cryptogram, ciphertext, codeword) by using some secret data (cryptographic key), which is so called encryption operation. Receiver of the ciphertext takes it back to a plaintext only by correct secret key, which is so called decryption operation.

Algorithms, protocols and systems satisfying cryptographic encryption-decryption operations are called cryptosystems. There are two types of cryptosystems: private-key cryptosystems and public-key cryptosystems.

In private-key cryptosystems, it is computationally easy to obtain decryption key from encryption key. They are also called symmetric type of cryptosystems. It is divided into two main categories: block ciphers and stream ciphers. Former encrypt the message in fixed length strings (blocks) at a time (eg. DES, IDEA, AES, etc). Latter, on the other hand, operate on a single bit of the message at a time (eg. RC4, A5, etc.).

Secondly, in public-key cryptosystems, it is computationally difficult to recover

decryption key from encryption key or vice-versa. It is also called the asymmetric type of cryptosystems. Public-key cryptosystems are mainly designed by some trap door one-way functions that are infeasible to be inverted in a time for today's technology; but easy to be inverted with the help of some extra information. There are many mathematical problems that serve us trap door one-way functions. First, integer factorization problem depends on computationally difficulty of finding a non-trivial factor of a given integer(eg. RSA is a designed cryptosystem depending on this problem.). Second, subset sum problem depends on filling a given empty bag with some pre-defined packages (eg. Knapsack is a designed cryptosystem depending on this problem.). Third, discrete log problem depends on computationally infeasibility of finding the exponent with respect to the some pre-defined base of a given element in a group G (eg. El-Gamal Encryption, Diffie-Helmann(DH) Key Exchange and Digital Signature Algorithm(DSA) are examples depending on this problem when $G = F_q^*$). In particular, one can study discrete logarithm problem on the group of points on an elliptic curve over a finite field, for which finding the scalar multiplied with a pre-defined element of the curve with a resultant curve point that is also given is computationally difficult. For instance, given a pair P, Q on the elliptic curve and if it is known that Q is a scalar multiple of P, finding that scalar is one of the most computationally difficult problem in mathematics.

Table 1.1 makes a comparison of some commonly used cryptosystems in terms of their security strength according to today's knowledge. For instance, systems using 80 bits of AES key, 1024 bits RSA key and 160 bits elliptic curve DSA key have the same security strength. The reader may consult to [Kob94], [MOV96], [Sch96], [Sti02] for more about cryptography.

How can one use elliptic curve discrete log problem and construct a cryptosystem? The answer is the following. Firstly, two parties A and B, not necessarily to be persons, choose a secure elliptic curve that is publicly known. Next, for a given message element M, A converts it to an element on the elliptic curve: P. A calculates G=rK for a pseudo-random integer r and a pre-defined point K on the elliptic curve. Additionally, H = rE is calculated by A where E is the public

Table 1.1: Security Strength Comparison

AES Key Size (bits)	RSA and DH Key Size (bits)	Elliptic Curve DSA Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

encryption key of the receiver, B(it is only known by B that E = dK). Next, A calculates T = P + H. Finally, (T, G) pair is sent to B. B recovers P as T - dG = P where d is the secret decryption key of B. The system is known as Elliptic Curve El-Gamal Encryption Scheme. The reader may consult to [HMV04], [Men93] for more about elliptic curve cryptosystems.

As it is seen from previous paragraph, the operations rK and dG are performed for each message element. That type of operations are called scalar multiplication or point multiplication. Scalar multiplication is the most commonly operated calculation in an elliptic curve cryptosystem. Hence, it should be defined and implemented correctly in an elliptic curve cryptosystem.

The contribution of this thesis is to discuss the theoretical issues of elliptic curves and to analyze the scalar multiplication methods. Chapter 2 gives a brief introduction to elliptic curves and chapter 3 concentrates on the scalar multiplication methods, their algorithms and their analysis.

Chapter 2

Introduction to Elliptic Curves

Not only mathematicians, but also cryptographers have been dealing with the theory of elliptic curves since [Kob87] and [Mil85], independently, presented the use of elliptic curves in cryptosystems in mids of 1980s. History of the theory of elliptic curves reaches to approximately two centuries before. Even Carl Frederic Gauss had studied on counting the number of elements of elliptic curves on finite fields. One comes across with elliptic curves when (s)he deals with congruent number problem, square pyramid problem, Fermat's last Theorem, etc. For more detailed information in elliptic curves see [Was03], [BSS99], [Sil86], [Enge99].

2.1 Preliminaries

For cryptographic purposes, we will study elliptic curves on finite fields in this thesis. So, in order to define an elliptic curve on a finite field, we need to give some preliminary definitions and facts.

Definition 2.1. *A set G is a **group** with respect to a well-defined binary operation $+$ if*

- *G is associative, i.e. for all x, y, z in G, $x + (y + z) = (x + y) + z$ holds,*
- *G contains identity element 0, i.e. 0 satisfies $a + 0 = a$ for all a in G,*
- *Every element in G has an inverse in G, i.e. for each $x \in G$ there exists $x' \in G$ satisfying $x + x' = 0$.*

A group is called **commutative group** if it has the property that $x + y = y + x$, for all $x, y \in G$.

$\mathbb{C}, \mathbb{R}, \mathbb{Q}, \mathbb{Z}$ are examples of group with respect to usual addition.

Definition 2.2. *A set K is a **field** with respect to well-defined binary operations $+$ and $\cdot$ where $+$ is additive operation and $\cdot$ is multiplicative operation on G if*

- *F forms a commutative group with respect to $+$,*
- *$K^* = K \backslash \{0\}$ forms a commutative group with respect to $\cdot$,*
- *Distributivity law holds in K, i.e. $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ for all x, y, z in K.*

$\mathbb{C}, \mathbb{R}, \mathbb{Q}$ are examples of field with respect to usual addition and usual multiplication.

Any field having only finitely many elements is called **finite field or *Galois Field***. Some of the basic facts about finite fields are given below. For more on the theory of finite fields, one may consult to [LiNi93].

Lemma 2.3. *A finite field K has exactly p^n elements where p is a prime number and $n \geq 1$ integer. Then, p is known as the characteristic of K. Moreover, this K is usually represented as $GF(p^n)$.*

Lemma 2.4. *For any finite field K, K^* is cyclic.*

Now, it is time to define elliptic curve and concentrate on its properties.

Definition 2.5. *An **elliptic curve** E over a field K is defined by a non-singular projective plane curve over the algebraic closure $\overline{K}$ of K by the affine equation*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1.1}$$

where $a_1,\ a_2,\ a_3,\ a_4,\ a_6$ are elements of K. Non-singularity of a curve is known to be no multiple tangent lines at any point of the curve exist.

Let E' be a non-singular plane affine curve defined over $\overline{K}$ by

$$E' : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_1,\ a_2,\ a_3,\ a_4,\ a_6$ are elements of K. Then $E = E' \cup \{\infty\}$ where $\infty = [0, 1, 0]$ is a point in projective space and it is called as point at infinity.

Moreover, if L is any extension of K then

$$E(L) = \{(x, y) \in L \times\ L : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\infty\}$$

is defined as the set of L-**rational** points on E. In fact, E(L) where L is a finite field is the group which is used in cryptosystems.

Note that $E = E(\overline{K})$ and $E(L) = E'(L) \cup \{\infty\}$.

Equation 2.1.1 is named as **generalized Weierstrass equation** and it has the following descriptive quantities

$$\begin{aligned} d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \\ c_4 &= d_2^2 - 24d_4 \\ \Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ j(E) &= c_4^3/\Delta \end{aligned}$$

The quantity Δ is called the **discriminant** of E, while $j(E)$ is called the **j-invariant** of E.

Example 2.6. $y^2 + xy + 3y = x^3 + \frac{1}{3}x^2 + \frac{1}{2}x + 1/\mathbb{R}$,
$y^2 = x^3 + 5x + 25/F_{29}$,
$y^2 = x^3 - 25x/\mathbb{R}$
are examples of elliptic curve.

Definition 2.7. *[Enge99] Let E_1 and E_2 be two elliptic curves over a field K. Then E_1 and E_2 are isomorphic over K if there exist u, r, s, t in K, $u \neq 0$ with change of variables*

$$(x, y) \longmapsto (u^2x + r, u^3y + u^2sx + t)$$

changing equation E_1 to equation E_2.

Equivalently, matrix version is given as

$$\begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} u^2 & 0 \\ u^2s & u^3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} r \\ t \end{pmatrix}$$

This transformation is called **admissible change of variables**.

The following theorems explains the relation between Δ and non-singularity of elliptic curve, and between j-invariant and isomorphism of elliptic curves.

Theorem 2.8. *[Sil86] The elliptic curve E is non-singular if and only if $\Delta \neq 0$.*

Theorem 2.9. *[Sil86] If two elliptic curves E_1/K and E_2/K are isomorphic over K, then $j(E_1) = j(E_2)$. The converse is also true if K is an algebraically closed field.*

Let K be a finite field with $char(K) = p$. Then, an elliptic curve E over K consisting a point of order p is called **supersingular curve**, otherwise it is called **non-supersingular curve**. In particular, for $p = 2$ or 3, E is supersingular if and only if $j(E) = 0$. Supersingular elliptic curves are not used in cryptosystems due to the fact that they are highly vulnerable to MOV attack [MOV93].

Theorem 2.10. *[Was03] Let $E_1 : y_1^2 + a_1x_1y_1 + a_3y_1 = x_1^3 + a_2x_1^2 + a_4x_1 + a_6$ and $E_2 : y_2^2 + A_1x_2y_2 + A_3y_2 = x_2^3 + A_2x_2^2 + A_4x_2 + A_6$ be two elliptic curves over K with j-invariant j_1 and j_2. If $j_1 = j_2$, then there exists $\mu \neq 0$ in $\overline{K}$ (algebraic closure of K) such that $A_i = \mu^i a_i$ for $i = 1, 2, 3, 4, 6$. Moreover, E_1 is isomorphic to E_2 by the admissible change of variables*

$$x_2 = \mu^2 x_1, y_2 = \mu^3 y_1.$$

Proof : See section 2.6 of [Was03].

However, if we work on a field K that is not algebraically closed, then it is possible to find two elliptic curves with the same j-invariant that is not isomorphic to each other over K.

Example 2.11. *$E_1 : y^2 = x^3 - 25x$ and $E_2 : y^2 = x^3 - 4x$ over $\mathbb{Q}$ have the same j-invariant $= 1728$. It can be shown that $E_1(\mathbb{Q}) = < (-4, 6) >$ which has infinitely many elements, but $E_2(\mathbb{Q}) = \{(2, 0), (-2, 0), (0, 0), \infty\}$; hence one cannot find an admissible change of variables over $\mathbb{Q}$ transforming E_1 to E_2. However, if one works over $\mathbb{Q}(\sqrt{10})$ instead of $\mathbb{Q}$. Then,*

$$(x, y) \longmapsto (\mu^2 x, \mu^3 y)$$

where $\mu = \sqrt{10}/2$ transforms E_1 to E_2, namely $E_1(\mathbb{Q}(\sqrt{10})) \cong E_2(\mathbb{Q}(\sqrt{10}))$.

Finally, two elliptic curves E_1 and E_2 defined over a field K with the same j-invariant are called **twists** of each other because of the fact that there exists F an extension field of K such that $E_1(F) \cong E_2(F)$

2.2 Simplified Weierstrass Equations

It is possible to transform the generalized Weierstrass equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \ /K$$

to simple forms by using admissible change of variables with appropriate u,r,s,t candidates in K so that resultant equation of E is as simple as possible. We have three different cases as they are stated in [Sil86]:

1. *$char(K) \neq 2$ or 3*: The admissible change of variables[1]

$$(x, y) \longmapsto ((x - 3a_1^2 - 12a_2)/36, (y - 3a_1x)/216 - (a_1^3 + 4a_1a_2 - 12a_3)/24)$$

[1]The choice for the admissible change of variables is given in details in section 2.1 of [Was03]; in deed, it is not hard to imagine

transforms E to the curve with equation

$$y^2 = x^3 + ax + b$$

where $a, b \in K$. The discriminant of the simplified equation is $-16(4a^3 + 27b^2)$ and $j(E) = 1728\dfrac{4a^3}{4a^3 + 27b^2}$

2. $char(K) = 2$:

   - $a_1 \neq 0$: The admissible change of variables

     $$(x, y) \longmapsto (a_1^2 x + a_3/a_1, a_1^3 y + (a_1^2 a_4 + a_3^2)/a_1^3)$$

     transforms E to the curve with equation

     $$y^2 + xy = x^3 + ax^2 + b$$

     where $a \in K$ and b $\in K^*$ see section 2.7 of. $\Delta = b$, $j = 1/b$. So, this type of curves are non-supersingular.

   - $a_1 = 0$: The admissible change of variables

     $$(x, y) \longmapsto (x + a_2, y)$$

     transforms E to the curve with equation

     $$y^2 + cy = x^3 + ax + b$$

     where $a, b \in K$ and $c \in K^*$. $\Delta = c^4$, $j = 0$. So, this type of curves are supersingular.

3. $char(K) = 3$:

   - $a_1^2 \neq -a_2$: The admissible change of variables

     $$(x, y) \longmapsto (x + \frac{a_4 - a_1 a_3}{a_1^2 + a_2}, y + a_1 x + a_1 \frac{a_4 - a_1 a_3}{a_1^2 + a_2} + a_3)$$

transforms E to the curve with equation

$$y^2 = x^3 + ax^2 + b$$

where $a, b \in K^*$. $\Delta = -a^3 b$ and $j = -a^3/b$. So, this type of curves are non-supersingular.

- $a_1^2 = -a_2$: The admissible change of variables

$$(x, y) \longmapsto (x, y + a_1 x + a_3)$$

transforms E to the curve with equation

$$y^2 = x^3 + ax + b$$

where $a \in K^*$ and $b \in K$. $\Delta = -a^3$ and $j = 0$. So, this type of curves are supersingular.

2.3 Group Law

In this section, we will explain the group structure of the elliptic curve group E(K) which is needed for the cryptosystems.

