YAŞAR UNIVERSTIY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER OF SCIENCE THESIS

Efficient Implemantation Techniques of Elliptic Curve Cryptography

Author

Hülya EVKAN

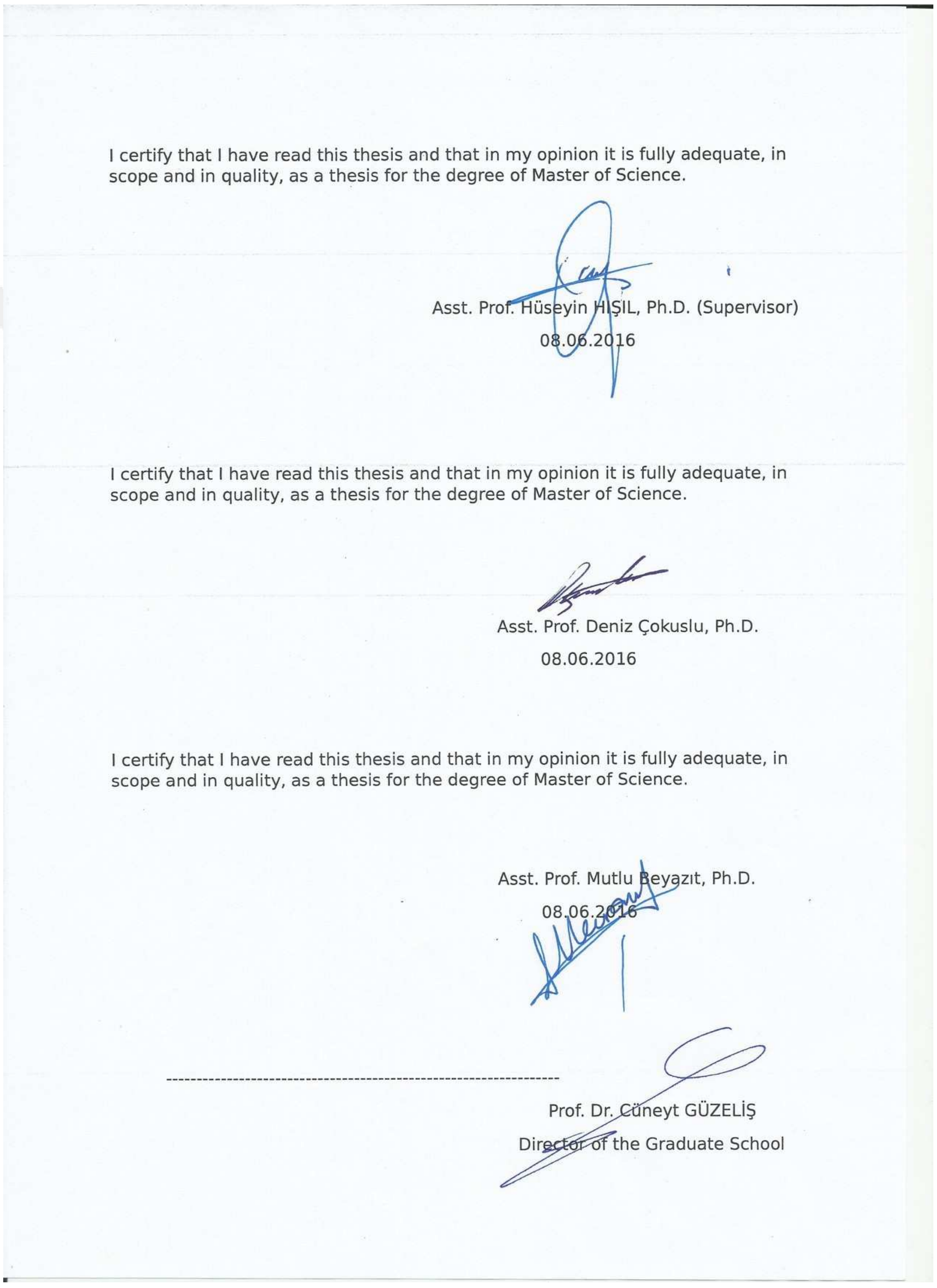
Academic Advisor

Asist. Prof. Hüseyin HIŞIL, Ph.D.

Izmir, 2016

Keywords

Cryptography, elliptic curves, scalar multiplication, NAF representation, windowed exponentiation, efficiently computable endomorphism, GLV method, GLS endomorphism, scalar decomposition, operation counting, complexity analysis, magma, computer algebra system, group law, point doubling, point addition, projective coordinates.



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

[illegible]

Asst. Prof. Hüseyin HIŞIL, Ph.D. (Supervisor)

08.06.2016

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

[illegible]

Asst. Prof. Deniz Çokuslu, Ph.D.

08.06.2016

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

Asst. Prof. Mutlu Beyazıt, Ph.D.

08.06.2016

[illegible]

[illegible]

Prof. Dr. Cüneyt GÜZELİŞ

Director of the Graduate School

Öz

Eliptik eğri tabanlı kriptografi, diğer kriptosistemlerle kıyaslandığında çok daha küçük anahtarlara ve dolayısıyla çok daha az belleğe ihtiyaç duyar. Bu sebeple son on yılda elliptik eğri tabanlı kriptografi üzerine yapılan çalışmalar artmıştır. Skalar çarpma işlemi eliptik eğri tabanlı kriptosistemlerin uygulanmasındaki temel aritmetik işlemidir. Yüksek hız için tasarlanan uygulamalar, skalar çarpmanın mevcut en düşük karmaşıklık düzeyine sahip algoritmalarının kullanılmasıyla gerçekleştirilmektedir. Dolayısıyla skalar çarpma işlemenin minimum karmaşıklık düzeyi önemli bir bilimsel araştırma alanı oluşturmuştur. Bu tez, bilinen en hızlı skalar çarpma algoritmalarının araştırılmasını, uygulanmasını ve kıyaslanmasını içermektedir.

Abstract

Since elliptic curve cryptology needs smaller keys and less memory for higher speed than other crypto systems, elliptic curve cryptography researches increased in last twenty years. Scalar multiplication is the primitive operation in the implementation of elliptic curve cryptosystems. Implementations, which are designed for high speed, are realized with scalar multiplication algorithms which have minimum complexity. Because of this reason, minimum complexity of scalar multiplication became an important research area. This thesis includes review, implementation and comparison of known fastest scalar multiplication algorithms.

Acknowledgements

I would like to express my special thanks and gratitude to my supervisor Hüseyin Hışıl for his guidance, encouragement and patience to me and my endless questions during my master. He didn't let me to get drowned between code lines as a matematician. I thank to him to make this journey fun for me. It was the great joy to study with him.

I would like to thank my mother and sister. They gave me lots of love, supported me in every case, and thought about me more than themselves. Without them I can't do this, specially to my mother. She is the greatest power behind me. Thanks mom!

Contents

List of Codes

Chapter 1

Introduction

In 1985, Neal Kolbitz [22] and Victor Miller [26] proposed elliptic curve cryptography (ECC) independently. That proposal was a public-key cryptosystem which is based on some properties of a special equation that created from a finite field [2] [13]. Multiplying a point of elliptic curve with a number will give another point over the elliptic curve but it is really hard to find the number that was used at the first step even if we know the first point and the point that we got as the result of multiplication.

At the end of 90's and the beginning of 00's the U.S. National Institute of Standards and Technology (NIST) endorsed ECC among their set of recommended algorithms. The U.S. National Security Agency (NSA) also allowed their use for protecting information which is classified up to top secret [31]. With these improvements ECC algorithms entered wide use.

In 2001, Dan Boneh and Matthew K. Franklin developed identity-based encryption schemes on elliptic curves [3]. These schemes perform probabilistic encryption of arbitrary chiphertexts using an approach like Elgamal.

Beside all of this chronology, ECC used in many different places. Since nineteenth century, algebraists, number theorists and algebraic geometers are studying elliptic curves. Hendrik Lenstra described an algorithm to factor integers in subexponential time (1985). This algorithm relies on properties of elliptic curves. Shafi Goldwasser proposed an idea called elliptic curve primality proving

(ECPP) which is testing the primality of general numbers (1986). Schoof's algorithm is a deterministic polynomial time algorithm that counts points on elliptic curves and published by René Schoof first time (1985) [33]. Elliptic curve gained more importance with the emergence and post of these algorithms and the discovery of relationships between elliptic curve complex multiplication, algebraic number theory and modular forms and elliptic curves.

Whitfield Diffie and Martin Hellman proposed a key exchange protocol, which is also the beginning of public-key cryptography, in 1976 [12]. It is an asymmetric cryptographic system which is widely used in secure electronic communication.

The main idea of public-key cryptography is same with one way functions. Basically, there are functions whose inverse functions are computed in large amount of time. If we use such a function to encrypt, it will be hart to decrypt even if the function is public knowledge. DH uses Discrete Logarithm Problem(DLP) as its asymmetric operation. Basically DLP focuses finding a logarithm of a number within a finite field arithmetic system.

In 1977, Ron Rivest, Adi Shamir and Len Adleman proposed RSA cryptosystem. The RSA algorithm is the most popular and best understood public-key cryptosystem. It is based on the intractability of the integer factorization problem. It is easy to multiply two numbers where thay are quite large prime numbers, but it is almost impossible to factor the product of two large primes except if there is some incredible stroke of luck, or bad choice of the primes.

ECC is also a public-key cryptosystem as RSA with a difference. Security of ECC and RSA comes from the hardness of different problems. Security of ECC is based on the hardness of the elliptic curve discrete logarithm problem (ECDLP). The best algorithms known to solve this problem have exponential running time, in contrast to the subexponential time algorithms known for the integer factorization problem. This means we can use shorter keys for security levels where RSA would need much bigger keys and using today's algorithms and computer technologies it seems possible to factor 700-bit numbers [6] [8] [21] [37]. That is why we prefer to use ECC even if RSA is the most popular and best understood public-key cryptosystem.

1.1 Motivation

As described above, because elliptic curve scalar multiplication constitutes bottleneck of lots of algorithms, gathering the fastest scalar multiplication methods has been a good subject for research [7]. This thesis aims to research and integrate state-of-art methods.

1.2 Outline

This thesis contains five chapter. First chapter is the introduction chapter. After it, chapters built as a pyramid. Every chapter includes basics for next chapter.

Elliptic curve chapter, chapter 2, gives some basic algebric structures over elliptic curves which we will need while we are building other chapters using these structures like group law and group order, defines endomorphisms and shows elliptic curve operations on projective coordinates.

Chapter 3 is about point multiplication on elliptic curves. This chapter will show different point multiplication algorithms, their algebraic background and superficial complexity analysis.

Next chapter, chapter 4, includes more efficient algorithm for point multiplications. It has two main parts. First part explains faster point multiplication which uses endomorphisms. Other part gives special endomorphisms which make this algorithm faster.

And the last chapter, we give the comparisons of algorithms and explain the implimentations that we did using MAGMA [4] [1].

1.3 Literature Review

The discrete logarithm problem is defined as follows in the multiplicative group ${\mathbb{Z}_q}^*$; let q be a prime, let a and b be the given elements of the group then find a number k such that $a \equiv b^k \pmod{q}$. If we adapt this to elliptic curves, let E be an elliptic curve over a finite field. Suppose there are points P and Q on E then find k such that $Q = [k]P$.

Elliptic curves can be represented with couple of different types of equations. We will focus on simplified Weierstrass equations and group law over them. Especially, Weierstrass elliptic curves which are defined over a finite field of characteristic $\neq 2, 3$. Hence elliptic curve discrete logarithm problem, our focus point is $[k]P$ which is called as point multiplication or scalar multiplication. Basically, point multiplication is adding two points. If these points are different from each other, we call this as point addition. If we add same points, we call this as point doubling.

The cost of inversion is more expensive than multiplication. Since point addition and point doubling formulas contain inversion, we will use Jacobian or mixed coordinates instead of using only affine coordinates. This will help to speed up point multiplication algorithms little more.

Gallant, Lambert and Vanstone's method (GLV) speeds up point multiplication of elliptic curves which have an efficiently computable endomorphism. The method also applies to curves defined over large prime fields. GLV method uses efficiently computable endomorphism ϕ to rewrite $[k]P$ as $[k]P = [k_1]P + [k_2]\phi(P)$. This method is faster point multiplication technique as long as chosen curve has efficiently computable endomorphisms. Hence only special curves can be used with GLV method. At this point GLS endomorphisms give chance to work over large classes.

1.4 Aim, Objectives and Deliverables

In this thesis, our aim is gathering faster point multiplication techniques and endomorphisms together and finding its cost using complexity analysis and operation counting.

As deliverable, MAGMA [4] implementations of algorithms are in the Appendix.

Chapter 2

Elliptic Curves

2.1 Preliminaries

This thesis focuses on elliptic curves over finite fields. We make necessary definitions and provide theorems to define an elliptic curve over a finite field [14] [9] [11].

Definition 1. *A **group** is a set G, closed under a binary operation $+$, such that the following axioms are satisfied;*

- *The binary operation $+$ is **associative** such that $x + (y + z) = (x + y) + z$ for all $x, y, z \in G$.*
- *There is an element 0 in G such that $0 + x = x + 0 = x$ for all $x \in G$ and this element is **the identity element** of G for $+$.*
- *For each x in G, there is a x', **the inverse element**, in G with the property that $x+x' = x'+x = 0$.*

Example 1. *$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ are some of familiar group examples.*

Definition 2. *A group G is **abelian** if $+$ is commutative property that $x + y = y + x$ for all $x, y \in G$.*

Definition 3. *A **ring** is a set R together with additive operation $+$ and multiplicative operation $\cdot$ defined on R such that following axioms are satisfied;*

- *R is an abelian group under $+$.*
- *Multiplication is associative.*
- *For all elements, **the distributive law** holds in R for both left and right sides. That is $x \cdot (y+z) = x \cdot y + x \cdot z$ for all $x, y, z \in R$.*

Definition 4. *Let R be a ring. If for R, a positive integer n exists such that $na = 0$ for all $a \in R$, then the least such positive integer is the **characteristic** of the ring R. If there is no such positive integer exists, then R is of characteristic 0.*

Note that we shall be using characteristic mostly for fields.

Definition 5. *Let R_1 and R_2 be rings and let $\phi \colon R_1 \to R_2$ be a homomorphism. Then,*

$$\phi^{-1}[0] = \{r \in R_1 | \phi(r) = 0\}$$

*is the **kernel** of ϕ, denoted by $\ker(\phi)$.*

Definition 6. *A ring K is a **field** if the set of non-zero elements of K forms an abelian group under multiplication.*

Example 2. *$\mathbb{C}$, $\mathbb{R}$, $\mathbb{Q}$ are examples of fields with ordinary $+$ and $\cdot$ operations.*

A **finite field** or **galois field** is a field that contains a finite number of elements [24]. The simplest example of finite fields is a prime field which is denoted by $GF(p)$ or $\mathbb{F}_p$ where p is a prime number.

Definition 7. *A **subfield** is a subset of the field that is a field under induced operations from the whole field.*

Definition 8. *Let L and K fields. L is an **extension field** of K if L is the subfield of K and denoted K/L.*

Theorem 1. *Let K be a field and let L be an extension field of K. Then*

$$\bar{K} = \{a \in L | a \text{ is algebraic over } K\},$$

*is a subfield of L, the **algebraic closure** of K in L.*

Definition 9. *Let K be a field. K is* ***algebraically closed*** *if every nonconstant polynomial in $K[x]$ has a zero in K.*

Definition 10. *Let K be a field. E is an elliptic curve over K defined by an equation*

$$E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

where $a_i \in K$ and $\Delta \neq 0$ is the discriminant of E and is defined as follows:

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

and $j(E)$ is called the ***j-invariant*** *of E.*

Equation 1 is called a **Weierstrass equation**.

If L is any extension field of K then the set of *L-rational points* on E is

$$E(L) = \{(x, y) \in L \times L\colon y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\infty\}$$

where ∞ is the **point at infinity**.

Example 3. *These are elliptic curve examples;*

- $y^2 + 8xy + 6y = x^3 + 4x^2 + 3$ *over* $\mathbb{Q}$.
- $y^2 = x^3 + 132x + 27$ *over* $\mathbb{F}_{157}$.
- $y^2 = x^3 + 36318x + 718621$ *over* $\mathbb{F}_{742981}$.

Definition 11. *Let R_1 and R_2 be rings. A map $\phi\colon R_1 \to R_2$ is a* ***homomorphism*** *if the following two conditions are satisfied for all $a, b \in R_1$;*

- $\phi(a+b) = \phi(a) + \phi(b)$,
- $\phi(ab) = \phi(a)\phi(b)$.

Definition 12. *Let R_1 and R_2 be rings. An* ***isomorphism*** *$\phi\colon R_1 \to R_2$ from R_1 to R_2 is a homomorphism that is one to one and onto R_2. The rings R_1 and R_2 are then* ***isomorphic****.*

Definition 13. *Let E_1 and E_2 be two elliptic curves over field K defined by Weierstrass equations*

$$E_1\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$
$$E_2\colon y^2 + \bar{a}_1xy + \bar{a}_3y = x^3 + \bar{a}_2x^2 + \bar{a}_4x + \bar{a}_6$$

Then E_1 and E_2 are isomorphic over K if there exist $u, r, s, t \in K$, $u \neq 0$ such that the change of variables

$$(x, y) \longrightarrow (u^2x + r, u^3y + u^2sx + t) \tag{2}$$

transforms E_1 to E_2 and this is called ***the admissible change of variables*** *[13].*

2.2 Simplified Weierstrass Equations

Section 2.1 gives a generalized form of Weierstrass in Equation 1. But we can simplify a Weierstrass equation to have less number of terms by applying an admissible changes of variables. We consider that there are three different cases where K has characteristic equal to 2 or 3 or not equal 2 and 3 [34] [35].

Case 1. If $\mathrm{char}(K) \neq 2$ and $\mathrm{char}(K) \neq 3$, the admissible change of variables

$$(x, y) \longrightarrow \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1x}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{24}\right) \tag{3}$$

transforms E to curve

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

where $a, b \in K$ and the discriminant of this simplified equation is $\Delta = -16(4a^3 + 27b^2)$.

Case 2. If $\mathrm{char}(K) = 2$, then there are two different cases that depend on value of a.

- If $a \neq 0$, then the admissible change of variables

$$(x, y) \longrightarrow \left(a_1^2 x + \frac{a_3}{a_1}, a_1^3 + \frac{a_1^2 a_4 + a_3^2}{a_1^3}\right)$$

  transforms E to curve

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

  where $a, b \in K$ and the discriminant of curve is $\Delta = b$. These kind of curves are called as **non-supersingular**.

- If $a = 0$, then the admissible change of variables

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

  transforms E to the curve

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

  where $a, b, c \in K$ and the discriminant of curve is $\Delta = c^4$. These kind of curves are called as **supersingular**.

Case 3. If $\mathrm{char}(K) = 3$, then there are two different cases.

- If $a_1^2 \neq -a_2$, then the admissible change of variables

$$(x, y) \longrightarrow \left(x + \frac{d_4}{d_2}, y + a_1 x + a_1 \frac{d_4}{d_2} + a_3\right) \tag{4}$$

where $d_2 = a_1^2 + a_2$ and $d_4 = a_4 - a_1 a_3$, transforms E to the curve

$$y^2 = x^3 + ax^2 + b \tag{5}$$

where $a, b \in K$. This curve is called as **non-supersingular** and its discriminant is $\Delta = -a^3 b$.

- If $a_1^2 = -a_2$, then the admissible change of variables

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

transforms E to the curve

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

where $a, b \in K$. This curve is called as **supersingular** and its discriminant is $\Delta = -a^3$.

2.3 Group Law

Let K be a field. For simplicity, assume that $\mathrm{char}(K) \neq 2, 3$ in the remainder of this thesis. Let E be an elliptic curve defined over K. There is a construction called **chord-and-tangent rule** for adding two points in $E(K)$ that give a third point in $E(K)$. The set of points $E(K)$ builds a group with this point addition rule which also includes ∞ as identity of the group.

Let P and Q be points on $E(K)$. If $P \neq Q$ then $P + Q$ is defined as follows:

- Connect P and Q with a line l.
- l intersects $E(K)$ at a third point A.
- Reflection of A about the x-axis is the sum $R = P + Q$.

If $P = Q$, this special case is called as **point doubling** and the sum R is defined as follows:

- Draw the tangent line l to $E(K)$ at P which by definition of a tangent line intersects the curve two times.
- l intersects $E(K)$ at a third point A.

- Reflection of A about the x-axis is the sum $R = [2]P$.

Let's investigate algebraically.

Group Law for $E/K\colon y^2 = x^3 + ax + b$, $\mathrm{char}(K) \neq 2, 3$

Let $P = (x_1, y_1)$ be a point in $E(K)$. ∞ is the identity, such that

$$P + \infty = \infty + P = P \tag{6}$$

for all $P \in E(K)$. The negative of P point is $-P$, and satisfies $(x_1, y_1) + (x_1, -y_1) = (x_1, -y_1) + (x_1, y_1) = \infty$.