Theorem 2.12. (Bezout) *Let C_1 and C_2 be two curves over an algebraically closed field defined by the equations F(x,y)=0 and G(x,y)=0 respectively. Then, the number of points counted with multiplicity in the set $C_1 \cap C_2 := \#(C_1 \cap C_2) = degF degG$. In particular, let $C_1 = E$, an elliptic curve, and $C_2 = l$, a line, then $\#(C_1 \cap C_2) = 3$.*

Therefore, we can define the group operation of elliptic curves with the help of *Bezout's Theorem* as: Let E be an elliptic curve over K

- Take two point on the curve: $P, Q \in E(K) \backslash \{\infty\}$
- First, draw a line through P and Q: l

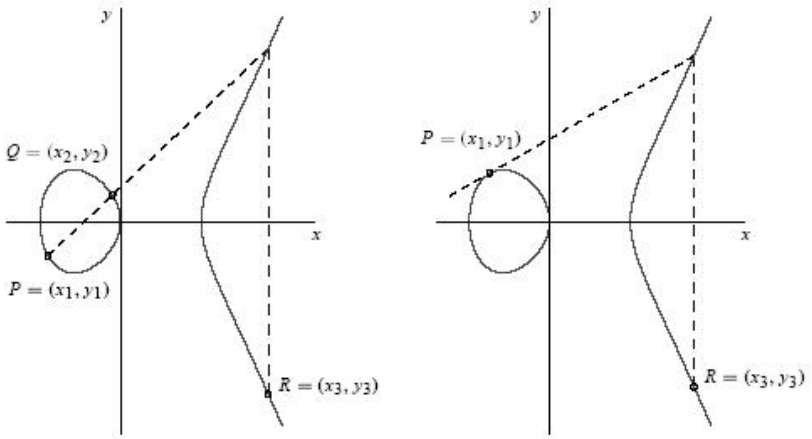

Figure 2.1: Point addition and point doubling on an elliptic curve.

- Then, l intersects the curve at a third point: $T \in E(K)$
- Then, reflect T with respect to x axis.
- Resultant point, $R \in E(K)$, is the addition of P and Q.

Remark 2.13. *One needs to take care of followings while adding points on an elliptic curve:*

- *If* $P, Q \in E(K)$ *then* $P + Q \in E(K)$.
- *If* $P = Q$, *then the line joining* P *to* Q *is the tangent line of* E *at* P.
- *If* l *is a vertical line, then* $T = \infty$.
- *If* $T = \infty$, *then* $R = \infty$, *too.*
- *Point at infinity:*$= \infty$ *is the identity element of the group E(K).*

It is easy to obtain algebraic formulas using easily remarks above as we will do now:

Definition 2.14. *Let E be an elliptic curve over a field K. Let P_1 and P_2 be points on $E(K)$. Define $P_3 = P_1 + P_2$.*
If $P_1 = \infty$ or $P_2 = \infty$, then $P_3 = P_2$ or $P_3 = P_1$, respectively.
Else let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$

1. *$char(K) \neq 2$ or 3.*

   - *$x_1 = x_2$ but $y_1 \neq y_2$: $P_3 = \infty$.*
   - *$x_1 = x_2$ and $y_1 = y_2$ and $y_1 = 0$: $P_3 = \infty$.*
   - *$x_1 = x_2$ and $y_1 = y_2$ but $y_1 \neq 0$ and let $P_3 = (x_3, y_3)$:*

     $$x_3 = m^2 - 2x_1 \text{ and } y_3 = m(x_1 - x_3) - y_1$$

     where $m = \dfrac{3x_1^2 + a}{2y_1}$.
   - *$x_1 \neq x_2$ and let $P_3 = (x_3, y_3)$:*

     $$x_3 = m^2 - x_1 - x_2 \text{ and } y_3 = m(x_1 - x_3) - y_1$$

     where $m = \dfrac{y_2 - y_1}{x_2 - x_1}$

2. *$char(K) = 2$:*

   - *non-supersingular type:*
     - *$x_1 = x_2$ and $y_2 = x_1 + y_1$: $P_3 = \infty$.*
     - *$x_1 = x_2$ and $y_1 = y_2$ and let $P_3 = (x_3, y_3)$:*

       $$x_3 = m^2 + m + a = x_1^2 + b/x_1^2 \text{ and } y_3 = x_1^2 + mx_3 + x_3$$

       where $m = x_1 + y_1/x_1$
     - *$x_1 \neq x_2$ or $y_2 \neq x_1 + y_1$ and let $P_3 = (x_3, y_3)$:*

       $$x_3 = m^2 + m + x_1 + x_2 + a \text{ and } y_3 = m(x_1 + x_3) + x_3 + y_1$$

       where $m = \dfrac{y_2 + y_1}{x_2 + x_1}$

- *supersingular type:*
  - $x_1 = x_2$ *or* $y_1 = y_2 + c$*:* $P_3 = \infty$.
  - $x_1 = x_2$ *and* $y_1 = y_2$ *and let* $P_3 = (x_3, y_3)$*:*

$$x_3 = m^2 \text{ and } y_3 = m(x_1 + x_3) + y_1 + c$$

    where $m = (x_1^2 + a)/c$
  - $x_1 \neq x_2$ *or* $y_1 \neq y_2 + c$ *and let* $P_3 = (x_3, y_3)$*:*

$$x_3 = m^2 + x_1 + x_2 \text{ and } y_3 = m(x_1 + x_3) + y_1 + c$$

    where $m = \dfrac{y_2 + y_1}{x_2 + x_1}$

3. $char(K) = 3$*:*

- *non-supersingular type:*
  - $x_1 = x_2$ *but* $y_1 \neq y_2$*:* $P_3 = \infty$.
  - $x_1 = x_2$ *and* $y_1 = y_2$ *and* $y_1 = 0$*:* $P_3 = \infty$.
  - $x_1 = x_2$ *and* $y_1 = y_2$ *but* $y_1 \neq 0$ *and let* $P_3 = (x_3, y_3)$*:*

$$x_3 = m^2 - a - 2x_1 \text{ and } y_3 = m(x_1 - x_3) - y_1$$

    where $m = \dfrac{3x_1^2 + 2ax_1}{2y_1}$.
  - $x_1 \neq x_2$ *and let* $P_3 = (x_3, y_3)$*:*

$$x_3 = m^2 - a - x_1 - x_2 \text{ and } y_3 = m(x_1 - x_3) - y_1$$

    where $m = \dfrac{y_2 - y_1}{x_2 - x_1}$
- *supersingular type:*
  - $x_1 = x_2$ *but* $y_1 \neq y_2$*:* $P_3 = \infty$.
  - $x_1 = x_2$ *and* $y_1 = y_2$ *and* $y_1 = 0$*:* $P_3 = \infty$.

– $x_1 = x_2$ *and* $y_1 = y_2$ *but* $y_1 \neq 0$ *and let* $P_3 = (x_3, y_3)$*:*

$$x_3 = m^2 - 2x_1 \text{ and } y_3 = m(x_1 - x_3) - y_1$$

where $m = \dfrac{3x_1^2 + a}{2y_1}$.

– $x_1 \neq x_2$ *and let* $P_3 = (x_3, y_3)$*:*

$$x_3 = m^2 - x_1 - x_2 \text{ and } y_3 = m(x_1 - x_3) - y_1$$

where $m = \dfrac{y_2 - y_1}{x_2 - x_1}$

Finally, we also construct the addition formulas for generalized Weierstrass equation where $R = P + Q$, and $P = (x_1, y_1)$, $Q = (x_2, y_2)$ and $R = (x_3, y_3)$:

$$x_3 = \begin{cases} \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)^2 + a_1\left(\dfrac{y_2 - y_1}{x_2 - x_1}\right) - a_2 - x_1 - x_2 & (P \neq Q) \\ \left(\dfrac{3x_1^2 + 2a_2x + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}\right)^2 + a_1\left(\dfrac{3x_1^2 + 2a_2x + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}\right) - a_2 - 2x_1 & (P = Q) \end{cases}$$

$$y_3 = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1 - (a_1x_3 + a_3) & (P \neq Q) \\ \dfrac{3x_1^2 + 2a_2x + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}(x_1 - x_3) - y_1 - (a_1x_3 + a_3) & (P = Q) \end{cases}$$

Theorem 2.15. *The addition of points on an elliptic curve satisfies the following properties:*

i. Commutativity

ii. Existence of identity element

iii. Existence of unique inverse element for each element

iv. Associativity

Proof: Since line through P and Q is the same as the line through Q and P, the addition is commutative. Next, point at infinity is the unique identity element of the group by definition. By definition inverse of any point $P = (x, y)$ can be obtained as follows:

- In the case of $char(K) \neq 2$ or 3 : $-(x, y) = (x, -y)$;
- $Char(K) = 2$

  Non-supersingular case: $-(x, y) = (x, x + y)$;

  Supersingular case: $-(x, y) = (x, y + c)$;
- $Char(K) = 3$ for both non-supersingular and supersingular case:

$$-(x, y) = (x, -y).$$

Note that if $P = (x, y)$ satisfies the generalized Weierstrass equation, then

$$-P = (x, -a_1x - a_3 - y).$$

Finally, for the proof of associativity see section 2.4 of [Was03].

Table 2.1: Simplified Forms of Elliptic Curves

Simplified form	Δ	j	$-(x_0, y_0)$
$char(K) \neq 2, 3$ $y^2 = x^3 + ax + b$	$-16(4a^3 + 27b^2)$	$1728\frac{4a^3}{4a^3 + 27b^2}$	$(x_0, -y_0)$
$char(K) = 3$ and $j \neq 0$ $y^2 = x^3 + ax^2 + b$	$-a^3b$	$-\frac{a^3}{b}$	$(x_0, -y_0)$
$char(K) = 3$ and $j = 0$ $y^2 = x^3 + ax + b$	$-a^3$	0	$(x_0, -y_0)$
$char(K) = 2$ and $j \neq 0$ $y^2 + xy = x^3 + ax^2 + b$	b	$\frac{1}{b}$	$(x_0, x_0 + y_0)$
$char(K) = 2$ and $j = 0$ $y^2 + cy = x^3 + ax + b$	c^4	0	$(x_0, y_0 + c)$

2.4 Endomorphisms

Endomorphisms have crucial role in the theory of elliptic curves. **An endomorphism** α of an elliptic curve E over a field K is defined as homomorphism on $E(\overline{K})$ given by rational functions. That is,

$$\alpha : E(\overline{K}) \mapsto E(\overline{K})$$
$$P \mapsto (g_1(P), g_2(P))$$

where g_1 and g_2 are rational functions on E (quotient of polynomials) and

$$\alpha(P_1 + P_2) = \alpha(P_1) + \alpha(P_2)$$

for all P_1 and $P_2 \in E(\overline{K})$. The set of all endomorphisms of E over K forms a ring under addition and composition [Enge99], called *the endomorphism ring of E over K*. The characteristic polynomial of an endomorphism α is defined to be the least degree monic polynomial $f(x) \in \mathbb{Z}[x]$ satisfying $f(\alpha)(P) = \infty$ for all $P \in E$, if it exists[2].

Example 2.16. *[HMV04]*

(i) Let E be given by $y^2 = x^3 + ax + b$ over K.

Define $\alpha : E(\overline{K}) \mapsto E(\overline{K})$ $\alpha(P) = 2P$

It can be checked that α is a homomorphism. Moreover, we can define α as

$$\alpha(x, y) = (g_1(x, y), g_2(x, y))$$

where

$$g_1(x, y) = (3x^2 + \frac{a}{2y})^2 - 2x \text{ and}$$
$$g_2(x, y) = (3x^2 + \frac{a}{2y})(3x - ((3x^2 + \frac{a}{2y})^2) - y).$$

Since g_1 and g_2 are rational functions, α is an endomorphism of E. The char-

[2]The term *characteristic polynomial* is not equivalent to linear algebraic one

acteristic polynomial of α is $f(x) = x - 2$. Generalization of this example is the following:

(ii) Let E be an elliptic curve defined over $K = F_q$. For each n, the multiplication by n defined as $\alpha : E(\overline{K}) \mapsto E(\overline{K})$, $\alpha(P) = nP$ is an endomorphism of E. The characteristic polynomial of α is $f(x) = x - n$.

(iii) Let E be an elliptic curve defined over $K = F_q$. The map ϕ defined by

$$\phi : E(\overline{K}) \mapsto E(\overline{K})$$

$$(x, y) \mapsto (x^q, y^q),\ \phi(\infty) = \infty$$

is an endomorphism of E. The above map is generally shown by ϕ_q and called Frobenius endomorphism. *The characteristic polynomial of ϕ_q is $f(x) = x^2 - (q + 1 - \#E(F_q))x + q$ [Was03].*

(iv) Let $p \equiv 1 (mod\ 4)$ be a prime, and $E : y^2 = x^3 + ax$ be defined over F_p. Let i be a 4^{th} root of unity in F_p, and let $\alpha : E(\overline{K}) \mapsto E(\overline{K})$

$$\alpha(x, y) = (-x, iy),\ \alpha(\infty) = \infty.$$

It is easy to see that α is an endomorphism of E having the characteristic polynomial of $f(x) = x^2 + 1$.

(v) Let $p \equiv 1 (mod\ 3)$ be a prime, and $E : y^2 = x^3 + b$ be defined over F_p. Let β be a 3^{rd} root of unity in F_p. Define $\alpha : E(\overline{K}) \mapsto E(\overline{K})$

$$\alpha(x, y) = (\beta x, y),\ \alpha(\infty) = \infty.$$

is an endomorphism of E. The characteristic polynomial of α is $f(x) = x^2 + x + 1$.

Remark 2.17. *In examples (iv) and (v) $\alpha(P)$ can be computed easily, that is using only one field multiplication.*

As we said before, endomorphisms have crucial role in the theory of elliptic curves; especially, *Frobenius endomorphism* is used in a wide range. For instance,

it is used in proving the Hasse's Theorem, which constructs a nice range for the order of an elliptic curve over a finite field:

Theorem 2.18. *(Hasse) Let E be an elliptic curve over F_q. Then the order of $E(F_q)$ satisfies*

$$|q + 1 - \#E(F_q)| \leq 2\sqrt{q}.$$

In order to decide the security strength of an elliptic curve cryptosystem, number of rational points is a crucial feature of the system. It is desirable to be large and devided by large primes. Here, Hasse's Theorem does not give us any information about factors of the $\#E(F_q)$; but says that $\#E(F_q)$ is approxiamately q.