Let $Q = (x_2, y_2)$ be a point on the elliptic curve E where Q is not equal P or negative of P. Then $P + Q = (x_3, y_3)$ where

$$(x_3, y_3) = \left(\left(\frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2, \left(\frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3) - y_1 \right). \tag{7}$$

This is called as point addition. When we want to double a point, Equation (7) is useless because of vanishing denominators. There exists a special another formula for point doubling. If $P \neq -P$ then $[2]P = (x_3, y_3)$ where

$$(x_3, y_3) = \left(\left(\frac{3x_1^2 + a}{2y_1} \right)^2 - 2x_1, \left(\frac{3x_1^2 + a}{2y_1} \right) (x_1 - x_3) - y_1 \right). \tag{8}$$

Remark 1. *Following steps include all possibilities addition of two points for $E/K\colon y^2 = x^3 + ax + b$ elliptic curve [5]:*

- *If $x_1 = x_2$ but $y_1 \neq y_2$ then $R = P + Q = \infty$.*
- *If $x_1 = x_2$ and $y_1 = 0 = y_2$ then $R = P + Q = \infty$.*
- *If $x_1 = x_2$ and $y_1 = y_2 \neq 0$ then $R = [2]P$.*
- *If $x_1 \neq x_2$ then $R = P + Q$*

Example 4. *Let $p = 31$, $a = 2$, $b = 11$, and consider the elliptic curve*

$$E\colon y^2 = x^3 + 2x + 11$$

defined over $\mathbb{F}_{31}$. We know that E is an elliptic curve because

$$\begin{aligned}\Delta &= -16(4a^3 + 27b^2)\\ &= -16(4\cdot 2^3 + 27\cdot 11^2)\\ &= -52784 \neq 0\end{aligned}$$

The points in $E(\mathbb{F}_{31})$ are following:

$$\begin{gathered}\{\infty, (1,13), (1,18), (9,13), (9,18), (10,15), (10,16),\\ (11,0), (13,8), (13,23), (16,4), (16,27), (18,12), (18,19),\\ (21,13), (21,18), (22,15), (22,16), (23,14), (23,17), (25,0),\\ (26,0), (27,1), (27,30), (28,3), (28,28), (30,15), (30,16)\}\end{gathered}$$

Here is addition and doubling examples for this curve:

$$(9,13) + (23,14) = (27,30)$$

$$[2](9,13) = (9,13) + (9,13) = (27,1)$$

There are two more group law parts for supersingular and non-supersingular elliptic curves but in this thesis, we will work on $E/K\colon y^2 = x^3 + ax + b$ elliptic curve. Therefore, we won't give group laws for other elliptic curves. In addition, we will continue to work with $y^2 = x^3 + ax + b$ curve in the next sections.

2.4 Group Order

Let E be an elliptic curve defined over $\mathbb{F}_q$. **Order** of E is the number of points in $E(\mathbb{F}_q)$ and denoted as $\#E(\mathbb{F}_q)$. Since there are two $y \in \mathbb{F}_q$ for each $x \in \mathbb{F}_q$ satisfying Equation (3), we can say that $\#E(\mathbb{F}_q) \in [1, 2q+1]$.

Theorem 2 (Hasse)**.** *Let E be an elliptic curve defined over $\mathbb{F}_q$. Then*

$$q + 1 - 2\sqrt{q} \leq E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q}$$

Theorem 2 gives us tighter bounds, and interval $[q+1-2\sqrt{q}, q+1+2\sqrt{q}]$ is called **the Hasse interval**. There is another denotation for Hasse's theorem.

Theorem 3. *Let E be an elliptic curve defined over $\mathbb{F}_q$. Then $\#E(\mathbb{F}_q) = q + 1 - t$ where $|t| \leq 2\sqrt{q}$; t is called the **trace** of E over $\mathbb{F}_q$.*

Here, t is a small relative q so we can say $\#E(\mathbb{F}_q) \approx q$. To know this is important for us because we would like to a have large prime divisor of $\#E(\mathbb{F}_q)$ to ensure the security of the elliptic curve cryptosystem. Hasse's theorem reassures that there are approximately q points for E.

Definition 14. *Let p be the characteristic of $\mathbb{F}_q$. An elliptic curve E defined over $\mathbb{F}_q$ is **supersingular** if p divides t, where t is the trace. If p doesn't divide t, then E is **non-supersingular**.*

2.5 Endomorphisms

Definition 15. *[34] Let K be a field, $\bar{K}$ its algebraic closure, and E be an elliptic curve over K. An **endomorphism** of E is a rational map $\phi\colon E(\bar{K}) \longrightarrow E(\bar{K})$ given by rational functions. In other words, if ϕ satisfies following conditions;*

- *$\phi(\infty) = \infty$,*
- *$\phi(P + Q) = \phi(P) + \phi(Q)$,*

and there are rational functions (i.e quotients of polynomials) $R_1(x, y)$, $R_2(x, y)$ with coefficients in $\bar{K}$ such that for each $P = (x, y) \in E(\bar{K})$:

$$\phi(x, y) = (R_1(x, y), R_2(x, y)),$$

then phi is an endomorphism of E.

Example 5. *[18] Let E be an elliptic curve an elliptic curve over $\mathbb{F}_q$. For each $m \in \mathbb{Z}$ **multiplication by m** map $[m]\colon E \longrightarrow E$ defined by $P \mapsto [m]P$ is an endomorphism defined over $\mathbb{F}_q$. A special case is the **negation** map defined by $P \mapsto -P$.*

Example 6. *[19] Let E be given by $y^2 = x^3 + ax + b$, and let $\phi(P) = 2P$ (multiplication by 2). Then ϕ is a homomorphism and we have*

$$\phi(x, y) = (R_1(x, y), R_2(x, y)).$$

where

$$\begin{aligned} R_1(x,y) &= \left(\frac{3x^2+a}{2y}\right)^2 - 2x \\ R_2(x,y) &= \left(\frac{3x_1^2+a}{2y_1}\right)(x_1 - x_3) - y_1 \end{aligned}$$

are obtained by using Equation (8). Since ϕ is a homomorphism given by rational functions, it is an endomorphism of E.

Example 7. *[19] [38] Let E be an elliptic curve defined over $\mathbb{F}_q$. Then the q-th power map $\phi\colon E \longrightarrow E$ defined by*

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

*is an endomorphism of E defined over $\mathbb{F}_q$, called the **Frobenius endomorphism**. The characteristic polynomial of ϕ is $X^2 - tX + q$, where $t = q + 1 - \#E(\mathbb{F}_q)$.*

Example 8. *[9] Let $q \equiv 1 \pmod 4$ be a prime, and consider the elliptic curve*

$$E\colon y^2 = x^3 + ax$$

defined over $\mathbb{F}_q$. Let $i \in \mathbb{F}_q$ be an element of order 4. Then the map $\phi\colon E \longrightarrow E$ defined by

- $\phi\colon (x, y) \mapsto (-x, iy)$
- $\phi\colon \infty \mapsto \infty$

is an endomorphism of E defined over $\mathbb{F}_q$. The characteristic polynomial of ϕ is X^2+1.

Example 9. *[9] Let $q \equiv 1 \pmod 3$ be a prime, and consider the elliptic curve*

$$E\colon y^2 = x^3 + b$$

defined over $\mathbb{F}_q$. Let $\beta \in (\mathbb{F}_q)^*$ be an element of order 3. Then the map $\phi\colon E \longrightarrow E$ defined by

- $\phi\colon (x,y) \mapsto (\beta x, y)$
- $\phi\colon \infty \mapsto \infty$

is an endomorphism of E defined over $\mathbb{F}_q$. The characteristic polynomial of ϕ is X^2+X+1.

2.6 Projective Coordinates

In section 2.3, we gave point addition and point doubling formulas for elliptic curve $E\colon y^2 = x^3 + ax + b$ defined over a field K ([34]). The formulas 7 and 8 need an inversion and few multiplications to make point addition and point doubling. However, the cost of inversion in field K can be much more expensive than multiplication. In this case, using projective coordinates and Jacobian coordinates eliminates the disadvantage of slow inversions.

Definition 16. *Let K be a field and let c and d be positive integers. We can define an equivalence relation $\sim$ on the set $K^3 \setminus \{(0,0,0)\}$ of nonzero triples over K such that*

$$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \text{ if } X_1 = \lambda^c X_2, Y_1 = \lambda^d Y_2, Z_1 = \lambda Z_2 \text{ for some } \lambda \in K^*.$$

The equivalence class that includes $(X, Y, Z) \in K^3 \setminus \{(0,0,0)\}$ is denoted by

$$(X : Y : Z) = \{(\lambda^c X_2, \lambda^d Y_2, \lambda Z_2) : \lambda \in K^*\}$$

and $(X : Y : Z)$ is called a projective point, (X, Y, Z) is called a representative of $(X : Y : Z)$.

Notice that if $Z \neq 0$ then $(X/Z^c, Y/Z^d, 1)$ is a representative of the projective point $(X : Y : Z)$ because any element of an equivalence class can serve as its representative.

Jacobian coordinates is one type of projective coordinates such that $c = 2$ and $d = 3$. Hence $(X : Y : Z)$ corresponds $(X/Z^2, X/Y^3)$ for $Z \neq 0$. The projective equation of $y^2 = x^3 + ax + b$ Weierstrass equation as is

$$Y^2 = X^3 + aXZ^4 + bZ^6.$$

Let $(1 : 1 : 0)$ implies ∞ as the identity of E. The negative form of $(X : Y : Z)$ is $(X : -Y : Z)$.