Chapter 3

Scalar Multiplication on Elliptic Curves

Scalar multiplication (or point multiplication) is the operation of calculating an integer multiple of an element in additive group of elliptic curve. In other words, it is calculation of kP for any integer k and a point P on the curve. Scalar multiplication corresponds to group element exponentiation in a multiplicative group, i.e. x^k, for some x in the multiplicative group. Therefore, one can adapt easily historical exponentiation methods to scalar multiplication, replacing multiplication by addition and squaring by doubling. Mathematicians have been dealing with exponentiation methods for more than two thousands years, efficient methods have been grown up in that period.

In this chapter, we will give many scalar multiplication methods starting from generic ones up to curve specific ones. There are two main situations that occur in practice: base point P is not known a priori and known a priori, there is, moreover, a sub case in which the scalar is used several times. Additionally, different coding methods of k gives different scalar multiplication methods.

The expected running time and the worst case running time is important to analyze a scalar multiplication method. Thus, one of them is always given in details when algorithm is coded. Algorithms are constructed for the points P in $E(F_q)$ for some $q = p^m$ where p is prime and m in $\mathbb{Z}^+$ since cryptosystems are designed on $E(F_q)$.

3.1 Unknown Point

In this section both k and P are unknown until the run-time, i.e. they are seeded in to the program at the run time. Since k and P may vary, methods given in this section may be realized as generic methods.

3.1.1 Binary Method

Binary method is the first known exponentiation method, so a scalar multiplication method. Binary representation of the scalar enables us to interpret the multiplication as cumulative addition of non-zero components. Namely, if k has binary representation $(k_{l-1}, k_{l-2}, \dots, k_0)_2$ where $k_i \in \{0, 1\}$, then $k = \sum_{i=0}^{l-1} k_i 2^i$.

$$
\begin{aligned}
kP &= \sum_{i=0}^{l-1} k_i 2^i P && (3.1.1)\\
&= k_0 P + k_1 2^1 P + k_2 2^2 P + \dots\ k_{l-1} 2^{l-1} P && (3.1.2)\\
&= k_0 P + 2(k_1 P + 2(k_2 P + ... + (2(k_{l-2} P + 2(k_{l-1} P))...) && (3.1.3)
\end{aligned}
$$

Equation 3.1.2 can be interpreted as starting from k_0 and summing the terms $k_i 2^i P$ up cumulatively for each non-zero k_i up to k_{l-1} to end up with the result kP. $2^i P$ can be calculated by $2 * 2^{i-1} P$ if $2^{i-1} P$ is known. Thus, in order to speed up more, $2^i P$ is needed to be calculated for each $i \in \{0, \dots, l-1\}$ by doubling the previous one, and added to the cumulative summand if k_i is 1. Because of which, this case of the method is known as *double-and-add*. Also, it is called *right-to-left binary method* since it starts from k_0 ends with k_{l-1}. The pseudo-code of right-to-left binary method is Algorithm 3.1.

Next, equation 3.1.3 enables us to interpret the multiplication as starting from k_{l-1} down to k_0 and adding P if k_i is non-zero and continuously doubling whatever k_i is. In contrast to previous case, it is not needed to keep at hand a doubled version of P. In other words, memory is not allocated for a doubled version of P. Because of similar reasons, this case of the method is known as *add-and-double* or *left-to-right binary method*. And, its pseudo-code is given in

the Algorithm 3.2.

Algorithm 3.1. *[Knu81]*
Right-to-left binary method for scalar multiplication
input: $k = (k_{l-1}, k_{l-2}, \dots, k_0)_2$, $P \in E(F_q)$.
output: kP
1. $Q = \infty$.
2. for i from 0 to l-1 do
2.1 if $k_i = 1$ *then* $Q = Q + P$.
2.2 $P = 2P$.
3. output(Q)

Algorithm 3.2. *[Coh93]*
Left-to-right binary method for scalar multiplication
input: $k = (k_{l-1}, k_{l-2}, \dots, k_0)_2, P \in E(F_q)$.
output: kP
1. $Q = \infty$.
2. for i from l-1 down to 0 do
2.1 $Q = 2Q$
2.1 if $k_i = 1$ *then* $Q = Q + P$.
3. output(Q)

The running time of a algorithm is determined as how many operations are performed throughout its execution. In order to do that, it is needed to analyze each line of the algorithm in detail. For instance, line 1 of Algorithm 3.1 is equivalence and it is very fast and it is not included into the running time analysis. Similarly, line 2 is determination of the loop and it is very fast and not included into the running time. Next, if k_i is 1, then a point addition is performed in the line 2.1. Point addition needs some field operations; because of which, it is heavy operation and included into the running time. Since the expected number of ones in the binary representation of k is half of its length $(l/2)$, line 2.1 is expected to run $l/2$ times. Finally, a doubling is performed in the line 2.2 for each value of i, that is, l times. Because of the same reasons of point addition,

point doubling is included into the running time. Therefore, the expected running time of Algorithm 3.1 is $l/2Additions + lDoubling$ denoted as

$$\frac{l}{2}A + lD.$$

Algorithm 3.2 has the same operations as Algorithm 3.1 with reverse order, so they have the same running time. The analysis of Algorithm 3.1 was explained in details for clarity, but for the next times, it will be done directly. Also, the running time analysis will be given in terms of point addition and point doubling to compare them.

Example 3.3. $k = 26 = (11010)$. *26P can be calculated by Algorithm 3.1 and Algorithm 3.2. The algorithm steps are given in Table 3.1 and Table 3.2.*

Table 3.1: Right-to-Left Binary Method

i	0	1	2	3	4
k_i	0	1	0	1	1
P	∞	2P	4P	8P	16P
Q	∞	∞+2P=2P	2P	2P+8P=10P	10P+16P=26P

Table 3.2: Left-to-Right Binary Method

i	4	3	2	1	0
k_i	1	1	0	1	0
Q	∞+P=P	2(P)+P=3P	2(3P)=6P	2(6P)+P=13P	2(13P)=26P

3.1.2 Non-adjacent Form(NAF)

By previous chapter, we know that inverse of $P = (x, y) \in E(F_q)$ is $-P = (x, x + y)$ in binary fields and $-P = (x, -y)$ in the fields of characteristic ≥ 3. Thus, taking inverse of an element on elliptic curve is very fast in terms of

computational time. This brings up the question of representing k in the form

$$k = \sum_{i=0}^{l-1} k_i 2^i \ where \ k_i \in \{-1, 0, 1\}$$

to get fast computation for kP. Additional to binary method, when minus one is come across, subtraction of P is performed during the scalar multiplication kP.

A representation whose set consists also negative values is called **signed digit representation(SDR)**. If the representation set is {-1,0,1}, then it is the most trivial type signed digit representation and known as **signed binary representation**.

In the binary method, we have noticed that running time of algorithm increases proportional to number of ones in its representation. Hence, the aim is to form a representation of an integer k whose weight(number of nonzero elements) and length is as small as possible. There is a representation which was studied before satisfies this aim:

Definition 3.4. *A non-adjacent form(NAF) of a positive integer k is an expression* $k = \sum_{i=0}^{l-1} k_i 2^i$ *where* $k_i \in \{-1, 0, 1\}$, $k_{l-1} \neq 0$ *and no two consecutive digits are nonzero. According to definition, the length of the NAF is l.*

For any $k \geq 1$, NAF exists and has the following properties:

Theorem 3.5. *Let k be a positive integer*

i. k has a unique NAF denoted NAF(k) [Rei60].

ii. NAF(k) has the fewest nonzero digits of any signed binary representation of k [Rei60].

iii. The expected value of number of the ones of NAF(k) over the length of NAF(k) is 1/3 [MoOl90].

iv. The length of NAF(k) is at most one more than the length of the binary representation of k [MoOl90].

Algorithm 3.6, 3.7, 3.8 and 3.9 give the recoding[1] of a given integer k into NAF(k), then Algorithm 3.11, which takes NAF(k), computes the scalar multiplication kP using NAF representation of k, this is called NAF method. We will, first, give the algorithms and then explain the steps of them.

Algorithm 3.6. *[MoOl90]*
Right-to-left NAF recoding
input: $k = (k_{l-1}, \ldots, k_0)_2 \in \mathbb{Z}^+$
output: NAF(k)
1. $i = 0$
2. while $k > 0$ *do*
2.1 if k is even then $k'_i = 0$
2.2 else $\{k'_i = 2 - (k \bmod 4),\ k = k - k'_i\}$
2.3 $k = k/2$, $i = i + 1$
3. output $(k'_l, k'_{l-1}, \ldots\ k'_0)$

Algorithm 3.6 recodes NAF(k) by repeatedly dividing k by 2 and assigning k'_i to be 0 if k is even or 1 if $(k-1)/2$ is even or -1 if $(k+1)/2$ is even.

Algorithm 3.7. *[Rei60]*
Right-to-left NAF recoding
input: $k = (k_{l-1}, k_{l-2}, \ldots\ , k_0)_2$
output: NAF(k)
1. $c_0 = 0$
2. for $i = 0$ *to* l *do*
2.1 $c_{i+1} = \lfloor (c_i + k_i + k_{i+1})/2 \rfloor$
2.2 $k'_i = c_i + k_i - 2c_{i+1}$
3. output $(k'_l, \ldots, k'_0)$

Algorithm 3.7 uses the identity $3k - k = 2k$ and performs the subtraction bit by bit with exception $0 - 1 = \overline{1}$. Let $k = (k_{l-1}, \ldots, k_0)_2$ be the input scalar, $(k'_l, \ldots, k'_0)$ be the output sequence, c_i be carry sequence at the i^{th} iteration and

[1]Recoding is an operation of transferring a representation to another one

3k be $(s_l, \ldots, s_0, r_0)$. Then,

$$\begin{aligned} s_i &= k_i + k_{i+1} + c_i \ mod\ 2 \\ &= k_i + k_{i+1} + c_i - 2\lfloor (k_i + k_{i+1} + c_i)/2 \rfloor \\ &= k_i + k_{i+1} + c_i - 2c_{i+1} \end{aligned}$$

So, $k'_i = s_i - k_{i+1} = c_i + k_i - 2c_{i+1}$.

In order to show the output of Algorithm 3.7 is in non-adjacent form, it is enough to show $k'_i k'_{i+1} = 0 \,\forall\, i \leq l-1$ Suppose $k'_i \neq 0$. We know $k'_i = c_i + k_i - 2c_{i+1}$. So, $c_i + k_i$ must be 1, otherwise k'_i is not zero. Then by using this fact, $c_{i+1} = \lfloor (1 + k_{i+1})/2 \rfloor = k_{i+1}$. Hence, $k'_{i+1} = 2k_{i+1} - 2c_{i+2}\ mod\ 2 = 0$. Therefore, $k'_i k'_{i+1} = 0$, as desired.

Note that Algorithm 3.6 and 3.7 recodes k from right to left. However, as we have seen in binary method, left-to-right scalar multiplication is better than right to left one in terms of memory consumption. In addition, we have to wait until the end of Algorithm 3.6 or 3.7 in order to run scalar multiplication. Now we will discuss so called *on the fly recoding* method. NAF(k) recoding can be obtained left-to-right(starting from k_{l-1}) by using Algorithm 3.8 or 3.9, and simultaneously, we can compute kP by Algorithm 3.11.

Algorithm 3.8. *[JoYe00]*
Left-to-Right NAF recoding
input: $(k_{l-1}, \ldots, k_0)_2$
output: NAF(k)
1. $j = m, b = 0, k_l = 0$
2. for i form l-1 down to 0 do
2.1 if $(k_{i+1} = k_i)$ *then*
2.1.1 $k'_j = k_i - b$
2.1.2 while $(j > i + 1)$ *do*
2.1.2.1 $j = j - 1$, $k'_j = 1 - k_i - b$, $b = 1 - b$
2.1.3 $b = k_i$, $j = j - 1$
3. $k'_j = -b$

4. while $(j > 0)$ *do*
4.1 $j = j - 1$, $k'_j = 1 - b$, $b = 1 - b$
5. output$(k'_l, \ldots, k'_0)$

Algorithm 3.8 uses the trick of r+2r by LtoR and then similar to previous algorithm subtracts r from 3r bit by bit with the exception $0 - 1 = \overline{1}$.

Algorithm 3.9. *[JoYe00]*
Left-to-Right minimum weight signed digit recoding
input: $(k_{l-1}, \ldots, k_0)_2$
output: $(k'_l, \ldots, k'_0)_{SDR}$
1. $b_l = 0$, $k_l = 0$, $k_{-1} = 0$, $k_{-2} = 0$
2. for i from l down to 0 do
2.1. $b_{i-1} = \lfloor (b_i + k_{i-1} + k_{i-2})/2 \rfloor$
2.2. $k'_i = -2b_i + k_i + b_{i-1}$
3. output$(k'_l, \ldots, k'_0)_{SDR}$

Algorithm 3.9 has also minimum weight property; but, it is not in the non-adjacent form. In addition, a table look-up version of Algorithm 3.9 can be used to recode of a positive integer $k = (k_{l-1}, \ldots, k_0)$. The details of the algorithm and the table can be obtained from [JoYe00].

Table 3.3: Look-up Table of Algorithm 3.9

b_i	k_i	k_{i-1}	k_{i-2}	b_{i-1}	k'_i
0	0	0	x	0	0
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	x	0	1
1	0	1	x	1	$\overline{1}$
1	1	0	0	0	$\overline{1}$
1	1	0	1	1	0
1	1	1	x	1	0

Here is an example to illustrate the Algorithms 3.6, 3.7, 3.8 and 3.9.

Example 3.10. *In order to trace it easily, a small number 26 is chosen.*
$26 = (10\bar{1}010)_{NAF}$ *by Algorithms 3.6, 3.7 and 3.8*
$26 = (100\bar{1}\bar{1}0)_{SDR}$ *by Algorithm 3.9.*

Algorithm 3.11. *Left-to-right NAF multiplication*
input: NAF(k) $= (k_{l-1}, k_{l-2}, \dots, k_0)$ *and* $P \in E(F_q)$
output: $kP \in E(F_q)$
1. $Q = \infty$
2. For i from $l-1$ *down to 0 do*
2.1. $Q = 2Q$
2.2. If $k_i = 1$ *then* $Q = Q + P$
2.3. If $k_i = -1$ *then* $Q = Q - P$
3. Output(Q)

Note that, the line 2.1 of Algorithm 3.11 performs exactly l times and by Theorem 3.5 (iii) and (iv). It is expected that the lines 2.2 and 2.3 together performs approximately $l/3$ times. Therefore, expected running time of Algorithm 3.11 is

$$l/3A + lD.$$

3.1.3 Window Method

If the digits of representation of k are allowed to be the elements of a larger set instead of only $\{-1, 0, 1\}$, then running time of above algorithms are decreased. In this case, not only P is added or subtracted, but also some small scalar multiple of P is added or subtracted. So, those values have to be calculated at the beginning of the scalar multiplication algorithm and saved to the memory. The window method may be interpreted as processing some consecutive digits of the scalar at a time. There are unsigned and signed versions of window method. Unsigned width-w window representation of a positive integer k is $k = \sum_{i=0}^{l-1} k_i 2^i$ where k_i is either zero or odd integer smaller than 2^w and $k_{l-1} \neq 0$. Similarly, signed

width-w window representation of a positive integer k is $k = \sum_{i=0}^{l-1} k_i 2^i$ where $|k_i|$ is either zero or odd integer smaller than 2^{w-1}, $k_{l-1} > 0$, and particularly *width-w NAF* representation of a positive integer k is $k = \sum_{i=0}^{l-1} k_i 2^i$ where $|k_i|$ is either zero or odd integer smaller than 2^{w-1}, $k_{l-1} > 0$, and additionally at most one of any w consecutive digits is nonzero. Since width-w NAF decreases nonzero terms fairly, we will only deal with properties and running time of it.

Theorem 3.12. *[MuSt04a]*
Let k be positive integer

i. k has unique width-w NAF. It is denoted by $NAF_w(k)$.

ii. The length of $NAF_w(k)$ is at most one more than the length of binary representation of k.

iii. The expected value of number of nonzero digits of $NAF_w(k)$ over length of $NAF_w(k)$ is $1/(w+1)$.

Note that $NAF(k) = NAF_2(k)$.

$NAF_w(k)$ can be computed easily similar to $NAF(k)$. In order to ensure that w consecutive digits contains at most one nonzero digit, reduction modulo 2^w has to be done by choosing least residue as represented in the line 2.2 of Algorithm 3.13.

Algorithm 3.13. *[Sol00]*
Computing the NAF_w of a positive integer
input: window width w, positive integer k
output: $NAF_w(k)$
1. $i = 0$
2. while $k > 0$ do
2.1. if k is even $k_i = 0$
2.2. else $k_i = k$ mods 2^w, $k = k - k_i$
2.3. $k = k/2$, $i = i + 1$
3. output $(k_{l-1}, \ldots, k_1, k_0)$

The difference of $NAF_3(26)$ and $NAF_2(26)$ is clear in the following example.

Example 3.14. *$NAF_3(26) = 1000\bar{3}0$ is the output of Algorithm 3.13 and $NAF_2(26) = (10\bar{1}010)$ is the output of Algorithm 3.6.*

As in the binary method and NAF method, if one can do the recoding of $NAF_w(k)$ left-to-right, scalar multiplication operation using $NAF_w(k)$ can be performed on-the-fly. That is, scalar multiplication and recoding operations can be performed simultaneously.

Avanzi[Ava04], Muir et al.[MuSt04b] and Okeya et al.[OSST04] independently obtained similar results of left-to-right recoding of an integer k having the least Hamming weight. We will give Avanzi's method, and also explain others a little. [MuSt04b]'s left-to-right algorithm is optimal, is different from Avanzi's one and can output up to two different recodings of the same integer, one of which is equal to that of Avanzi's algorithm, whereas the other one differs on some of the least significant digits. Okeya et al. also have a left-to-right algorithm, do not prove equivalence to the w-NAF but only give asymptotic density estimates using Markov chains.

Algorithm 3.15. *[Ava04]*
Computing width-w left-to-right representation of an integer($LtoR_w$)
input: $k = (k_{l-1}, \ldots, k_0)_2$, *width* $w \in \mathbb{Z}^+$
1. for j from 0 up to $l + w - 1$ *do* $k'_j = 0$
2. $k_l = k_{-1} = 0$ *and* $i = l$
3. while $(i \geq 0)$ *do*
3.1 if $(k_i = k_{i-1})$ *then* $i = i - 1$
3.2 else
3.2.1 $w' = min\{w, i + 1\}$
3.2.2 $v = -k_i 2^{w'-1} + \sum_{j=0}^{w'-2} k_{i-(w'-1)+j} 2^j + k_{i-w'}$
3.2.3 $k'_{i-(w'-1)+s} = v/2^s$, *where* $2^s || v$
3.2.4 $i = i - w'$
4. output $(k'_l, \ldots, k_0)$

Avanzi states that Algorithm 3.15 outputs an expression which evaluates the integer represented by its input, and that Hamming weight of $NAF_w(k)$ is equivalent to Hamming weight of $LtoR_w(k)$ for all k and w. In particular, the $LtoR_w$ is a recoding of minimal weight among all the width w SDR's, besides the NAF_w

Example 3.16. *[Ava04] The NAF_4 and the $LtoR_4$ of 1971 coincide and are equal to $(1000000\bar{5}0003)$. On the other hand, the NAF_4 of 2004 is $(10000\bar{1}000500)$ whereas its $LtoR_4$ is $(10000000\bar{5}\bar{1}00)$. Here, the two recodings differ but have the same length. Let us consider now 2359, the NAF_4 is $(1000\bar{7}00030007)$ but the $LtoR_4$ is $(500\bar{3}00\bar{1}00\bar{1})$, which is shorter. Of course, in all examples shown, the NAF_w and the $LtoR_w$ have the same Hamming weight. The recodings of 2004 and 2359 are examples of the fact that the $LtoR_w$ does not necessarily satisfy the generalized non-adjacency property.*

Computation of scalar multiplication by $NAF_w(k)$ or by $LtoR_w(k)$ is a general version of usual $NAF(k)$ scalar multiplication. However, there exists a precomputation stage.

Algorithm 3.17. *[Sol00]*
Left-to-right $NAF_w(k)$ multiplication
input: $NAF_w(k), P \in E(F_q)$
output: kP
0. compute $2P$
1. for i=3 up to $2^{w-1}-1$ do
1.1 if i is odd then compute $P_i = P_{i-1} + 2P$
2. $Q = \infty$
3. for i = l-1 down to 0 do
3.1 $Q = 2Q$
3.2 if $k_i \neq 0$ then
3.2.1 if $k_i > 0$ then $Q = Q + P_{k_i}$
3.2.2 else $Q = Q - P_{-k_i}$
4. output(Q)

Running time of line 0 and line 1 of Algorithm 3.17 is $1D$ and $(2^{w-2}-1)A$ respectively; precomputation cost, therefore, is $1D+(2^{w-2}-1)A$. Next, running time of line 3.1 is lD and expected total running time of line 3.2.1 and line 3.2.2 is $(l/(w+1))A$. To sum up, the expected running time of Algorithm 3.17 is

$$(l+1)D+(2^{w-2}+\frac{l}{w+1}-1)A.$$

3.1.4 Sliding Window Method

This method operates left-to-right over the digits of k with a window width at most w, at which the value in the window is odd. In contrast to window method, it has no exact window width; but similar to window method, it ignores zero digits. This method can be applied to binary or NAF representation of k. It may be applied to $NAF_w(k)$ with window width at most w', but, the same algorithm of window NAF is obtained unless $w' > w$.

Algorithm of sliding window method applied to NAF_2 is given in the Algorithm 3.18. So, it has to be computed before, and seeded to the algorithm. The first stage is precomputation of P_i for some i. Observe that a block of digits in an arbitrary window can have a value at most either $101010\ldots10101$(w-digits) or $101010\ldots101001$(w-digits) for w is odd or even respectively. Therefore, the value of the most valuable block is either $(2^{w+1}-1)/3$ or $(2^{w+1}-5)/3$ which implies that the upper bound for the precomputation stage is

$$2\frac{2^w-(-1)^w}{3}-1.$$

Algorithm 3.18. *[BSS99]*
NAF Sliding window method for scalar multiplication
input: window width w, NAF(k), $P \in E(F_q)$
output: kP
0. $P_1 = P$
1. compute 2P
2. for i from 3 to $2(2^w-(-1)^w)/3\ -1$

2.1 $P_i = P_{i-2} + 2P$

3. $Q = \infty$, $i = length(NAF(k)) - 1$

4. while $i \geq 0$ *do*

4.1 if $k_i = 0$ *then* $t = 1$, $u = 0$

4.2 else

4.2.1 $t = 1$; $j = w$

4.2.2 while ($t = 1$ *and* $j > 1$) *do*

4.2.2.1 if $(k_i, \ldots, k_{i-j+1})$ *is odd* $t = j$; $u = (k_i, \ldots, k_{i-j+1})$

4.3 $Q = 2^t Q$

4.4 if $u > 0$ *then* $Q = Q + P_u$; *else if* $u < 0$ *then* $Q = Q - P_{-u}$

4.5 $i = i - t$

5. output(Q)

The average length of a run of zeros between windows in the NAF sliding window method is stated in [MoOl90] as

$$\nu(w) = \frac{4}{3} - \frac{(-1)^w}{3 \cdot 2^{w-2}}.$$

precomputation stage consists of the lines 0, 1 and 2. Line 1 costs $1D$ and line 2 costs $((2^w - (-1)^w)/3 \ - 1)A$. Thus, totally

$$(1D + \frac{2^w - (-1)^w}{3} - 1)A$$

is done at the precomputation stage. Next, lines 4.1, 4.2 are worthless. The line 4.3 is executed l times, hence costs lD. In order to find the running time of the line 4.4 we need to find the average number of nonzero terms and number of zeros between windows. The average length of a run of zeros between windows in the NAF sliding window method is

$$\nu(w) = \frac{4}{3} - \frac{(-1)^w}{3 \cdot 2^{w-2}}.$$

Then, expected cost of line 4.4 is

$$\frac{l}{w+\nu(w)}.$$

Therefore, expected running time of Algorithm 3.18 is

$$(1+l)D + (\frac{2^w-(-1)^w}{3} + \frac{l}{w+\nu(w)} - 1)A.$$

3.1.5 Montgomery's Ladder

Montgomery in [Mon87] presented a ladder method to perform fast exponentiation(scalar multiplication). After that, [LoDa99] presented the elliptic curve version of Montgomery's ladder. However, their ideas were applicable only for non-supersingular curves over binary fields. [OkSa01] extended Montgomery's ladder method for elliptic curves over non-binary fields. Hence, it became a generic method to compute scalar multiplication on elliptic curves.

The general idea of this method is starting from left-most bit of the scalar and a pair $(P,\ 2P)$ corresponding to left-most bit. Then, iterate to next left-most bit with a pair $(2(P),\ P+2P)$ (i.e. doubling first component of previous pair and addition of first and second component of previous pair) or $(P+2P,\ 2(2P))$ (i.e. addition of first and second component of previous pair and doubling second component of previous pair) if the next left-most bit of the scalar is 0 or 1, respectively. Continue this procedure until reaching to the last bit and naturally to the pair $(kP,\ (k+1)P)$. This idea is scalar multiplication equivalence of exponentiation. In the above iteration, it is enough to compute *x-coordinate* of both components for each pairs. Each iteration requires only an addition and a doubling.

Also, there is a shortcut for addition operation for elliptic curves over binary fields: For arbitrarily given $Q_i = (x_i, y_i)$ $i = 1, 2, 3, 4$ satisfying $Q_3 = Q_1 + Q_2$

and $Q_4 = Q_1 - Q_2$. Then,

$$x_3 = x_4 + \frac{x_2}{x_1 + x_2} + (\frac{x_2}{x_1 + x_2})^2.$$

In our case, $Q_1 = (l+1)P$, $Q_2 = lP$ and $Q_4 = P$ are given and we are asked to compute $(2l+1)P = Q_3$. Therefore, this shortcut can be applied at each iteration stage. Moreover, the *y-coordinate* of kP can be recovered, if needed, as:

$$y_1 = x^{-1}(x_1 + x)[(x_1 + x)(x_2 + x) + x^2 + y] + y,$$

where $kP = (x_1, y_1)$, $(k+1)P = (x_2, y_2)$ and $P = (x, y)$.

Algorithm 3.19. *[LoDa99]*
Montgomery scalar multiplication
input: $k = (k_{l-1}, \ldots, k_1, k_0),\ P \in E(F_q)$
output: kP
1. $X = P$ *and* $Y = 2P$
2. for i *from* $l-2$ *down to 0 do*
2.1 if $k_i = 0$ *then* $X = 2X$ *and* $Y = X + Y$
2.2 else $X = X + Y$ *and* $Y = 2Y$
3. output(X)

Algorithm 3.19 performs a doubling and an addition for each i whatever k_i is. Thus, expected running-time of the algorithm is $l(D + A)$. However, it is not needed to compute the y-coordinate of corresponding points until the last iteration. Hence, it is not possible to compare the running-time of this algorithm with the running-time of the others.

Example 3.20. *If* $26 = (11010)_2$ *is given,* $26P$ *is calculated by Montgomery's ladder by*

1	*1*	*0*	*1*	*0*
P	$P + 2P = 3P$	$2(3P) = 6P$	$6P + 7P = 13P$	$2(13P) = 26P$
$2P$	$2(2P) = 4P$	$3P + 4P = 7P$	$2(7P) = 14P$	$13P + 14P = 27P$

3.2 Fixed Point

Fixed point means that the point for which a scalar multiple is to be computed is known previously, and so the algorithm of scalar multiplication can be designed according to this privilege. For instance, if the point P is fixed and some storage is available, then some of the multiples of the P can be precomputed and saved to memory, and then used during the computation of scalar multiplication directly by memory call.