Definition 17 (Point Doubling)**.** *Let $P = (X_1 : Y_1 : Z_1) \in E$ and $P \neq -P$. We can write $(X_1/Z_1^2 : X_1/Y_1^3 : 1)$ instead of $(X_1 : Y_1 : Z_1)$ for $Z_1 \neq 0$. Hence the point doubling formula of E in affine coordinates can be used to calculate* $[2]P = (X_3^{'} : Y_3^{'} : 1)$.

$$\begin{aligned} X_3^{'} &= \left(\frac{3\frac{X_1^2}{Z_1^4}+a}{2\frac{Y_1}{Z_1^3}}\right)^2 - 2\frac{X_1}{Z_1^2} \\ &= \frac{(3X_1^2+aZ_1^4)^2-8X_1Y_1^2}{4Y_1^2Z_1^2}, \end{aligned}$$

$$\begin{aligned} Y_3^{'} &= \left(\frac{3\frac{X_1^2}{Z_1^4}+a}{2\frac{Y_1}{Z_1^3}}\right)\left(\frac{X_1}{Z_1^2}-X_3\right)-\frac{Y_1}{Z_1^3} \\ &= \frac{3X_1^2+aZ_1^4}{2Y_1Z_1}\left(\frac{X_1}{Z_1^2-X_3}\right)-\frac{Y_1}{Z_1^3}. \end{aligned}$$

To eliminate denominators and to obtain linear formulas, we set $X_3 = X_3^{'}Z_3^2$ *and* $Y_3 = Y_3^{'}Z_3^3$ *where* $Z_3 = 2Y_1Z_1$*. This gives*

$$X_3 = (3X_1^2+aZ_1^4)^2-(8X_1Y_1^2), \tag{9}$$

$$Y_3 = (X_1^2+aZ_1^4)(4X_1Y_1^2-X_3)-8Y_1^4, \tag{10}$$

$$Z_3 = 2Y_1Z_1. \tag{11}$$

formulas give the result of $[2]P = (X_3 : Y_3 : Z_3)$ *and* X_3*,* Y_3*,* Y_3 *can be computed using following formulas;*

$$\begin{aligned} &A \leftarrow Y_1^2 \\ &B \leftarrow 4X_1A \\ &C \leftarrow 8A^2 \\ &D \leftarrow 3X_1^2+aZ_1^4 \\ &X_3 \leftarrow D^2-2B \\ &Y_3 \leftarrow D(B-X_3)-C \\ &Z_3 \leftarrow 2Y_1Z_1 \end{aligned}$$

The cost of point doubling is $4M+6S$ *in Jacobian coordinates, where* M *means field multiplication and* S *means field squaring [10].*

Definition 18 (Point Addition using mixed Jacobian-affine coordinates)**.** *Let $P = (X_1 : Y_1 : Z_1) \in E$, where $Z_1, \neq 0$ and $Q = (X_2 : Y_2 : 1)$ where $P \neq \pm Q$. We know that $P = (X_1/Z_1^2 : X_1/Y_1^3 : 1)$. Hence we can use the point addition formula of E in affine coordinates to calculate $P + Q = (X_3^{'} : Y_3^{'} : 1)$.*

$$\begin{aligned} X_3^{'} &= \left(\frac{Y_2 - \frac{Y_1}{Z_1^3}}{X_2 - \frac{X_1}{Z_1^2}}\right)^2 - \frac{X_1}{Z_1^2} - X_2 \\ &= \left(\frac{Y_2 Z_1^3 - Y_1}{\left(X_2 Z_1^2 - X_1\right) Z_1}\right)^2 - \frac{X_1}{Z_1^2} - X_2 \\ Y_3^{'} &= \left(\frac{Y_2 - \frac{Y_1}{Z_1^3}}{X_2 - \frac{X_1}{Z_1^2}}\right)\left(\frac{X_1}{Z_1^2} - X_3^{'}\right) - \frac{Y_1}{Z_1^3} \\ &= \left(\frac{Y_2 Z_1^3 - Y_1}{\left(X_2 Z_1^2 - X_1\right) Z_1}\right)\left(\frac{X_1}{Z_1^2} - X_3^{'}\right) - \frac{Y_1}{Z_1^3} \end{aligned}$$

To eliminate the denominators, as in point doubling, we set $X_3 = X_3^{'} Z_3^2$ and $Y_3 = Y_3^{'} Z_3^3$ where $Z_3 = 2(X_2 Z_1^2 - X_1)Z_1$. As a result the following formulas gives us $P + Q = (X_3 : Y_3 : Z_3)$:

$$X_3 = (Y_2 Z_1^3 - Y_1)^2 - (X_2 Z_1^2 - X_1)^2 (X_1 + X_2 Z_1^2) \quad (12)$$

$$Y_3 = (Y_2 Z_1^3 - Y_1)(X_1 (X_2 Z_1^2 - X_1)^2 - X_3) - Y_1 (X_2 Z_1^2 - X_1)^3 \quad (13)$$

$$Z_3 = 2(X_2 Z_1^2 - X_1) Z_1. \quad (14)$$

X_3, Y_3, Y_3 can be computed using fallowing formulas;

$$
\begin{aligned}
A &\leftarrow Z_1^2\\
B &\leftarrow Z_1 A\\
C &\leftarrow X_2 A\\
D &\leftarrow Y_2 B\\
E &\leftarrow C - X_1\\
F &\leftarrow D - Y_1\\
G &\leftarrow E^2\\
H &\leftarrow GE\\
I &\leftarrow X_1 G\\
X_3 &\leftarrow F^2 - (H + 2I)\\
Y_3 &\leftarrow F(I - X_3) - Y_1 H\\
Z_3 &\leftarrow Z_1 E
\end{aligned}
$$

The point doubling costs $8M + 3S$ *[19].*

There is a special case where $a = -3$ for $y^2 = x^3 + ax + b$. We can select $a = -3$ without much loss of generality [19]. In this case point doubling costs $3M + 5S$ [8].

For more about projective and Jacobian coordinates, the reader may consult [23], [27], [8].

Chapter 3

Point Multiplication on Elliptic Curves

This section considers methods to calculate $[k]P$ over the field $\mathbb{F}_q$ where q is a prime number. $[k]P$ denotes adding P point to itself k times where k is an integer and P is a point on elliptic curve E over $\mathbb{F}_q$. This operation is called scalar multiplication or scalar multiplication. In these methods, the computation time of scalar multiplication, in other words cost of scalar multiplication is the important part for us. Hence we will give details about cots of algorithms during section 3.

There are two main subsections for scalar multiplication methods; base point P is unknown or fixed. Under section 3, we will examine point multiplication methods just for unknown point. In scalar multiplication for unknown point, point P and integer k are selected at the beginning of program. So we will not know values of both P and k before the run time.

3.1 Binary Method

This method is the simplest and oldest method to compute $[k]P$. It uses the binary representation of k,

$$k = \sum_{i=0}^{l-1} k_i 2^i \tag{1}$$

where $k_i \in 0, 1$ for every $i \in 0, \cdots, l-1$ and l is number of digits of binary k. Hence, $[k]P$ scalar multiplication can be computed by

$$\begin{aligned} [k]P &= \sum_{i=0}^{l-1} [k_i 2^i]P \\ &= k_0 P + k_1 2^1 P + k_2 2^2 P + \cdots k_{l-1} 2^{l-1} P \qquad (2) \\ &= 2(k_1 P + 2(k_2 P + \cdots + 2(k_{l-2} P + 2(k_{l-1} P)))) + k_0 P \qquad (3) \end{aligned}$$

This multiplication can be interpreted with two ways. These are called as **right-to-left binary method** and **left-to-right binary method**.

Equation (2) shows us the way that we fallow up for right-to-left binary method. It sums the terms $[k_i 2^i]P$ for each nonzero k_i from k_0 to k_{l-1} and gives $[k]P$ at the end as the result. k_i is known term so the next step is calculating $[2^i]P$. If we know $[2^{i-1}]P$, the previous term, it can be calculated by $2 \cdot 2^{i-1}P$. Fundamentally work of point multiplication is multiplying a point with a scalar and getting a new point on elliptic curve. So $2^{i-1}P$ is a point and $2 \cdot 2^{i-1}P$ is a point doubling. As a result, in equation (2), if k_i is nonzero then we are multiplying k_i with $2^i P$, which is calculated by doubling the previous one, and cumulatively adding to the sum for each $i \in 0, \cdots, l-1$. At this point using point doubling speeds up the algorithm. Algorithm 1 is the pseudo code of right-to-left binary method.

Algorithm 1 Right to Left Binary Method Scalar Multiplication [21]

```
Input: $k = (k_{l-1}, \dots, k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$
Output: $[k]P$
1: $Q \leftarrow \infty$
2: for $i = 0$ to $l - 1$ do
3:     if $k_i = 1$ then
4:         $Q \leftarrow P + Q$
5:     end if
6:     $P \leftarrow [2]P$
7: end for
8: return $Q$
```

Previous equation processes the bits of k from right to left, equation 3 processes the bits of k to opposite side. We will begin to calculate from k_{l-1} and count down to k_0. If k_i is nonzero we will add P and we will double the sum for every step independently of k_i. There is another difference between left-to-right and right-to-left methods except starting term. Unlike 1, we don't need to keep doubled version of P. The pseudo code of left-to-right binary method is given as Algorithm 2.

Algorithm 2 Left to Right Binary Method Scalar Multiplication [9]

```
Input: $k = (k_{l-1}, \dots, k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$
Output: $[k]P$
1: $Q \leftarrow \infty$
2: for $i$ from $t - 1$ downto $0$ do
3:     $Q \leftarrow 2Q$
4:     if $k_i = 1$ then
5:         $Q \leftarrow P + Q$
6:     end if
7: end for
8: return $Q$
```

There is a running time for every operation and if we count operations one by one we can determine the running time of algorithm. Addition and doubling are the operations that we used in Algorithm

1 and algorithm 2. Both algorithms have same operations so there will be an explanation for only Algorithm 1. Algorithm 1 doing point addition for each k_i if k_i is nonzero. Since k_i is bit of the binary representation of k, expected number of nonzero bits is half of length. Hence there are $l/2$ point addition. In the continuation of algorithm, there is a point doubling for every k_i which means l point doubling. Let A represents addition and D represents doubling, then the expected running time of Algorithm 1 is

$$\frac{l}{2}A + lD. \tag{4}$$

Here is the toy example to see steps for both of algorithms.

Example 10. *Let $k = 53 = (110101)$ and P is a point on elliptic curve. $53P$ can calculate as fallows with right-to-left binary method;*

Algorithm 1: Right-to-Left Binary Method

index i	k_i	Q	P
-	-	∞	P
0	1	$\infty + P = P$	$2P$
1	0	P	$4P$
2	1	$P + 4P = 5P$	$8P$
3	0	$5P$	$16P$
4	1	$5P + 16P = 21P$	$32P$
5	1	$21P + 32P = 53P$	$64P$

Here is the calculation of $53P$ with left-to-right binary method;

Algorithm 2: Left-to-Right Binary Method

index i	k_i	Q
-	-	∞
5	1	$2(\infty) + P = P$
4	1	$2(P) + P = 3P$
3	0	$2(3P) = 6P$
2	1	$2(6P) + P = 13P$
1	0	$2(13P) = 26P$
0	1	$2(26P) + P = 53P$

3.2 Non-adjacent From

Let $P = (x, y) \in E(\mathbb{F}_q)$. Then $-P = (x, -y)$ if $\mathbb{F}_q$ is a binary field and $\mathbb{F}_q$ has characteristic more than 3. Hence subtraction of points on an elliptic curve has almost same running time with addition. This brings up to use signed binary expansion of k to speed up.