3.2.1 Fixed-base Windowing Method

If P is fixed, then the simplest idea that can be applied to the scalar multiplication is precalculation of all doublings of P up to 2^tP i.e. $2P$, $4P$, $8P$, ..., $2^{t-1}P$ where t is equal to approximately extension degree m of our finite field. Then, for any given scalar k, one can compute kP by summing up only 2^iP for which k_i is nonzero. Hence, all doublings are removed, and expected running time of binary algorithms decreases to $(l/2)A$.

A refinement to the above idea is first described in [Yao76] and more refinement is given in section 4.6.3 of [Knu81]. Finally, [BGMW92] proposed a patented version of previous ideas. The basic idea behind these refinements is the following equality:

$$kP = \sum_{i=0}^{l-1} k_i P = \sum_{j=1}^{L-1} (j \sum_{i:K_i=j} 2^{wi} P) \ where \ k = \sum_{i=0}^{d-1} K_i 2^{wi}.$$

We now define the BGMW's algorithm. Let

$$\begin{aligned} k = (k_{l-1}, \ldots, k_1, k_0)_2 &= (K_{d-1} || \ldots || K_0) \\ &= (K_{d-1}, K_{d-2}, \ldots, K_0)_{2^w} \\ &= \sum_{i=0}^{d-1} K_i 2^{wi} \end{aligned}$$

Then

$$\begin{aligned} kP &= \sum_{i=0}^{d-1} K_i 2^{wi} P \\ &= \sum_{i=0}^{d-1} K_i (2^{wi} P) \\ &= \sum_{j=0}^{2^w-1} (j \sum_{i:K_i=j} 2^{wi} P) \\ &= \sum_{j=1}^{2^w-1} j Q_j \\ &= Q_{2^{w-1}} + (Q_{2^{w-1}} + Q_{2^{w-2}}) + \ldots + (Q_{2^{w-1}} + Q_{2^{w-2}} + \ldots + Q_1). \end{aligned}$$

Then, the corresponding algorithm is coded as, first, calculate Q_j by cumulative addition of K_is and add Q_js together cumulatively to reach kP.

Algorithm 3.21. *[BGMW92]*
Binary BGMW's algorithm
input: $w, d = \lceil l/w \rceil$, $k = (K_{d-1}, K_{d-2}, \ldots, K_0)_{2^w}$, $P \in E(F_q)$
precomputed values: $P_i = 2^{wi} P$, $0 \leq i \leq d-1$
output: kP
1.$A = \infty, B = \infty$
2. for j from $2^w - 1$ *down to 1 do*
2.1 for i from 0 up to $d-1$ *do*
2.1.1 if $j = K_i$ *do* $B = B + P_i$
2.2 $A = A + B$
3. output(A)

The precomputation stage is not included into the running-time. Expected running-time of line 2.2.1 of Algorithm 3.21 is $(d/(2^w - 1))(2^w - 1)$ since the equivalence $j = K_i$ occurs expectedly $d/(2^w - 1)$ times and outer loop performs this expectancy $2^w - 1$ times. Consequently, the addition at the line 2.2.1 performs d times. In fact, it costs $(d-1)A$ if we discard the trivial first addition which is identity plus a point. Next, the line 2.2 performs $2^w - 1$ times, but costs

$(2^w - 2)A$ by discarding first trivial addition. Therefore, the expected running time of Algorithm 3.21 is $(2^w + d - 3)A$ where $d = \lceil l/w \rceil$

Algorithm 3.22 uses the same argument presented above with NAF representation of k instead of binary representation. $NAF(k) = (K_{d-1}, \ldots, K_1, K_0)$ where each K_i is in non-adjacent form. Thus, K_i can be at most

$$(1010 \ldots 1010)_{NAF_2} = \frac{2^{w+1} - 2}{3} \text{ or } (1010 \ldots 101)_{NAF_2} = \frac{2^{w+1} - 1}{3}$$

if w is even or odd, respectively.

Algorithm 3.22. *[Gor98]*
NAF BGMW's algorithm
input: $w, NAF(k), P \in E(F_q)$
precomputed values: $P_i = 2^(wi)P,\ 0 \leq i \leq \lceil (l+1)/w \rceil$
output: kP
1. $d = \lceil l/w \rceil$
2. if w is even $S = (2^{w+1} - 2)/3$ *and else* $S = (2^{w+1} - 1)/3$
3. $A = \infty,\ B = \infty$
4. for j from S to down to 1 do
4.1 for i from 0 up to d-1 do
4.1.1 if $j = K_i$ *do* $B = B + P_i$
4.1.2 else if $-j = K_i$ *do* $B = B - P_i$
4.1.3 $A = A + B$
5. output(A)

As the similar expectations of Algorithm 3.21 are made, the expected running time of Algorithm 3.22 is $(2^{w+1}/3\ + d - 2)A$ where $d = \lceil l/w \rceil$.

3.2.2 Fixed-base Comb Method

This method is also known as Lim-Lee Method. Let $d = \lceil l/w \rceil$. If necessary, pad on the left of the representation with zeros and then, split k into w

concatenated parts: $k = K^{w-1}, \ldots, K^0$.

$$\begin{aligned} kP &= \sum_{i=0}^{l-1} k_i 2^i P \\ &= k_0 P + k_1 2P + k_2 4P + ... + k_{t-1} 2^{l-1} P \\ &= k_0 P + ... + k_{d-1} 2^{d-1} P + k_{(w-1)d} 2^{(w-1)d} P + ... + k_{wd-1} 2^{wd-1} P \\ &= k_0 P + ... + (k_{d-1} P) 2^{d-1} + k_{(w-1)d} 2^{(w-1)d} P + ... + (k_{wd-1} 2^{(w-1)d} P) 2^{d-1}. \end{aligned}$$

The general idea of the method is operating on k column by column. Firstly, $k_{d-1}P + \ldots + k_{id}2^{id} + \ldots + k_{wd-1}2^{wd-1}$ is calculated and it is doubled. Secondly, $k_{d-2}P + \ldots + k_{id-1}2^{id-1} + \ldots + k_{wd-2}2^{wd-2}$ and added to first calculated and final result is doubled. And, it goes like this. Finally, $k_0P + \ldots + k_{id}2^i + \ldots + k_{(w-1)d}2^{(w-1)d}$ is calculated and added to previous sum. The obtained result is kP. In order to accelerate the computation, one can compute $[a_{w-1}, \ldots, a_2, a_1, a_0]P = a_{w-1}2^{(w-1)d} + \ldots + a_1 2^d P + a_0 P$ for all possible values of string $(a_{w-1}, \ldots, a_2, a_1, a_0)$. The exact algorithm is given below.

Algorithm 3.23. *[LL94]*
Fixed-base comb method point multiplication
input: $w, d = \lceil t/w \rceil, k = (k_{t-1}, \ldots, k_0)_2 = K^{w-1}|| \ldots ||K^0$ *where* K^i *is d-bit-integer and* K^i_s *is* s^{th} *bit of* K^i, $P \in E(F_q)$
Precomputed values: $[a_{w-1}, \ldots, a_2, a_1, a_0]P$ *for all possible strings* $(a_{w-1}, \ldots, a_0)$
output: kP
1. $Q = \infty$
2. for i from $d-1$ *down to 0 do*
2.1 $Q = 2Q$
2.2 $Q = Q + [K^{w-1}_i, K^{w-2}_i \ldots, K^2_i, K^1_i]P$
3. output(Q)

Probability of $[K^{w-1}_i, K^{w-2}_i \ldots, K^2_i, K^1_i]$ is a zero array is $1/2^w$. Hence, it is a non-zero array with a probability $1 - 1/2^w$ and non-infinity addition in the line 2.2 occurs expectedly $(\frac{2^w - 1}{2^w} d - 1)$ times (-1 comes from first infinity addition). Also, in the line 2.1 a non-infinity doubling is executed $(d-1)$ times(-1 comes

from first infinity doubling). Therefore, the expected running time of Algorithm 3.23 is

$$(\frac{2^w-1}{2^w}d-1)A+(d-1)D.$$

In the case of additional memory is available for the algorithm, two columns can be executed simultaneously. The idea is to divide the columns of k in to two parts. For both sides apply the previous procedure simultaneously. This idea is given in Algorithm 3.24

Algorithm 3.24. *[LL94]*
Fixed-base comb method point multiplication with two tables
input: $w, d = \lceil l/w \rceil, e = \lceil d/2 \rceil, k = (k_{l-1}, \ldots, k_0)_2 = K^{w-1}|| \ldots ||K^0$ *where* K^i *is d-bit-integer and* K^i_s *is* s^{th} *bit of* $K^i, P \in E(F_q)$
precomputed values: $[a_{w-1}, \ldots, a_2, a_1, a_0]P$ *and* $2^e[a_{w-1}, \ldots, a_2, a_1, a_0]P$ *for all possible string* $(a_{w-1}, \ldots, a_2, a_1, a_0)$
output: kP
1. $Q = \infty$
2. for i from e-1 down to 0 do
2.1 $Q = 2Q$
2.2 $Q = Q + [K^{w-1}_i, K^{w-2}_i \ldots, K^2_i, K^1_i]P + 2^e[a_{w-1}, \ldots, a_2, a_1, a_0]P$
3. output(Q)

Similar to the Algorithm 3.23, the expected time of Algorithm 3.24 can be calculated, and it is $(\frac{2^w-1}{2^w}2e-1)A+(e-1)D$. On the other hand, Algorithm 3.24 requires twice as much storage for precomputation as Algorithm 3.23. If memory is limited, then two algorithms can be compared for a given fixed amount of precomputation.

3.3 Curve Specific Methods

In this section, we will study scalar multiplication methods on some specific curves. These methods developed from nice properties of these curves. Specific

scalar multiplication methods are included into the many standards, e.g. NIST, IEEE, ISO, ANSI. The methods, studied in the previous two sections, are also applicable to the curves in this section. Moreover, generic methods are used with curve specific methods in order to decrease the computation time of the scalar multiplication.

3.3.1 Koblitz Curves

The non-supersingular curves defined over F_2 are called Koblitz curves[Kob92] also known as anomalous binary curves. Any non-supersingular curve over F_2 is isomorphic to one of the following two curves:

$$E_0 : y^2 + xy = x^3 + 1$$

$$E_1 : y^2 + xy = x^3 + x^2 + 1.$$

Hence, there exist only two non-isomorphic Koblitz Curves: E_0, E_1. By simply counting, $\#E_0(F_2) = 4$ and $\#E_1(F_2) = 2$. Then, $\#E_0(F_{2^m}) = 4n$ and $\#E_1(F_{2^m}) = 2n'$, for some n and $n' \in \mathbb{Z}$. In cryptography, n and n' are desired to be prime. n and n' can only be prime if m is prime; otherwise, there exist a subgroup $E_a(F_{2^d})$ of $E_a(F_{2^m})$ for any $d|m$.

3.3.2 $\mathbb{Z}[\tau]$ and τ-Representation

Let $a \in \{0, 1\}$. Then the Frobenius map on $E_a(F_{2^m})$ is

$$\begin{aligned} \tau : \quad & E_a(F_{2^m}) \to E_a(F_{2^m}) \\ & \tau(\infty) = \infty \\ & \tau(x, y) = (x^2, y^2). \end{aligned}$$

Squaring in a binary field, when polynomial base is used, requires only inserting zeros between the components and then reduction. It is very easy compared to

other operations. Furthermore, when normal base is used instead of polynomial base, it is just a circular shift. Therefore, computing Frobenius map for any binary field element is very fast.

From previous chapter, it is known that Frobenius map is an endomorphism over $E_a(F_{2^m})$. Its characteristic polynomial is $x^2 - tx + 2$ where t is trace of Frobenius and equals to $2+1-\#E_a(F_2)$. Explicitly, the characteristic polynomial of Frobenius map over $E_0(F_{2^m})$ is x^2+x+2, and similarly, over $E_1(F_{2^m})$ is x^2-x+2.

Let $\lambda = (-1)^{1-a}$. Then, the characteristic polynomial of Frobenius over E_a is $x^2 - \lambda x + 2$. Hence,

$$(\tau^2 - \lambda\tau + 2)P = 0 \ for\ all\ P \in E_a(F_{2^m}).$$

It can be observed that $\tau = (\lambda + \sqrt{-7})/2$ is one of the roots of the characteristic polynomial of Frobenius map over $E_a(F_{2^m})$.

We can naturally lift the action of Frobenius map to the action of the commutative ring

$$\mathbb{Z}[x]/(x^2 - \lambda x + 2) \cong \mathbb{Z}[\tau]\ by\ identifying\ x\ with\ \tau:\ E_a(F_{2^m}) \to E_a(F_{2^m})$$

This shows that the natural action of $\mathbb{Z}[\tau]$ over $E_a(F_{2^m})$ induces the action of the ring $\mathbb{Z}[\tau]$ on $E_a(F_{2^m})$:

$$(s_l\tau^l + \ldots + s_1\tau + s_0)P = s_l\tau^l(P) + \ldots s_1\tau(P) + s_0P.$$

In general, for an efficient scalar multiplication it is preferred to constitute the representation of the scalar as short as possible, and small-sparse digits are surely desired. In the next pages, the length and digits of the representation of a scalar are investigated.

Any element $\alpha \in \mathbb{Z}[\tau]$ can be written uniquely in the form $\alpha_0 + \alpha_1\tau$ for some integers α_0 and α_1 since $\tau^2 = \lambda\tau - 2$. In order to investigate the digits, we need to know the norm of an element in $\mathbb{Z}[\tau]$.

Definition 3.25. *The norm of $\alpha = \alpha_0 + \alpha_1\tau \in \mathbb{Z}[\tau]$ is the square of the absolute value of the complex number α. Namely,*

$$\begin{aligned} N(\alpha_0 + \alpha_1\tau) &= (\alpha_0 + \alpha_1\tau)(\alpha_0 + \alpha_1\overline{\tau}) \\ &= \alpha_0^2 + \lambda\alpha_0\alpha_1 + 2\alpha_1^2. \end{aligned}$$

Theorem 3.26. *[Sol00]*
Properties of the norm function
(i) $N(\alpha) \in \mathbb{Z}^+$ for all non-zero $\alpha \in \mathbb{Z}[\tau]$.
(ii) $N(\alpha) = 0$ if and only if $\alpha = 0$. Also, $N(\alpha) = 1$ if and only if $\alpha = \pm 1$.
(iii) $N(\tau) = 2$ and $N(\tau - 1) = 2^{2-a}$.
(iv) $N(\tau^m - 1) = \#E_a(F_{2^m})$ and $N((\tau^m - 1)/(\tau - 1)) = \#E_a(F_{2^m})/2^{2-a}$.
(v) $N(\alpha\beta) = N(\alpha)N(\beta)$ for all $\alpha, \beta \in \mathbb{Z}[\tau]$.
(vi) $\mathbb{Z}[\tau]$ is a Euclidean domain with respect to the norm function. That is for any $\alpha, \beta \in \mathbb{Z}[\tau]$ with $\beta \neq 0$, there exist $\kappa, \rho \in \mathbb{Z}[\tau]$(not necessarily unique) such that $\alpha = \kappa\beta + \rho$ and $N(\rho) < N(\beta)$.

Proof:
(i)Any $\alpha \in \mathbb{Z}[\tau] \subset \mathbb{C}$ can be written as $\alpha = a_0 + ia_1$ for some a_0 and $a_1 \in \mathbb{R}$. Then,

$$\begin{aligned} N(\alpha) &= \alpha\overline{\alpha} \\ &= (a_0 + ia_1)(a_0 - ia_1) \\ &= a_0^2 + a_1^2 \geq 0. \end{aligned}$$

(ii) $N(\alpha) = 0$ iff $a_0^2 + a_1^2 = 0$ iff $a_0 = a_1 = 0$ iff $\alpha = 0$.
Next, $N(\alpha) = 1$ iff $\alpha = \pm 1$:
For $\lambda = 1$, $\alpha_0^2 + \alpha_0\alpha_1 + 2\alpha_1^2 = 1$ iff $\alpha_0^2 + \alpha_0\alpha_1 + \frac{1}{4}\alpha_1^2 + \frac{7}{4}\alpha_1^2 = 1$ iff $(\alpha_0 + \frac{1}{2}\alpha_1)^2 + \frac{7}{4}\alpha_1^2 = 1$ iff $\alpha_1 = 0$ and $\alpha_0 = \pm 1$.
Similarly, for $\lambda = -1$, $\alpha_0^2 - \alpha_0\alpha_1 + 2\alpha_1^2 = 1$ iff $\alpha_1 = 0$ and $\alpha_0 = \pm 1$.

(iii) $\tau\overline{\tau} = 1/4 \ + \ 7/4 = 2$ gives $N(\tau) = 2$. Next,

$$\begin{aligned} N(\tau - 1) &= (\tau - 1)(\overline{\tau - 1}) \\ &= (\tau - 1)(\overline{\tau} - 1) \\ &= \tau\overline{\tau} - (\tau + \overline{\tau}) + 1 \\ &= N(\tau) - \lambda + 1 \\ &= 2 - \lambda + 1 \\ &= 3 - \lambda \\ &= 2^{2-a} \end{aligned}$$

(iv)

$$\begin{aligned} N(\tau^m - 1) &= (\tau^m - 1)(\overline{\tau^m - 1}) \\ &= N(\tau)^m - (\tau^m + \overline{\tau}) + 1 \\ &= 2^m + 1 - (\tau^m + \overline{\tau}) \\ &= \#E_a(F_{2^m}) \end{aligned}$$

Next,

$$\begin{aligned} N((\tau^m - 1)/(\tau - 1)) &= N(\tau^m - 1)/N(\tau - 1) \\ &= (\#E_a(F_{2^m}))/2^{2-a} \end{aligned}$$

(v)

$$\begin{aligned} N(\alpha\beta) &= (\alpha\beta)(\overline{\alpha\beta}) \\ &= \alpha\overline{\alpha}\beta\overline{\beta} \\ &= N(\alpha)N(\beta) \end{aligned}$$

(vi) see [Sol00].

Theorem 3.26 enables us to represent any positive integer k in terms of τ. Similar to binary representation of $k = \sum_{i=0}^{l'-1} k_i 2$ where $k_i \in \{0, 1\}$, τ-adic rep-

resentation of k can be obtained by repeatedly dividing k by τ and the digits u_i are remainders of the division steps. Since $N(\tau) = 2$, remainders are $-1,\ 0$ or 1. Then any positive integer k can be represented in τ-adic representation uniquely as $k = \sum_{i=0}^{l-1} u_i \tau^i$ where each digit $u_i \in 0, \pm 1$.

Any generic method studied in the previous sections can be applied to τ-adic representation of k. Firstly, in order to decrease the number of point additions, namely, decrease the number of non-zero digits, NAF method can be applied to this representation. It becomes so called τ-adic NAF or TNAF. τ-adic NAF is obtained in a similar way of 2-adic NAF.

Definition 3.27. *TNAF of an element $\kappa \in \mathbb{Z}[\tau]$ is $\kappa = \sum_{i=0}^{l-1} u_i \tau^i$ where each $u_i \in \{0, \pm 1\},\ u_{l-1} \neq 0$, and no two consecutive digits are nonzero. The length of the TNAF is l.*

Theorem 3.28. *[Sol00]*
Properties of TNAF
Let $\kappa \in \mathbb{Z}[\tau],\ \kappa \neq 0$

1. *κ has a unique TNAF denoted $TNAF(\kappa)$.*
2. *The average density of nonzero digits among all TNAF representations of length l is approximately $1/3$.*
3. *If the length $l(\kappa)$ of $TNAF(\kappa)$ is greater than 30, then*

$$log_2(N(\kappa)) - 0.55 < l(\kappa) < log_2(N(\kappa)) + 3.52.$$

Computation of $TNAF(\kappa)$ is similar to computation of NAF of an integer. That is , the i^{th} right-most digit of $TNAF(\kappa)$ is r which is the remainder of repeatedly i^{th} division of κ by τ, but at that step if r is not zero than it is chosen the element for which $(\kappa - r)/\tau$ is divisible by τ, ensuring that the next TNAF digit is 0. As it is observed, in order to generate $TNAF(\kappa)$, division of an element of $\mathbb{Z}[\tau]$ by τ or τ^2 is to be computed.

Theorem 3.29. *[Sol00]*
Division by τ and τ^2 in $\mathbb{Z}[\tau]$
Let $\alpha = r_0 + r_1\tau \in \mathbb{Z}[\tau]$

i. α is divisible by τ if and only if r_0 is even, in this case

$$\alpha/\tau = (r_1 + \lambda r_0/2) - (r_0/2)\tau.$$

ii. α is divisible by τ^2 if and only if $r_0 \equiv 2r_1 \ (mod\ 4)$.

Proof:

i. $\alpha = r_0 + r_1\tau$ is divisible by τ iff r_0 is divisible by τ (since $r_1\tau$ is divisible by τ) iff $N(r_0)$ is divisible by $N(\tau)$ iff $r_0^2 (= N(r_0))$ is divisible by $2 (= N(\tau))$ iff r_0 is even. Next, if r_0 is even,

$$\begin{aligned} \alpha/\tau &= (r_0 + r_1\tau)/\tau \\ &= r_0/\tau + r_1 \\ &= r_0\overline{\tau}/\tau\overline{\tau} + r_1 \\ &= r_0(-\tau + \lambda)/2 + r_1 \\ &= (r_1 + \lambda r_0/2) - (r_0/2)\tau \ \textit{as desired.} \end{aligned}$$

ii. $\alpha = r_0 + r_1\tau$ is divisible by τ^2 iff α/τ is divisible by τ iff 2 divides $r_1 + \lambda r_0/2$ (by previous item) iff 2 divides $(2r_1 + \lambda r_0)/2$ iff 4 divides $2r_1 + \lambda r_0$ iff $2r_1 \equiv \lambda r_0 \ (mod\ 4)$ iff $2r_1 \equiv r_0 (mod\ 4)$ (since λ is either 1 or -1) as desired.

The following algorithm computes the $TNAF(\kappa)$ for any $\kappa \in \mathbb{Z}[\tau]$.

Algorithm 3.30. *[Sol00]*
Computing $TNAF(\kappa)$
input: $\kappa = r_0 + r_1\tau \in \mathbb{Z}[\tau]$
output: $TNAF(\kappa)$.
1. $i = 0$

2. while $r_0 \neq 0$ *or* $r_1 \neq 0$ *do*
2.1 if r_0 *is odd then*
2.1.1 $u_i = 2 - (r_0 - 2r_1 \ mod \ 4)$
2.1.2 $r_0 = r_0 - u_i$
2.2 else $u_i = 0$
2.3 $t = r_0$
2.4 $r_0 = r_1 + \lambda r_0/2$
2.5 $r_1 = -t/2$
2.6 $i = i + 1$
3. output $(u_{l-1}, u_{l-2}, \ldots, u_1, u_0)$

By Theorem 3.28 (iii), the length of $TNAF(k)$ for any integer k is approximately $log_2 N(k) = log_2(k^2) = 2log_2 k$, which is twice the length of $NAF(k)$. This is an undesirable situation. The next observation reduces the length of $TNAF(k)$.

If the degree of the extension of binary field is m, then by the properties of Frobenius $\tau : E_a(F_{2^m}) \rightarrow E_a(F_{2^m})$, we have:

$$(\tau^m - 1)(P) = \tau^m(P) - P = P - P = \infty \ \forall P \in E_a(F_{2^m}).$$

Then, if $k \equiv \gamma \ (mod \ \tau^m - 1)$ then $kP = \gamma P$ for any integer k, and

$$length(\gamma) \approx length(\tau^m - 1) \approx m \approx length(NAF(k)).$$

That is, length of $TNAF(k)$ is approximately reduced to the length of $NAF(k)$ [MeSt93]. Furthermore, observe that

$$(\tau^m - 1)(P) = (\frac{\tau^m - 1}{\tau - 1}(\tau - 1))(P)$$

and it was observed that left hand side is ∞ for all $P \in E_a(F_{2^m})$. So is right hand side. Then,

$$(\frac{\tau^m - 1}{\tau - 1})(P) = \infty \ if \ (\tau - 1)(P) \neq \infty.$$

Therefore, if $k \equiv \delta \ (mod\ (\tau^m - 1)/(\tau - 1))$, then $kP = \delta P$ for all $P \in E_a(F_{2^m})$ except $\{P \mid \tau(P) = P\}$. In deed, for any $P \in E_a(F_{2^m})$, $\tau(P) = P$ iff $P \in E_a(F_2)$. Thus, $length(\delta) \approx length((\tau^m - 1)/(\tau - 1)) \approx n$ by Theorem 3.26.

In order to compute the modular reduction $\delta \equiv k \ (mod\ \rho)$, the division algorithm on $\mathbb{Z}[\tau]$ is needed to be defined, and the reduction should be done to the smallest norm. Algorithm 3.31 computes the division algorithm in $\mathbb{Z}[\tau]$. In step 5, it needs the rounding of an element in $\mathbb{Z}[\tau]$, which is performed by Algorithm 3.32.

Algorithm 3.31. *[HMV04]*
Division Algorithm in $\mathbb{Z}[\tau]$
input: $\alpha = a_0 + a_1\tau,\ \beta = b_0 + b_1\tau \in \mathbb{Z}[\tau]$ *with* $\beta \neq 0$.
output: $\kappa = q_0 + q_1\tau,\ \delta = r_0 + r_1\tau \in \mathbb{Z}[\tau]$ *with* $\alpha = \kappa\beta + \rho$ *and* $N(\delta) \leq \frac{4}{7}N(\beta)$.
1. $g_0 = a_0b_0 + \lambda a a_0b_1 + a_1b_1$
2. $g_1 = a_1b_0 - a_0b_1$
3. $N = b_0^2 + \lambda b_0b_1 + 2b_1^2$
4. $\mu_0 = g_0/N, \mu_1 = g_1/N$
5. Use Algorithm 3.32 to compute $(q_0, q_1) = Round(\mu_0, \mu_1)$
6. $r_0 = a_0 - b_1q_0 + 2b_1q_1$
7. $r_1 = a_1 - b_1q_0 - b_0q_1 - \lambda b_1q_1$
8. $\kappa = q_0 + q_1\tau$
9. $\delta = r_0 + r_1\tau$
10. output (κ, δ)

Algorithm 3.32. *[HMV04]*
Rounding off in $\mathbb{Z}[\tau]$
input: Rational numbers μ_0 *and* μ_1
output: integers q_0, q_1 *such that* $q_0 + q_1\tau$ *is close to complex number* $\mu_0 + \mu_1\tau$
1. for $i = 0$ *and* 1
1.1 $f_i = \lfloor \mu_i + \frac{1}{2} \rfloor$, $\eta_i = \mu_i - f_i, h_i = 0$
2. $\eta = 2\eta_0 + \lambda\eta_1$
3. if $\eta \geq 1$ *then*
3.1 if $\eta_0 - 3\lambda\eta_1 < -1$ *then* $h_1 = \mu$

3.2 else $h_0 = 1$

4. else

4.1 if $\eta_0 + 4\lambda\eta_1 \geq 2$ *then* $h_1 = \mu$

5. if $\eta < -1$ *then*

5.1 if $\eta_0 - 3\lambda\eta_1 \geq 1$ *then* $h_1 = -\mu$

5.2 else $h_0 = -1$

6. else

6.1 if $\eta_0 + 4\lambda\eta_1 < -2$ *then* $h_1 = -\mu$

7. $q_0 = f_0 + h_0$

8. $q_1 = f_1 + h_1$

9. output (q_0, q_1)

In the fourth line of Algorithm 3.31, there exist two multi precision integer divisions costing too much. Algorithm 3.33 computes modular reduction in $\mathbb{Z}[\tau]$ without expensive multi precision integer divisions. $k \equiv \delta'$ *partmod* ρ is the operation of the algorithm. [Sol00] proved that $length(\delta) \leq m + a$ and if $C \geq 2$ then $length(\delta') \leq m + a + 3$. In fact, $\delta' = \delta$ with a probability $1 - 2^{-(C-5)}$, hence a sufficiently large C ensures that they are probable equivalent.