Theorem 4 (Reitwiesner 1960)**.** *[32] Let $k \in \mathbb{Z}$ then there is exactly one signed binary expansion of k such that*

$$k = \sum_{i=0} k_i 2^i, k_i \in \{-1, 0, 1\}$$
$$k_i k_{i+1} = 0, \forall i \geq 0.$$

Actually, this theorem gives us definition of non-adjacent form (NAF) of a positive integer.

Definition 19. *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 k_i are nonzero. The length of the NAF is l.*

We can say that running time of algorithm increases in proportional to number of nonzero k_i in the binary methods. Hence less nonzero k_i means to speed up. In regular binary representations of k, expected number of nonzero digits is half of the length. But with NAF this, expected number of nonzero digits, drops to $1/3$ of all digits. Because of definition 19 each nonzero digit has to be adjacent to two zero digits.

Theorem 5 (properties of NAFs)**.** *[32] [28] Let k be a positive integer.*

(i) k has unique NAF denoted NAF(k).

(ii) NAF(k) has the fewest nonzero digits of any signed digit representation of k.

(iii) The length of NAF(k) is at most one more than the length of the binary representation of k.

(iv) If the length of NAF(k) is l, then $2^l/3 < k < 2^{l+1}/3$

(v) The average density of nonzero digits among all NAFs of length l is approximately $1/3$.

Algorithm 3 shows computing NAF(k) efficiently. Algorithm obtains digits of NAF(k) dividing k by 2 continuously and collecting remainders which are equal -1, 0 or 1.

Algorithm 3 Computing NAF [28] [32]

```
Input: $k$
Output: $NAF(k)$
1:  $i \leftarrow 0$
2:  while $k \geq 1$ do
3:      if $k \pmod 2 = 1$ then
4:          $k_i \leftarrow 2 - k \pmod 4$
5:          $k \leftarrow k - k_i$
6:      else if $k \pmod 2 = 0$ then
7:          $k_i \leftarrow 0$
8:      end if
9:      $k \leftarrow k/2$
10:     $i \leftarrow i + 1$
11: end while
12: return $(k_{i-1}, k_{i-2}, \dots, k_1, k_0)$
```

Here is an example to see how Algorithm 3 computes NAF(k).

Example 11. *Let $k = 53$. Algorithm 3 can calculate NAF(53) as fallows;*

index i	k_i	k
-	-	53
0	$2-(53 \bmod 4)=1$	$53-1=54, 54/2=27$
1	$2-(27 \bmod 4)=-1$	$27-(-1)=28, 28/2=14$
2	0	$14/2=7$
3	$2-(7 \bmod 4)=-1$	$7-(-1)=8, 8/2=4$
4	0	$4/2=2$
5	0	$2/2=1$
6	$2-(1 \bmod 4)=1$	$1-1=0, 0/2=0$

Algorithm 4 looks like similar with Algorithm 2. Actually both algorithms have some step with one difference; Algorithm 4 uses NAF(k) instead of the binary representation of k.

Algorithm 4 Binary NAF Method for Scalar Multiplication [19] [28]

```
Input: $k$, $P \in E(\mathbb{F}_q)$
Output: $[k]P$
 1: Use Algorithm 3 to compute $NAF(k) = (k_{l-1}, \ldots, k_1, k_0)$
 2: $Q \leftarrow \infty$
 3: for $i = l-1$ to $0$ do
 4:     $Q \leftarrow 2Q$
 5:     if $k_i = 1$ then
 6:         $Q \leftarrow P + Q$
 7:     else if $k_i = -1$ then
 8:         $Q \leftarrow (-P) + Q$
 9:     end if
10: end for
11: return $Q$
```

Expected running time of the algorithm can be calculated by counting additions and doublings of the algorithm. From (iii) and (v) parts of Theorem 5 there are $l/3$ addition for nonzero digits and l doublings for all digits. Therefore, the expected running time of Algorithm 4 is

$$\frac{l}{3}A + lD. \tag{5}$$

3.3 Window Method

Sİnce previous sections, we know that if we have less nonzero digits for representation fo k then algorithm will speed up. If there are some more available memory, we can speed up algorithm. In section 3.2 we used elements of $\{-1, 0, 1\}$ set to represent k and added or subtracted P. But Window method is using elements of larger set for representation of k. In this case, there will be additions or subtractions of some small scalar multiple of P.

Definition 20. *Let $w \geq 2$ be a positive integer. Then a positive integer k has exactly one width-w non-adjacent form*

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

where each nonzero coefficient k_i is odd, $|k_i| < 2^{w-1}$, $k_{l-1} \neq 0$, and at most one of any w consecutive digits is nonzero. The length of width-w NAF is l.

Theorem 6 (properties of width-w NAFs)**.** *[29] [19] Let k be a positive integer.*

(i) *k has a unique width-w NAF denoted NAF$_w$(k).*

(ii) *NAF$_2$(k) $=$ NAF(k).*

(iii) *The length of NAF$_w$(k) is at most one more than the length of the binary representation of k.*

(iv) *The average density of nonzero digits among all width-w NAFs of length l is approximately* $1/w + 1$.

NAF$_w$ (k) can be efficiently compute similar to NAF(k). Window method processes w digits of k at a time.

Algorithm 5 Computing w-NAF [19] [36]

```
Input: $w$, $k$
Output: $NAF_w(k)$
1: $i \leftarrow 0$
2: while $k \geq 1$ do
3:     if $k \pmod 2 = 1$ then
4:         $k_i \leftarrow k \pmod{2^w}$
5:         $k \leftarrow k - k_i$
6:     else if $k( \pmod 2) = 0$ then
7:         $k_i \leftarrow 0$
8:     end if
9:     $k \leftarrow k/2$
10:    $i \leftarrow i + 1$
11: end while
12: return $(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$
```

Due to Algorithm 5 the digits of NAF$_w$ (k) are obtained by repeatedly dividing k by 2 and remainders are in $[-2^{w-1}, 2^{w-1} - 1]$. If k is odd and remainder is $k_i = k \pmod{2^w}$ then $(k - k_i)/2$ will be divisible by 2^{w-1}. This ensures that there are $w - 1$ zero digits after a nonzero digit. Following example will also show this.

Example 12. *Let $k = 53$. We denote a negative integer $-a$ by $\overline{a}$. The binary representation of k is* (110101) *and width-w NAFs of k for $2 \leq w \leq 5$ are:*

w	*NAF*$_w(k)$
2	$(1\overline{1}0\overline{1}001)$
3	$(3003\overline{1})$
4	(30005)
5	$(100000\overline{1}\overline{1})$

We know that NAF$_2(k)$ = NAF(k) from Theorem 6 NAF$_w(k)$ scalar multiplication algorithm is general version of NAF(k) (Algorithm 4), only difference is NAF$_w(k)$ scalar multiplication has a precomputation step.

Algorithm 6 Window NAF Method for Scalar Multiplication [19] [36]

```
Input: $w$, $k$, $P \in E(\mathbb{F}_q)$
Output: $[k]P$
1:  Use Algorithm 5 to compute $NAF_w(k) = (k_{l-1}, \dots, k_1, k_0)$
2:  Compute $P_i = iP$ for $i = 1, 3, 5, \dots, 2^{w-1} - 1$
3:  $Q \leftarrow \infty$
4:  for $i \leftarrow l - 1$ to $0$ do
5:    $Q \leftarrow [2]Q$
6:    if $k_i \neq 0$ then
7:      if $k_i > 0$ then
8:        $Q \leftarrow Q + P_{k_i}$
9:      else if $k_i < 0$ then
10:       $Q \leftarrow Q - P_{-k_i}$
11:     end if
12:   end if
13: end for
14: return $Q$
```

From (iii) and (iv) parts of Theorem 6 expected running time of Algorithm 6 is approximately

$$\left[1D + (2^{w-2} - 1)A\right] + \left[\frac{l}{w+1}A + lD\right]. \tag{6}$$

There can be thought like if we increase w we can get more speed through (iv) of Theorem 6. But in Algorithm 6 there is a cost for precomputation step and this cost increasing, almost, exponentially. Hence there are optimum values for w. The following table shows costs of NAF$_w(k)$ scalar multiplication for some w values which are calculated using Formula 6 that expected running time of Algorithm 6.

w	Cost
3	$\frac{l+4}{4}A + (l+1)D$
4	$\frac{l+15}{5}A + (l+1)D$
5	$\frac{l+42}{6}A + (l+1)D$
6	$\frac{l+105}{7}A + (l+1)D$

The window NAF method uses sliding window which has similar to width-w window. But alternatively, sliding window can be used on binary or NAF representation of k. Sliding window method moves left-to-right and ignores consecutive zero digits after a nonzero digit k_i is progressed. The method works left-to-right over digits of NAF representation of k with a window, which has width at most w, therefore the value in the window is odd.

The average length of a run of zeros between windows in the sliding window method is

$$v(w) = \frac{4}{3} - \frac{(-1)^w}{3 \cdot 2^{w-2}}. \tag{7}$$

Hence the expected running time of Algorithm 7 is approximately

$$\left[1D + \left(\frac{2^w - (-1)^w}{3} - 1\right) A\right] + \frac{l}{wTv(w)} + lD. \tag{8}$$

Algorithm 7 Sliding Window Method for Scalar Multiplication [19] [2]

```
Input: $w$, $k$, $P \in E(\mathbb{F}_q)$
Output: $[k]P$
1: Use algorithm to compute $NAF(k) = (k_{l-1}, \dots, k_1, k_0)$
2: Compute $P_i = iP$ for $i = 1, 3, 5, \dots, 2(2^w - (-1)^w)/3 - 1$
3: $Q \leftarrow \infty$
4: $i \leftarrow l - 1$
5: while $i \geq 0$ do
6:     if $k_i = 0$ then
7:         $t \leftarrow 1$
8:         $u \leftarrow 0$
9:     else if $k_i \neq 0$ then
10:        Find the largest $t \leq w$ such that $u \leftarrow (k_i, \dots, k_{i-t+1})$ is odd.
11:    end if
12:    $q \leftarrow 2^t Q$
13:    if $u > 0$ then
14:        $Q \leftarrow Q + P_u$
15:    else if $u < 0$ then
16:        $Q \leftarrow Q - P_{-u}$
17:    end if
18:    $i \leftarrow i - t$
19: end while
20: return $Q$
```

Before finishing this section let's compare sliding window and Window NAF methods slightly. For a known w, the sliding window takes larger values in a window than width-w NAF. This returns as a higher cost in precomputation step that costs $2^w/3$ point operations for sliding window method (Algorithm 7) and $2^w/4$ point operations for window NAF method (Algorithm 6). If we continue to compare both algorithms over point operations the window NAF method will usually result in fewer point additions for the optimum w. For more certain comparison, we should take into consideration coordinate representations.