Algorithm 3.33. *[Sol00]*

Partial reduction modulo $\rho = (\tau^m - 1)/(\tau - 1)$

input: $k \in [1, n-1]$, $C \geq 2$, $s_0 = d_0 + \lambda d_1$, $s_1 = -d_1$, *where* $\rho = d_0 + d_1\tau$

output: $\delta' = k$ *partmod* ρ

1. $k' = \left\lfloor k/(2^{a-C+(m-9)/2}) \right\rfloor$

2. $V_m = 2^m + 1 - \#E_a(F_{2^m})$

3. for $i = 0$ *and* 1 *do*

3.1 $g' = s_i k'$

3.2 $j' = V_m \lfloor g'/2^m \rfloor$

3.3 $u_i = \left\lfloor (g' + j')/2^{(m+5)/2} + 1/2 \right\rfloor /2^C$

4. Use Algorithm 3.32 to compute $(q_0, q_1) = Round(\mu_0, \mu_1)$

5. $r_0 = k - (s_0 + \lambda s_1)q_0 - 2s_1 q_1$

6. $r_1 = s_1 q_0 - s_0 q_1$

7. output $(r_0 + r_1\tau)$

To sum up, in order to calculate kP one can calculate, equivalently, δP where $\delta \equiv k \ (mod \ (\tau^m - 1)/(\tau - 1))$ and $length(\delta) < log_2(k)$ which is desirable. Next, scalar multiplication is going to be performed by using TNAF.

3.3.3 Scalar Multiplication on Koblitz Curves

Since a specific coding of k, that is TNAF(k), is obtained , generic methods are applicable in this case. Algorithm 3.34 performs a scalar multiplication by using previously observed properties of the curve and TNAF.

Algorithm 3.34. *[Sol00]*
TNAF scalar multiplication on Koblitz curves
input: integer $k \in [1, n-1]$, $P \in E(F_{2^m})$ *of order* n
output: kP
1. compute $\delta' = k$ *partmod* ρ *by Algorithm 3.33*
2. compute $TNAF(\delta') = \sum_{i=0}^{l-1} u_i \tau^i$ *by Algorithm 3.30*
3. $Q = \infty$
4. for i *from* $l-1$ *down to 0*
4.1 $Q = \tau Q$
4.2 if $u_i = 1$ $Q = Q + P$
4.3 if $u_i = -1$ $Q = Q - P$
5. output(Q)

Running-time of Algorithm 3.34 is $\frac{l}{3}A$ since τQ is a very fast calculation and the weight of $TNAF(k)$ is 1/3.

3.3.4 Window TNAF Method

Window method can be applied to TNAF(k) to increase the speed of the algorithm and to secure it against the simple power attack. Similar to width-w NAF method, width-w TNAF method processes w digits of δ' at a time. However, obtaining width-w TNAF is different, and needs the following observation.

Theorem 3.35. *[Sol00] Let $\{U_k\}$ be the integer sequence defined by $U_0 = 0$, $U_1 = 1$, $U_{k+1} = \lambda U_k - 2U_{k-1}$ for $k \geq 1$. Then,*

i. $U_k^2 - \lambda U_{k-1}U_k + 2U_{k-1}^2 = 2^{k-1}$ *for* $k \geq 1$

ii. Let $t_k = 2U_{k-1}U_k^{-1} (mod\ 2^k)$ *for* $k \geq 1$. $t_k^2 + 2 \equiv \lambda t_k\ (mod\ 2^k)$ *for all* $k \geq 1$.

iii. The map $\phi_w : \mathbb{Z}[\tau] \rightarrow \mathbb{Z}_{2^w}$, $\tau \rightarrow t_w$, *is a surjective ring homomorphism with kernel* $\{\alpha \in \mathbb{Z}[\tau] : \tau^w \text{ divides } \alpha\}$.

iv. $\{0, \pm 1, \pm 2, \pm 3, \ldots, \pm(2^{w-1} - 1), -2^{w-1}\}$ *is the set of equivalence classes of* $\mathbb{Z}[\tau]$ *modulo* τ^w.

Theorem 3.35 enables us to construct the definition of $TNAF_w$:

Definition 3.36. *Let $w \geq 2$ be a positive integer, $\kappa \in \mathbb{Z}[\tau]$. Define $\alpha_i = i$ mod τ^w for $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$. $TNAF_w(\kappa)$ is defined as $\kappa = \sum_{i=0}^{l-1} u_i \tau^i$ where $u_i \in \{0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}\}$, $u_{l-1} \neq 0$, and at most one of any w consecutive digits is nonzero. The length of the width-w TNAF is l.*

The digits of $TNAF_w(\delta)$ are remainders obtained by repeatedly dividing δ by τ, and when δ is not divisible by τ, the remainder is chosen α_u from the set $\{\pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}\}$ where $u = \phi_w(\delta)$ *mods* 2^w. Hence, if a nonzero digit is obtained, then $(\delta - \alpha_u)/\tau$ will be divisible by τ^{w-1}, ensuring that the next $w - 1$ digits are 0.

Algorithm 3.37. *[Sol00]*
Computing a width-w TNAF of an element in $\mathbb{Z}[\tau]$
input: w, t_w, $\alpha_u = \beta_u + \gamma_u\tau$ for $u \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$, $\delta = r_0 + r_1\tau \in \mathbb{Z}[\tau]$
output: $TNAF_w(\delta)$
1. $i = 0$
2. while $r_0 \neq 0$ *or* $r_1 \neq 0$ *do*
2.1 if r_0 *is odd*
2.1.1 $u = r_0 + r_1 t_w$ *mods* 2^w
2.1.2 if $u > 0$ *then* $s = 1$

2.1.3 else $s = -1$, $u = -u$

2.1.4 $r_0 = r_0 - s\beta_u$, $r_1 = r_1 - s\gamma_u$

2.1.6 $u_i = s\alpha_u$

2.2 else $u_i = 0$

2.3 $t = r_0$

2.4 $r_0 = r_1 + \lambda r_0/2$, $r_1 = -t/2$

2.6 $i = i + 1$

3. output $(u_{l-1}, u_{l-2}, \ldots, u_1, u_0)$

By using $TNAF_w(\delta')$, scalar multiplication can be done with a similar algorithm of NAF_w.

Algorithm 3.38. *[Sol00]*
window TNAF scalar multiplication method for Koblitz curves
input: window width w, *integer* k *in* $[1, n-1]$, $P \in E_a(F_{2^m})$ *of order* n
output: kP
1. use Algorithm 3.33 to compute $\delta' = k$ *partmod* ρ
2. use Algorithm 3.37 to compute $TNAF_w(\delta') = \sum_{i=0}^{l-1} u_i \tau^i$
3. compute $P_u = \alpha_u P$, $u \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
4. $Q = \infty$
5. for $i = l - 1$ *down to 0*
5.1 $Q = \tau Q$
5.2 if $u_i \neq 0$
5.2.1 let u *be such that* $\alpha_u = u_i$ *or* $\alpha_{-u} = -u_i$
5.2.2 if $u > 0$ *then* $Q = Q + P_u$
5.2.3 else $Q = Q - P_{-u}$
6. output(Q)

Precomputation stage of Algorithm 3.38 costs $(2^{w-2} - 1)A$ and while loop costs approximately $\frac{m}{w+1}A$. Thus, expected running-time of Algorithm 3.38 is

$$(2^{w-2} - 1 + \frac{m}{w+1})A.$$

3.4 Endomorphism Based Scalar Multiplication

Scalar multiplication can be performed faster by using a special endomorphism of the curves. In section 2.4, general treatment of the endomorhisms was studied. In addition, examples of endomorphism of elliptic curves were given. Now, we will build bridges on to the section 2.4.

Example 3.39. *Let $p \equiv 1 (mod\ 3)$ be a prime, and $E : y^2 = x^3 + b$ over F_p. Let β be a 3^{rd} root of unity in F_p. Define $\alpha : E(\overline{K}) \mapsto E(\overline{K})$*

$$\alpha(x, y) = (\beta x, y),\ \alpha(\infty) = \infty.$$

is an endomorphism of E. The characteristic polynomial of α is $f(x) = x^2+x+1$.

In general, let ϕ be an endomorphism of an elliptic curve $E(F_q)$ and let $\#E(F_q)$ is divisible by a prime r, but not by r^2. Then by the properties of commutative groups, there exist only one subgroup of order r of $E(F_q)$. Having prime order implies it is a cyclic subgroup. Let it be generated by P. Then, $\phi(P)$ has order r since ϕ is an endomorphism. Hence, $\phi(P) \in < P >$, that is, $\phi(P) = \lambda P$ for some $\lambda \in [1, r-1]$. In fact, λ is a root modulo r of characteristic polynomial of ϕ. A good example of this situation is used in the Wireless Transport Layer Security standard:

Example 3.40. *Let $E :\ y^2 = x^3 + 3$ defined over F_p where $p = 2^{160} - 229233$. $p \equiv 1\ (mod\ 3)$ So, E satisfies the requirements of Example 3.39, $\phi(x, y) = (\beta x, y)$ is an endomorphism of E/F_p.*

Let $\beta = 771473166210819779552257112796337671037538143582$
whose order is 3. Furthermore,

$\#E(F_p) = 1461501637330902918203687013445034429194588307251$
is a prime number. So, $r = \#E(F_p)$, and the solution

$\lambda = 903860042511079968555273866340564498116022318806$
to the equation $x^2 + x + 1 \equiv 0\ (mod\ r)$ satisfies $\phi(P) = \lambda P$ for all $P \in E(F_p)$ since its cyclic subgroup is itself.

We can apply above observations to scalar multiplication kP as follows: we find the λ expansion of k. However, since λ is too large, it is desirable to represent k as $k = k_1 + k_2\lambda$ (mod r) where k_1 and k_2 have approximately half length of k. Then, $kP = k_1P + \lambda k_2P = k_1P + \phi(k_2P)$. Computing $\phi(k_2P)$ is easy, it is just one field multiplication. So, it reduces to computing k_1P and k_2P. After finding k_1 and k_2, applying a interleaved right-to-left scalar multiplication method to k_1P and k_2P reduces number of doubling operations approximately to half. The gain is considerable if finding k_1 and k_2 for a given k and λ, which is called decomposition of k, can be computed efficiently.

3.4.1 Decomposition of scalar

It is needed to find k_1 and k_2 satisfying $k = f(k_1, k_2) = k_1 + k_2\lambda(mod\ r)$ and number of bits of k_1 and k_2 are approximately half of the number of bits of k, meaning k_1 and k_2 are small or $\sqrt{k_1^2 + k_2^2}$ is small. Thus, the aim is to find a short vector u such that $f(u) = k$. Trivial solution is $v = (k, 0)$, but this is not short. The approach is the following:

(1) find two vectors $v_1 = (a_1, b_1)$ and $v_2 = (a_2, b_2)$ in $\mathbb{Z} \times \mathbb{Z}$ satisfying

(i) v_1 and v_2 are linearly independent over $\mathbb{R}$.

(ii) $f(v_1) = f(v_2) = 0$.

(iii) v_1 and v_2 are short itself, that is $\sqrt{a_i^2 + b_i^2}$ is small since a_i and b_i have half bits of k which can be at least r, then it is approximately $\sqrt{r}$.

(2) find a vector v in the integer lattice generated by v_1 and v_2 that is close to $(k, 0)$. Then, $u = (k, 0) - v$ is a short vector with

$$f(u) = f((k, 0)) - f(v) = k - 0 = k.$$

Note that subproblems (1) and (2) can be solved using lattice basis reduction algorithms. However, we will give [GLV01]'s method, which is much more faster.

First, extended Euclidean algorithm (EEA) applied to r and λ can be used to find v_1 and v_2 satisfying (i), (ii) and (iii). EEA produces a sequence of equations

$$s_i r + t_i \lambda = r_i \ for \ i = 0, 1, 2..$$

where $s_0 = 1$, $t_0 = 0$, $v_0 = n$, $s_1 = 0$, $t_1 = 1$, $r_1 = \lambda$ and $r_i \geq 0 \ \forall i$.

properties of EEA:

- $r_i > r_{i+1} \geq 0 \ \forall i \geq 0$.
- $|s_i| < |s_{i+1}$ for $i \geq 1$.
- $|t_i| < |t_{i+1}|$ for $i \geq 1$.
- $r_{i-1}|t_i| + r_i|t_{i-1}| = n \ \forall i \geq 1$.

Then, choose m be the greatest index for which $r_m \geq \sqrt{n}$. We know $r_m|t_{m+1}| + r_{m+1}|t_m| = r$ by last properties of EEA. Thus, $|t_{m+1}| < \sqrt{n}$. Then, choose $v_1 = (r_{m+1}, -t_{m+1})$. $f(v) = (r_{m+1} - t_{m+1}\lambda) \ mod \ r = 0$ by EEA. Next, $|v_1| = \sqrt{r_{m+1}^2 + t_{m+1}^2} < \sqrt{n+n} \approx \sqrt{n}$ meaning short, as desired.

Then, choose v_2 as to be the shorter of $(r_{m+2}, -t_{m+2})$, $(r_m, -t_m)$ Similarly, by EEA $f(v_2) = 0$ and, heuristically, v_2 is short.

Next, it comes to show that v_1 and v_2 are linearly independent:
Assume v_1 and v_2 are linearly dependent and without lost of generality let v_2 be $(r_m, -t_m)$. Then,

$$\frac{r_{m+1}}{r_m} = \frac{t_{m+1}}{t_m}.$$

LHS is strictly less than 1 by first properties of EEA, but RHS is strictly greater than 1 by third properties of EEA. Therefore, v_1 and v_2 are linearly independent. To sum up, v_1 and v_2 satisfy (i), (ii) and (iii).

Next, in order to solve the subproblem (2), one needs to follow the following routine:

Consider v_1 and $v_2 \in \mathbb{Q} \times \mathbb{Q}$.

Find β_1 and $\beta_2 \in \mathbb{Q}$ satisfying $(k,0) = \beta_1 v_1 + \beta_2 v_2$. By linear algebra, it is easy, and $\beta_1 = b_2 k/r$ and $\beta_2 = -b_1 k/r$.

Let $c_1 = \lfloor \beta_1 \rceil$ and $c_2 = \lfloor \beta_2 \rceil$ where $\lfloor x \rceil$ is the nearest integer to x.

Finally, let $v = c_1 v_1 + c_2 v_2$.