Chapter 4

Faster Point Multiplication Methods

Until this section, we gave point addition and point doubling on elliptic curves in affine coordinates. Then we transported these point operations formulas to Jacobian projective coordinates. After that we use these formulas to compute scalar multiplication which is the fundamental operation in elliptic curve cryptology. We talked about different scalar multiplication methods. In every new method our aim was getting effective and fast point multiplication algorithms. In this section we will look at faster and efficient scalar multiplication methods than we gave in Section 3 and we will link up with Section 2.5 for that.

4.1 GLV Method

This section briefly restates GLV method. GLV is Gallant, Lambert and Vanstone's method in speeds up scalar multiplication on elliptic curves using classes of elliptic curve which have efficiently computable endomorphisms. As an advantage GLV method allows to work on larger classes of elliptic curves. For this special class of curves, [18] says that a speedup of up to $\%50$ can be expected over the best general methods for point multiplication.

4.1.1 Endomorphisms

Let E be an elliptic curve defined over the finite field $\mathbb{F}_q$. Due to Definition 15 an endomorphism of E is a rational map $\phi\colon E \longrightarrow E$ that satisfies $\phi(\infty) = \infty$ where ∞ denotes the point at infinity. Since the rational map is defined over $\mathbb{F}_q$, the endomorphism ϕ is also defined over $\mathbb{F}_q$. Hence ϕ is a group homomorphism of $E(\mathbb{F}_q)$.

Let P be a point on this curve with order n such that $h = \#E(\mathbb{F}_q)/n$ co-factor is a small integer such that $h \leq 4$. Let ϕ be a non trivial and effectively computable endomorphism defined over $\mathbb{F}_q$ and $X^2 + rX + s$ is characteristic polynomial of ϕ. There is $\lambda \in [0, n-1]$ such hat $\phi(P) = \lambda P$ where λ is a root of characteristic polynomial $X^2 + rX + s$ modulo n.

Example 13 (Elliptic Curves with $j = 1728$)**.** *[25]*

Let $p \equiv 1 \pmod 4$ be a prime and consider the elliptic curve

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

defined over $\mathbb{F}_p$ with $p+1-t$ points and $\alpha \in \mathbb{F}_p$ is an element of order 4, that satisfies the characteristic equation $\alpha^2 + 1 \equiv 0$. Then there is an efficiently computable endomorphism defined by

$$\phi \;:\; E \longrightarrow E, \;\; (x, y) \longmapsto (-x, \alpha y).$$

Let $P \in E(\mathbb{F}_p)$ be a random point of prime order n such that $h = \#E/n$ is a "small" integer; typically $h \leq 4$. Then $\phi(Q) = \lambda Q$ for every $Q \in \langle P \rangle$ where $\lambda \in \mathbb{Z}$ satisfies $\lambda^2 + 1 \equiv 0 \pmod n$.

Example 14 (Elliptic Curves with $j = 0$)**.** *[20]*

Let $p \equiv 1 \pmod 3$ be a prime and the elliptic curve is

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

defined over $\mathbb{F}_p$ with $p+1-t$ points. Let $\beta \in \mathbb{F}_p$ be an element order of 3 that satisfies the characteristic equation $\beta^2 + \beta + 1 \equiv 0$. Then ϕ is an endomorphism defined by

$$\phi \;:\; E \longrightarrow E, \;\; (x, y) \longmapsto (\beta x, y).$$

Let $P \in E(\mathbb{F}_p)$ be a point with prime order n such that $h = \#E/n$ is a "small" integer; typically $h \leq 4$. Then $\phi(Q) = \lambda Q$ for every $Q \in \langle P \rangle$ where $\lambda \in \mathbb{Z}$ satisfy $\lambda^2 + \lambda + 1 \equiv 0 \pmod{n}$.

Example 15 (the elliptic curve P-160)**.** *[19]*

Consider the elliptic curve

$$E\colon y^2 = x^3 + 3$$

defined over the $160-$bit prime field $\mathbb{F}_p$, where

$$\begin{aligned} p &= 2^{160} - 229233 \\ &= 1461501637330902918203684832716283019655932313743. \end{aligned}$$

Since $p \equiv 1 \pmod 3$, the curve is of the type described in Example 14. The group of $\mathbb{F}_p$-rational points on E has prime order

$$\#E(\mathbb{F}_p) = n = 1461501637330902918203687013445034429194588307251.$$

An element of order 3 in $\mathbb{F}_p$ is

$$\beta = 771473166210819779552257112796337671037538143582$$

and so the map $\phi\colon E \longrightarrow E$ defined by $\phi\colon \infty \mapsto \infty$ and $\phi\colon (x, y) \mapsto (\beta x, y)$ is an endomorphism of E defined over $\mathbb{F}_p$. The solution

$$\lambda = 903860042511079968555273866340564498116022318806$$

to the equation $\lambda^2 + \lambda + 1 \equiv 0 \pmod{n}$ has the property that $\phi(P) = \lambda P$ for all $P \in E(\mathbb{F}_p)$.

This curve is included in the WAP specification of the Wireless Transport Layer Security (WTLS) protocol [39].

The way for computing kP for selected equally random k from $[0, n-1]$ that given by [18] is the following. Assume that $k = k_1 + k_2\lambda \pmod{n}$ where $k_1, k_2 \leq \lceil \sqrt{n} \rceil$. Hence

$$\begin{aligned} kP &= (k_1 + k_2\lambda)P \\ &= k_1P + k_2(\lambda P) \\ &= k_1P + k_2\phi(P). \end{aligned}$$

$\phi(P)$ can be computed easily. Since k_1 and k_2 are approximately half the length of k, applying Algorithm 6 to k_1P and k_2P will reduce points doublings to half. Sure we don't forget that this is effective provided when a decomposition and $\phi(P)$ can be computed efficiently.

4.1.2 Decomposition of the scalar

Let $k \in (0, n)$ be an integer constant. We want to write $k = k_1 + k_2\lambda \pmod n$ where $k_1, k_2 \leq \lceil\sqrt{n}\rceil$. Let define the group homomorphism

$$\begin{aligned} f\colon \mathbb{Z}\times\mathbb{Z} &\longrightarrow \mathbb{Z}/n \\ (i,j) &\longmapsto i + \lambda j \pmod n. \end{aligned}$$

We can say that $(k, 0)$ is a vector that satisfies $f(k, 0) = k$. We want to find $(k_1, k_2) = u$ vector such that $f(u) = k_1 + \lambda k_2 = k$. Note that it is easy to find a vector $u \in \mathbb{Z}\times\mathbb{Z}$ such that $f(u) = k$. $u = (k, 0)$ is this kind of vector. But the problem is finding a vector that is short at the same time. In that point, we should pass some steps to find a proper u. First we should find linearly independent short vectors $v_1, v_2 \in \mathbb{Z}\times\mathbb{Z}$ such that $f(v_1) = 0$ and $f(v_2) = 0$. After this, we find a vector v in the integer lattice generated by v_1 and v_2 that close to $(k, 0)$. Then $u = (k, 0) - v$ with $f(u) = f((k, 0)) - f(v)$.

Note that using lattice basis reduction algorithms can be solve our problems about finding linearly independent v_1 and v_2 vectors, and finding v vector generated by v_1 and v_2 that close to $(k, 0)$. However, methods are faster which are presented in [18].

Let $\mathscr{K} = \ker f$ and v_1, v_2 be linearly independent vectors of $\mathscr{K}$ satisfy $f(v_1) = 0$ and $f(v_2) = 0$. We can write that,

$$(k, 0) = \beta_1 v_1 + \beta_2 v_2$$

where $\beta_i \in \mathbb{Q}$ because v_1, v_2 are linearly independent vectors.Then $b_i = \lceil \beta_i \rfloor$ and let $v = b_1 v_1 + b_2 v_2$.

We can compute v_1, v_2 linearly independent vectors using Extended Euclidean Algorithm. If we apply Extended Euclidean Algorithm to find greatest common divisor of n and λ, we get

$$s_i n + t_i \lambda = r_i \tag{1}$$

where $s_0 = 1$, $t_0 = 0$, $r_0 = n$, $s_1 = 0$, $t_1 = 1$, $r_1 = \lambda$, $r_i \geq 0$ for all i. In a specific step of this equation algorithm gives us v_1, v_2 vectors.

Lemma 1. *Let s_i, t_i, r_i be the sequence of variables in Equation 1 produced by an application of extended Euclidean algorithm to positive integers n and λ.*

(i) $r_i > r_{i+1} \geq 0$ for all $\S \geq 0$.

(ii) $|s_i| < |s_{i+1}|$ for $i \geq 1$.

(iii) $|t_i| < |t_{i+1}|$ for $i \geq 0$.

(iv) $r_{i-1}|t_i| + r_i|t_{i-1}| = n$ for all $i \geq 1$.

Let m be the greatest index for which $r_m \geq \sqrt{n}$. Due to Lemma 1 (iv), we know that $r_m|t_{m+1}| + r_{m+1}|t_m| = n$. So $|t_{m+1}| < \sqrt{n}$. Then we choose $v_1 = (r_{m+1}, -t_{m+1})$. By Equation 1, $f(v_1) = (r_{m+1} - t_{m+1}\lambda) \pmod{n} = 0$. Since $|t_{m+1}| < \sqrt{n}$ and $|r_{m+1}| < \sqrt{n}$, we have $\|v_1\| = \sqrt{r_{m+1}^2 + t_{m+1}^2} < \sqrt{2n}$ which means v_1 is short. Then we also choose v_2 to be shorter of $(r_m, -t_m)$ and $(r_{m+2}, -t_{m+2})$. With similar way that we trace with v_1, $f(v_2) = 0$ and v_2 is also short.

We found two short vectors v_1 and v_2 but there is one more condition that they should be linearly independent. Assume that they are linearly dependent. Let $v_2 = (r_m, -t_m)$ then

$$\frac{r_{m+1}}{r_m} = \frac{-t_{m+1}}{-t_m} = \frac{t_{m+1}}{t_m}. \tag{2}$$

But, by Lemma 1 (i) and (iii), $r_{m+1}/r_m < 1$ and $|t_{m+1}/t_m| > 1$. Therefore, v_1 and v_2 are linearly independent.