Then, $u = (k,0) - v$ is the needed short vector:

If v is constructed as above, then

$$\begin{aligned} u &= (k,0) - v \\ &= (k,0) - (c_1 v_1 + c_2 v_2) \\ &= (k,0) - [(\beta_1 v_1 + \beta_2 v_2) + (c_1 - \beta_1)v_1 + (c_2 - \beta_2)v_2] \\ &= (\beta_1 - c_1)v_1 + (\beta_2 - c_2)v_2 \textit{ since } (\beta_1 v_1 + \beta_2 v_2) \textit{ is close to } (k,0). \end{aligned}$$

So,

$$\begin{aligned} |u| &\leq |(\beta_1 - c_1)v_1| + |(\beta_2 - c_2)v_2| \textit{ by triangle inequality} \\ &\leq \frac{1}{2}|v_1| + \frac{1}{2}|v_2| \textit{ since } |(\beta_1 - c_1| < \frac{1}{2} \textit{ and } |(\beta_2 - c_2| < \frac{1}{2} \\ &\leq max(|v_1|, |v_2|) \end{aligned}$$

Moreover, since both v_1 and v_2 are short vectors, and so is u.

Algorithm 3.41. *[HMV04]*
Balanced length-two representation of a scalar
input: Integers n, λ, $k \in [0, n-1]$
output: integers k_1 *and* k_2 *such that* $k = k_1 + k_2\lambda \bmod r$ *and* $|k_1|, |k_2| \approx \sqrt{r}$
1. use the extended Euclidean algorithm with inputs r *and* λ *to produce equations* $s_i r + t_i \lambda = r_i$ *where* $s_0 = 1$, $t_0 = 0$, $r_0 = r$, $s_1 = 0$, $t_1 = 1$, $r_1 = \lambda$ *and* r_i *sequence is non-negative and strictly decreasing. Let* m *be the greatest index for which* $r_l \geq \sqrt{r}$.
2. $(a_m, b_m) = (r_{m+1}, -t_{m+1}$
3. if $(r_m^2 + t_m^2) \leq (r_{m+2}^2 + t_{m+2}^2)$ *then* $(a_2, b_2) = (r_m, -t_m)$
4. else $(a_2, b_2) = (r_{m+2}, -t_{m+2})$

5. $b_1 = \lfloor \beta_1 \rceil$ *and* $b_2 = \lfloor \beta_2 \rceil$

6. $k_1 = k - c_1 a_1 - c_2 a_2$ *and* $k_2 - c_1 b_1 - c_2 b_2$

7. output(k_1, k_2)

Example 3.42. *One can check that if we apply Algorithm 3.41 to Example 3.40, we obtain*

$(r_m, t_m) = (218072875140953865599 3509, -186029539167685199353061)$

$(r_{m+1}, t_{m+1}) = (788919430192407951782190, 602889891024722752429129)$

$(r_{m+2}, t_{m+2}) = (602889891024722752429129, -139180932121713070421 1319)$

$(a_1, b_1) = (788919430192407951782190, -602889891024722752429129)$

$(a_2, b_2) = (602889891024722752429129, 139180932121713070421 1319)$

Now, let

$$k = 965486288327218559097909069724275579360008398257$$

then

$$c_1 = 919446671339517233512759 \text{ and } c_2 = 398276613783683332374156$$

So,

$$k_1 = -98093723971803846754077, \text{ and } k_2 = 381880690058693066485147$$

Algorithm 3.43. *[HMV04]*

Endomorphism based Scalar Multiplication

input: integer $k \in [1, n-1]$, $P \in E(F_q)$, w_1 *and* w_2*(window widths) and* λ

output: kP

1. use Algorithm 3.41 to find k_1 *and* k_2

2. for $j = 1$ *and* 2 *do*

2.1 Use Algorithm 3.13 to find $NAF_{w_j}(|k_j|) = \sum_{i=0}^{l_j - 1} k_{ji} 2^i$

3. $l = maxlength(NAF_{w_1}(k_1), NAF_{w_2}(k_2))$

4. equalize their lengths by padding zero to shorter one

5. if $k_j < 0$ *then* $k_{ji} = -k_{ji}$ $j = 1, 2$

6. precomputation iP *for* $i \in 1, 3, 5, 7, \ldots, 2^{w_j - 1} - 1$ $j = 1, 2$

7. Similarly precompute for $\phi(P)$

8. $Q = \infty$

9. for i *from* $l - 1$ *down to 0 do*

9.1 $Q = 2Q$
9.2 for $j = 1, 2$ *do*
9.2.1. if $k_{ji} > 0$ *then* $Q = Q + k_{ji}P_j$
9.2.2. else $Q = Q - |k_{ji}|P$
10. output(Q)

When running-time of Algorithm 3.43 is analyzed, cost of the decomposition operation and also cost of computing $\phi(P)$ must be included if it costs fairly (e.g. much more than Frobenius). Line 6 is the precomputation stage and it costs $\sum_{j=1}^{2}(2^{w_j-2}-1)A$. For each k_j NAF_w method is applied in lines of 8 and it cost $(D + \sum_{j=1}^{2}\frac{1}{w_j+1}A)l/2$. Therefore, running-time of Algorithm 3.43 is

$$\sum_{j=1}^{2}(2^{w_j-2}-1)A + C_k + C_\phi + (D + \sum_{j=1}^{2}\frac{1}{w_j+1}A)l/2$$

if C_k denotes the cost of line 1 and C_ϕ denotes the cost of $\phi(P)$.

Remark 3.44. *One can think of applying the method of Koblitz to the curve of Example 3.40. Roots of characteristic equation is* $x^2 + x + 1 \equiv 0$ *are* $\frac{1 \pm \sqrt{-3}}{2}$. *Let* κ *be* $\frac{1+\sqrt{-3}}{2}$. *Then the extension of integers with* κ, $\mathbb{Z}[\kappa]$, *is a Euclidean Domain which is a desired feature since division algorithm can be applied directly in order to obtain* κ*-representation of an integer. But the norm of* κ *is 1. So, any element of* $\mathbb{Z}[\kappa]$ *is divisible by* κ. *Hence, an integer* k *doesn't have a* κ*-representation. We remember that for a Koblitz curve,* τ *has a norm of 2 which was an ideal case. Moreover, Frobenius map on Koblitz curves over some small extensions of binary field giving fast scalar multiplication are investigated by [Mül98]. On the other hand, GLV method can be applied to Koblitz curves. The characteristic equation is needed to be solved modulo large prime divisor in order to get* λ *as in the Example 3.40.*

Chapter 4

Conclusion

In this thesis, we studied the scalar multiplication on elliptic curves. We made mathematical and computational analysis of many scalar multiplication methods. The implementation part of this work has not been done in this thesis, and we hope to do this as a part of BAP project.

In order to develop a scalar multiplication method or to improve an existing one, one, first, needs to analyze properties of elliptic curves in deep, second, to consider whether a recoding method of an integer corresponds to an efficient scalar multiplication method, and finally, to use some algebraic methods on elliptic curves.

In Table 4.1, there is a comparison of scalar multiplication in terms of their number of additions, doublings and memory consumption.

Table 4.1: Comparison of Scalar Multiplication Methods

Algorithm	Additions	Doublings	Memory
3.1	$\frac{l}{2}$	l	1
3.2	$\frac{l}{2}$	l	-
3.11	$\frac{l}{3}$	l	-
3.17	$2^{w-2} + \frac{l}{w+1} - 1$	$l+1$	2^{w-2}
3.18	$\frac{2^w-(-1)^w}{3} + \frac{l}{w+\nu(w)} - 1$	$l+1$	2^{w-2}
3.21	$(2^w + d - 3)$, $d = \lceil \frac{l}{w} \rceil$	-	$\lceil \frac{l}{w} \rceil$
3.22	$(\frac{2^{w+1}}{3} + d - 2)$, $d = \lceil \frac{l+1}{w} \rceil$	-	$\lceil \frac{l+1}{w} \rceil$
3.23	$(\frac{2^w-1}{2^w}d - 1)$, $d = \lceil \frac{l}{w} \rceil$	$(d-1)$	2^w
3.24	$(\frac{2^w-1}{2^w}2e - 1)$, $e = \lceil \frac{l}{2w} \rceil$	$(e-1)$	2^{w+1}
3.34	$\frac{l}{3}$	-	-
3.38	$2^{w-2} - 1 + \frac{l}{w+1}$	-	2^{w-2}
3.43	$\sum_{j=1}^{2}(2^{w_j-2} - 1) + (\sum_{j=1}^{2}\frac{1}{w_j+1})l/2$	$\frac{l}{2}$	$2^{w_1-2} + 2^{w_2-2}$

l refers to $\lceil log_2(k) \rceil$) and names of algorithms given in the left most column are
Algorithm 3.1 Binary Method RtoL
Algorithm 3.2 Binary Method LtoR
Algorithm 3.11 NAF Method
Algorithm 3.17 wNAF Method
Algorithm 3.18 Sliding NAF Method
Algorithm 3.21 Binary Windowing Method
Algorithm 3.22 NAF Windowing Method
Algorithm 3.23 Comb Method with 1 Table
Algorithm 3.24 Comb Method with 2 Tables
Algorithm 3.34 TNAF Method
Algorithm 3.38 wTNAF Method
Algorithm 3.43 Endomorphism Based Method (GLV's Method).

References

[Ava04] R. M. Avanzi. *A Note on the Sliding Window Integer Recoding and its Left-To-Right Analogue*, Workshop on Selected Areas in Cryptography SAC 2004, LNCS 3357, 130-143. Springer-Verlag, 2005.

[BSS99] I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, Cambridge, UK, 1999.

[BGMW92] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson, *Fast exponentiation with precomputation*, Advances in Cryptology Proceedings of Eurocrypt 92, LNCS 658, 200207. Springer-Verlag, 1992.

[Coh93] H. Cohen, *A Course in Computational Number Theory*, Springer-Verlag, 1993.

[Enge99] A. Enge, *Elliptic Curves and Their Applications to Cryptography: An Introduction*, Kluwer Academic Publishers, 1999.

[Gor98] D. M. Gordon, *A survey of fast exponentiation methods*, J. Algorithms, 27(1):129-146, 1998.

[GLV01] R. Gallant, R. Lambert and S. Vanstone, *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms*, LNCS 2139, CRYPTO 2001.

[HMV04] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag, New York, 2004.

[JoYe00] M. Joye and S. M. Yen, *Optimal left-to-right binary signed-digit recoding*, IEEE Transactions on Computers, 49(7):740-748, 2000.

[Knu81] D.E. Knuth, *Seminumerical Algorithms, volume 2 of The Art of Computer Programming*, Addison Wesley, 2nd edition, 1997.

[Kob94] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, 2nd edition, 1994

[Kob87] N. Koblitz, *Elliptic curve cryptosystems*,Mathematics of Computations, **48**, 203-209 (1987)

[Kob92] N. Koblitz, *CM-curves with good cryptographic properties*, in: Advances in Cryptology, CRYPTO 91, LNCS 576, 279287, 1992.

[LiNi93] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, Cambridge University Press, 1993

[LL94] C. H. Lim and P. J. Lee, *More flexible exponentiation with precomputation*, Advances in Cryptography, Crypto'94, LNCS 839, 95-107, Springer-Verlag, 1994.

[Lint82] J. H. van Lint, *Introduction to Coding Theory*, Springer-Verlag, 1982.

[LoDa99] J. Löpez and R. Dahab, *Fast multiplication on elliptic curves over GF(2^m) without precomputation* CHES 1999 LNCS 1717 316327. Springer-Verlag, 1999.

[MeSt93] W. Meier, O. Staffelbach, *Efficient multiplication on certain nonsupersingular elliptic curves*, Advances in Cryptology (CRYPTO 92), LNCS 740, 333344, 1993.

[Men93] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, (1993)

[Mil85] V. S. Miller, *Uses of Elliptic Curves in Cryptography*, Advances in Cryptology: Proceeding of Crypto '85, Lecture Notes in Computer Science, Springer-Verlag, **218**, 417-426, 1986

[MOV93] A. Menezes, T. Okamoto, S. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Trans. Inform. Theory IT-39 1639-1646 1993.

[MOV96] A. Menezes, P. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, 1996.

[Mon87] P. L. Montgomery, *Speeding the Pollard and Elliptic curve Methods of Factorization*, Mathematics of Computation, 48(170):243264, 1987.

[MoOl90] F. Morain and J. Olivos, *Speeding up the computations on an elliptic curve using addition-subtraction chains*, Inform. Theor. Appl., 24:531543, 1990.

[Möl02] B. Möller, *Improved Techniques for Fast Exponentiation*, LNCS 2587, 298312, 2002.

[MuSt04a] J. A. Muir and D. R. Stinson, *Minimality and other properties of the width-w nonadjacent form*, To appear in Mathematics of Computation.

[MuSt04b] J.A. Muir, and D. R. Stinson, *New MinimalWeight Representations for Left-to-RightWindow Methods*, Technical Report CACR 2004-03, Centre for Applied Cryptographic Research.

[Mül98] V. Müller, *Fast Multiplicaitonon Elliptic Curves over Small Fields of Characteristic two*, Journal Cryptology, 11:219-234, 1998

[OkSa01] K. Okeya and K. Sakurai, *Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery form elliptic curve*, CHES 2001, LNCS 2162 126141. Springer-Verlag, 2001.

[OSST04] K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi, *Signed Binary Representations Revisited*, Proceedings of Crypto 2004.

[Rei60] G. Reitwiesner, *Binary Arithmetic*, Adv. Comput. 1: 231-308, 1960.

[Sch96] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Codes in C*,Wiley, 2nd Ed, 1996.

[Sil86] J. H. Silverman, *The Arithmetic of Elliptic Curves*, Springer-Verlag, (1986)

[Sol00] J. Solinas, *Efficient arithmetic on Koblitz curves*, Designs, Codes and Cryptography 19, 195249, 2000.

[Sti02] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press, 2nd Ed, 2002.

[Was03] L. C. Washington, *Ellptic Curves: Number Theory and Cryptography*, CRC Press, 2003.

[Yao76] A. C. Yao, *On the Evaluation of Powers*, SIAM J. Comput. 5, 100-103, 1976