Algorithm 8 Extended Euclidean Algorithm for Integers [19]

```
Input: Positive integers $a$ and $b$ with $a \leq b$.
Output: $d = gcd(a,b)$ and integers $x, y$ satisfying $ax + by = d$.
1:  $u \leftarrow a$, $v \leftarrow b$
2:  $x_1 \leftarrow 1$, $x_2 \leftarrow 0$
3:  $y_1 \leftarrow 0$, $y_2 \leftarrow 1$
4:  while $u \neq 0$ do
5:      $q \leftarrow \lfloor v/u \rfloor$, $r \leftarrow v - qu$
6:      $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$
7:      $v \leftarrow u$, $u \leftarrow r$
8:      $x_2 \leftarrow x_1$, $x_1 \leftarrow x$
9:      $y_2 \leftarrow y_1$, $y_1 \leftarrow y$
10: end while
11: $d \leftarrow v$, $x \leftarrow x_2$, $y \leftarrow y_2$
12: return $(d, x, y)$
```

Linearly independent vectors v_1 and v_2 can be obtained by using Algorithm 8. If we put inputs as n and λ, the algorithm produces a sequence of equations $s_i n + i\lambda = r_i$ where $s_0 = 1$, $t_0 = 0$, $r_0 = n$, $s_1 = 0$, $t_1 = 1$, $r_1 = \lambda$.

Our aim was finding $v = b_1 v_1 + b_2 v_2$ that close to $(k, 0)$ where $b_i = \lceil \beta_i \rfloor$. The last step is finding β_1 and β_2. $(k, 0) = \beta_1 v_1 + \beta_2 v_2$ equation provides these values. We already calculated $v_1 = (v_{1x}, v_{1y})$ and $v_2 = (v_{2x}, v_{2y})$ vectors. If we put them in to equation

$$\begin{aligned}
(k,0) &= \beta_1 v_1 + \beta_2 v_2 \\
&= \beta_1(v_{1x}, v_{1y}) + \beta_2(v_{2x}, v_{2y}) \\
&= (\beta_1 v_{1x}, \beta_1 v_{1y}) + (\beta_2 v_{2x}, \beta_2 v_{2y}) \\
&= (\beta_1 v_{1x} + \beta_2 v_{2x}, \beta_1 v_{1y} + \beta_2 v_{2y}).
\end{aligned}$$

At the end, these steps gives two equation,

$$\begin{aligned} k &= \beta_1 v_{1x} + \beta_2 v_{2x}, \\ 0 &= \beta_1 v_{1y} + \beta_2 v_{2y}. \end{aligned}$$

Hence,

$$\beta_1 = \frac{k v_{2y}}{v_{1x} v_{2y} - v_{1y} v_{2x}}, \beta_2 = \frac{-k v_{1y}}{v_{1x} v_{2y} - v_{1y} v_{2x}}.$$

Since $b_i = \lceil \beta_i \rfloor$ and we have $v = b_1 v_1 + b_2 v_2$, $u = (k, 0) - v$ is the short vector that we need. The fallowing lemma proves that the vector u is indeed short.

Lemma 2. *[18] The vector $u = (k, 0) - v$, where v is constructed as above, has norm at most max$(\|v_1\|, \|v_2\|)$.*

Proof. We have

$$\begin{aligned} u &= (k, 0) - v \\ &= (\beta_1 v_1 + \beta_2 v_2) - (b_1 v_1 + b_2 v_2) \\ &= (\beta_1 - b_1) v_1 + (\beta_2 - b_2) v_2. \end{aligned}$$

Finally, since $|\beta_1 - b_1| \leq \frac{1}{2}$ and $|\beta_2 - b_2| \leq \frac{1}{2}$, by the Triangle Inequality we have

$$\begin{aligned} \|u\| &\leq \frac{1}{2}\|v_1\| + \frac{1}{2}\|v_2\| \\ &\leq max(\|v_1\|, \|v_2\|). \end{aligned}$$

□

Following algorithm calculates k_1 and k_2 and uses Extended Euclidean Algorithm (Algorithm 8) to obtain needed components. It is the summary of the method for decomposing k that we explained above.

Algorithm 9 Balanced length-two representation of a multiplier [19]

Input: Integers n, λ, $k \in [0, n-1]$.

Output: Integers k_1, k_2 such that $k = (k_1 + k_2\lambda) \pmod{n}$ and $|k_1|, |k_2| \approx \sqrt{n}$.

```
1:  Run the Algorithm 8 with inputs $n$ and $\lambda$. The algorithm produces a sequence of equations $s_i n + i\lambda = r_i$ where $s_0 = 1$, $t_0 = 0$, $r_0 = n$, $s_1 = 0$, $t_1 = 1$, $r_1 = \lambda$, and the remainders $r_i$ and are non-negative and strictly decreasing. Let $m$ be the greatest index for which $r_m \geq \sqrt{n}$.
2:  $(x_1, y_1) \leftarrow (r_{m+1}, -t_{m+1})$
3:  if $(r_m^2 + t_m^2) \leq (r_{m+2}^2 + t_{m+2}^2)$ then
4:      $(x_2, y_2) \leftarrow (r_1, -t_1)$
5:  else
6:      $(x_2, y_2) \leftarrow (r_{m+2}, -t_{m+2})$
7:  end if
8:  $b_1 \leftarrow \lfloor \beta_1 \rceil$
9:  $b_2 \leftarrow \lfloor \beta_2 \rceil$
10: $k_1 \leftarrow k - b_1 x_1 - b_2 y_2$
11: $k_2 \leftarrow -b_1 y_1 - b_2 y_2$
12: return $(k_1, k_2)$
```

Next example shows applying of Algorithm 9 to Example 15.

Example 16 (balanced lenght-two representation of a multiplier k)**.** *Consider the elliptic curve P*-160 *defined in Example 15. Since we apply Algorithm 9, we have*

$$\begin{aligned}
(r_m, t_m) &= (2180728751409538655993509, -1860295391676851993533061) \\
(r_{m+1}, t_{m+1}) &= (788919430192407951782190, 602889891024722752429129) \\
(r_{m+2}, t_{m+2}) &= (602889891024722752429129, -1391809321217130704211319) \\
(x_1, y_1) &= (788919430192407951782190, -602889891024722752429129) \\
(x_2, y_2) &= (602889891024722752429129, 1391809321217130704211319).
\end{aligned}$$

Let

$$k = 96548628832721855909790906972427557936000839825 7.$$

We obtain

$$\begin{aligned} b_1 &= 919446671339517233512759 \\ b_2 &= 398276613783683332374156 \end{aligned}$$

and

$$\begin{aligned} k_1 &= -980937239718038467540 77 \\ k_2 &= 381880690058693066485147. \end{aligned}$$

Algorithm 10 Scalar Multiplication with Efficiently Computable Endomorphisms [19]

```
Input: Integer $k \in [1, n-1]$, $P \in E(\mathbb{F}_q)$ window width $w_1$ and $w_2$, and $\lambda$
Output: $kP$
1: Use Algorithm 9 to find $k_1$ and $k_2$ such that $k = (k_1 + k_2\lambda) \pmod{n}$.
2: $P_1 \leftarrow P$
3: $P_2 \leftarrow \phi(P)$
4: Use Algorithm 5 to compute NAF$_{w^j}(|k_j|) = \sum_{i=0}^{m_j-1} k_{j,i}2^i$ for all $j = 1, 2$.
5: $m = \max\{m_1, m_2\}$
6: $k_{j,i} \leftarrow 0$ for $m_j \le i < m$, $1 \le j \le 2$
7: if $k_j < 0$ then
8:     $k_{j,i} \leftarrow -k_{j,i}$
9: end if
10: Compute $iP_j$ for $i \in \{1, 3, \cdots, 2^{w_j-1} - 1, 1 \le j \le 2$.
11: $Q \leftarrow \infty$
12: for $j = 1$ to $2$ do
13:     if $k_{j,i} \neq 0$ then
14:         $Q \leftarrow Q + k_{j,i}P_j$
15:     else
16:         $Q \leftarrow Q - |k_{j,i}|P_j$
17:     end if
18: end for
19: return $Q$
```

Algorithm 10 calculates kP using the decomposition $k = (k_1 + k_2\lambda) \pmod{n}$ and interleaving $k_1P + k_2\phi(P)$ where ϕ is a efficient endomorphism of elliptic curve E over a finite field $\mathbb{F}_q$. The expected running time of algorithm is approximately

$$\left[|j|D + \sum_{j=1}^{2}(2^{w_j-2} - 1)A + C_k + C_\phi\right] + \frac{t}{2}\left[D + \sum_{j=1}^{2}\frac{1}{w_j + 1}A\right] \tag{3}$$

where t is bitlength of n, $j\colon w_j > 2$, k_j is written with a NAF$-w_j$, C_k and C_ϕ denotes respectively the costs of the decomposition of k and finding $\phi(P)$. Lastly, the storage requirement is $2^{w_1-2} + 2^{w_2-2}$ points.

4.2 GLS Endomorphism

Section 4.1 gave the Gallant-Lambert-Vanstone (GLV) [18] method that accelerates elliptic curve scalar multiplication with efficiently computable endomorphisms. Since it requires less point operations than previous methods GLV method is important tool for faster scalar multiplication. However, there were efficiently computable endomorphism examples in only two cases like curves with special endomorphism class or groups $E(\mathbb{F}_{q^m})$ defined over $\mathbb{F}_q$ and if q isn't very small, these groups don't have prime or nearly prime order. Especially, we gave curves with $j-$invariants 0 and 1728 for GLV method. If you want to work with primes (e.g. $p = 2^{255} - 19$), only special curves can be used with the GLV. Because if there are small sets of applicable curves and fields, it can be hard to get a good group order. Hence, for randomly chosen prime order elliptic curves or for special primes GLV method isn't applicable every time. [17] gives endomorphisms for a large class of elliptic curves by working over $\mathbb{F}_{q^2}$ that can be applied to GLV method to solve this problem [15] [16].

Let q be a prime number and $\mathbb{F}_q$ is a finite field. If we define an elliptic curve over an extension field of $\mathbb{F}_q$, [17] guaranties the construction of an efficiently computable endomorphism. This endomorphism is then used to speed up scalar multiplication.

Let E be an elliptic curve over $\mathbb{F}_q$ with $q + 1 - t$ points. In that case we can also calculate number of points $\#E(\mathbb{F}_{q^m})$ and we can define $E(\mathbb{F}_{q^m})[r] = \{P \in E(\mathbb{F}_{q^m} : [r]P = \infty)\}$. Let $d = 1$ and $\phi^{'}$ be ϕ^{-1}. Then we can replace 'separable isogeny' with 'isomorphism'. If r is a prime, $r \mid\mid N$ denotes that $r \mid N$ but $r^2 \nmid N$.

Theorem 7. *Let E be an elliptic curve defined over $\mathbb{F}_q$ such that $\#E(\mathbb{F}_q) = q + 1 - t$ and let $\phi\colon E \to E^{'}$ be a separable isogeny of degree d defined over $\mathbb{F}_{q^k}$ where $E^{'}$ is an elliptic curve defined over $\mathbb{F}_{q^m}$ with $m \mid k$. Let $r \mid \#E^{'}(\mathbb{F}_{q^m})$ be a prime such that $r > d$ and such that $r \mid\mid \#E^{'}(\mathbb{F}_{q^k})$ which means $r \mid \#E^{'}(\mathbb{F}_{q^k})$ but $r^1 \nmid \#E^{'}(\mathbb{F}_{q^k})$. Let π be the q-power Frobenius map on E and let*

$\phi^{'} : E^{'} \to E$ be the dual isogeny of the ϕ. Define

$$\psi = \phi\pi\phi^{'}.$$

Then

(i) *$\psi \in End_{\mathbb{F}_{q^k}}(E^{'})$.*

(ii) *For all $P \in E^{'}(\mathbb{F}_{q^k})$ we have $\psi^k(P) - [d^k]P = \infty$ and $\psi^2(P) - [dt]\psi(P) + [d^2q]P = \infty$.*

(iii) *There is a unique $\lambda \in \mathbb{Z}/r\mathbb{Z}$ such that $\lambda^k - d^k \equiv 0 \pmod r$ such that $\psi(P) = [\lambda]P$ for all $P \in E^{'}(\mathbb{F}_{q^m})[r]$.*

Consider that if ϕ is an isomorphism then $E^{'} \cong E$ implies $End(E^{'}) \cong End(E)$ which means $End(E^{'})$ contains a corresponding endomorphism because $End(E)$ contains the $p-$power Frobenius map [17]. Theorem 7 forms the main construction. But in this section we will work over $\mathbb{F}_{q^2}$ with more specific endomorphisms. If we specialise Theorem 7, the proof of the specialised theorem will shows that it can applies to any elliptic curve over $\mathbb{F}_q$ and it also applies GLV method.

Theorem 8. *Let $p > 3$ be a prime and let E be an elliptic curve over $\mathbb{F}_q$ with $q + 1 - t$ points. Let $E^{'}$ over $\mathbb{F}_{q^2}$ be the quadratic twist of $E(\mathbb{F}_{q^2})$. Then $\#E^{'}(\mathbb{F}_{q^2}) = (q-1)^2 + t^2$. Let $\psi\colon E \to E^{'}$ be twisting isomorphism defined over $\mathbb{F}_{q^4}$. Let $r | \#E^{'}(\mathbb{F}_{q^2})$ be a prime such that $r > 2p$. Let $\psi = \phi\pi\phi^{'}$. For $P \in E^{'}(\mathbb{F}_{q^2})[r]$ we have $\psi^2(P) + P = \infty$.*

Proof. Let $E\colon y^2 = x^3 + ax + b$ with $a, b \in \mathbb{F}_q$. We know that $\#E^{'}(\mathbb{F}_{q^2}) = (q-1)^2 + t^2$. Let $u \in \mathbb{F}_{q^2}$ be a nonsquare in $\mathbb{F}_{q^2}$. Therefore $u^{(p^2-1)/2} = -1$. Define $A = u^2a$, $B = u^3b$ and $E^{'} : y^2 = x^3 + Ax + B$. Then $E^{'}$ is the quadratic twist of $E(\mathbb{F}_{q^2})$ and

$$\begin{aligned} \#E^{'}(\mathbb{F}_{q^2}) &= q^2 + 1 + (t^2 - 2q) \\ &= (q-1)^2 + t^2. \end{aligned}$$

The isomorphism $\psi\colon E \to E^{'}$ is given by

$$\phi(x, y) = (ux, \sqrt{u}^3 y)$$

and is defined over $\mathbb{F}_{q^4}$.

If $r \mid \#E^{'}(\mathbb{F}_{q^2})$ is prime such that $r > 2q$ then $r \nmid \#E(\mathbb{F}_{q^2}) = (q+1-t)(q+1+t)$ and so $r \mid\mid \#E^{'}(\mathbb{F}_q^4) = \#E(\mathbb{F}_{q^2})\#E^{'}(\mathbb{F}_{q^2})$. Hence we may apply Theorem 7. This shows that $\psi = \phi\pi\phi^{'}$ is a group homomorphism such that $\psi(P) = [\lambda]P$ for $P \in E^{'}(\mathbb{F}_{q^2})[r]$ where $\lambda^4 - 1 \equiv 0 \pmod r$. We show that, in fact, $\lambda^2 + 1 \equiv 0 \pmod r$,

By definition, $\psi(x,y) = (ux^q/u^q, \sqrt{u}^3 y^q/\sqrt{u}^{3q}/\sqrt{u}^{3q})$ where $u \in \mathbb{F}_{q^2}$ (i.e. $u^{q^2} = u$) and $\sqrt{u} \ni \mathbb{F}_{q^2}$ (and so $\sqrt{u}^{q^2} = -\sqrt{u}$). If $P = (x,y) \in E^{'}(\mathbb{F}_{q^2})$ then $x^{q^2} = x$, $y^{q^2} = y$ and so

$$\begin{aligned} \psi^2 &= (ux^{q^2}/u^{q^2}, \sqrt{u}^3 y^{q^2}/\sqrt{u}^{3q^2}) \\ &= (x, (-1)^3 y) \\ &= -(x,y). \end{aligned}$$

□

Now we can we can apply the result that we got from proof of Theorem 8 to elliptic curve E over $\mathbb{F}_q$ where $q > 3$. GLV method can also be applied. Since $E^{'}$ is defined over $\mathbb{F}_{q^2}$, $\#E^{'}(\mathbb{F}_{q^2})$ can be prime. Reader should check [30].

Let $q \equiv 5 \pmod 8$ be a prime and $E\colon y^2 = x^3 + ax + b$ be an elliptic curve over $\mathbb{F}_q$ with $q+1-t$ points, where t is trace of Frobenius which can be computed efficiently for all elliptic curves. Let u be a non-square element in $\mathbb{F}_{q^2}$. Let $E^{'} : y^2 = x^3 + Ax + B$ be the quadratic twist of $E(\mathbb{F}_{q^2})$ over $\mathbb{F}_{q^2}$ such that $A = u^2 a$ and $B = u^3 b$. Then there is a map defined by

$$\begin{aligned} \psi\colon E^{'} &\longrightarrow E^{'} \\ (x,y) &\longmapsto (x^q, iy^q) \end{aligned}$$

where $i \in \mathbb{F}_{q^2}$ satisfies $i^2 = -1$.

Let $r \mid \#E(\mathbb{F}_{q^2})$ be a prime such that $h = \#E'(\mathbb{F}_{q^2})/r$ is a small number and $P \in E^{'}(\mathbb{F}_{q^2})$ be a point order of r. Then $\phi(P) = \lambda P$ for some $\lambda \in \mathbb{Z}$ satisfy $\lambda^2 + 1 \equiv 0 \pmod r$.

Lemma 3. *Let notation be as previous theorem. The vectors* $(t, p-1)$, $(1-p), t$ *are orthogonal basis for a sub lattice* $L^{'}$ *of* L *of determinant* $\#E^{'}(\mathbb{F}_{q^2})$. *Given a point* $(a,b) \in \mathbb{R}^2$ *there exists a lattice point* $(x,y) \in L^{'}$ *such that* $||(a,b) - (x.y)|| \leq (p+1)/\sqrt{2}$.

Because of Lemma 3, we do not need to calculate linearly independent v_1, v_2 vectors using Extended Euclidean Algorithm (Algorithm 8). For GLV method, $(t, q-1)$ and $(q-1, -t)$ linearly independent vectors can be used as v_1, v_2. If we want to compute v_1, v_2 vectors with Algorithm 8, it will return same results with $(t, q-1)$ and $(q-1, -t)$. Therefore, computing coefficients k_1 and k_2 where $|k_1|, |k_2| \leq (q+1)/\sqrt{2}$ is easy in this case.

Following algorithm is a summary of this section. It generates a prime q ($q > 3$) and the quadratic twist of $E(\mathbb{F}_{q^2})$, calculates λ, and define efficient endomorphism ψ to use in GLV method for more speeding up.

Algorithm 11 Key Generation for Quadratic Twist Construction [17]

```
Output: $q$, $E^{'}$, $\psi$, $\lambda$
1:  Choose a prime $q = 5 \pmod 8$
2:  Set $u = \sqrt{2} \in \mathbb{F}_{q^2}$
3:  Set $A = -3$ and $a = A/2 \in \mathbb{F}_q$
4:  repeat
5:      Choose random $b \in \mathbb{F}_q$ and let $E\colon y^2 = x^3 + ax + b$
6:      Compute $t = p + 1 - \#E(\mathbb{F}_q)$
7:  until $(q-1)^2 + t^2 = hr$ where $r$ is prime and $h = 1$ or $h = 4$
8:  Set $B = bu^3 \in \mathbb{F}_{q^2}$ and $E^{'} : y^2 = x^3 + Ax + B$
9:  Set $\lambda = t^{-1}(q-1) \pmod r$
10: Compute $i \in \mathbb{F}_q$ so that $i^2 = -1$
11: Define $\psi(x, y) = (-x^q, iy^q)$
12: return $q$ $(A, B)$, $\psi$, $\lambda$
```

We used Algorithm 11 to define very simple map ψ and to generate an elliptic curve $E^{'} : y^2 = x^3 + Ax + B$ over $\mathbb{F}_{q^2}$, where $q > 3$ is a prime which has coefficient $A = -3$. Using coefficient $A = -3$ makes the curve convenient for efficient implementation when we use Jacobian coordinates (Section 2.6). We don’t forget that the algorithm can be used in more general cases.

Chapter 5

Implementation, Evaluation and Comparison

In previous sections, we gave various scalar multiplication algorithms and we always aimed to reach the fastest scalar multiplication algorithm ever known. At the end Gallant, Lambert and Vanstone's (GLV) method gave the speed up for scalar multiplication. But this method gave us narrow space to work. At that moment, Galbraith, Lin and Scott (GLS) endomorphisms gave larger classes.

In this section we will examine the magma implementation of GLV method which fortified with GLS endomorphisms. During the section we will use 256-bit primes for examples, algorithm implementations and comparisons.

```
p := NextPrime(Random([4..2^256]));
repeat
 p := NextPrime(p);
 F := GF(p);
until (p mod 8) eq 5;
_<x> := PolynomialRing(F);
K<u> := ext<F|x^2-2>;
```

Code 5.1: MAGMA/smul–ec–we–GLS

Let p be a 256-bit prime number and let $\mathbb{F}$ is a prime field with p. we need to find a p such that $p \equiv 5 \pmod 8$. This is one of the most important steps of algorithm. This line allows to find at least one root for i^2+1 polynomial in $\mathbb{F}$ and at the same time there isn't any x value that makes zero x^2-2

polynomial in $\mathbb{F}$.

After we found the proper p prime and built $\mathbb{F}$ as $\mathbb{F}_p$ prime field, the next step is extending $\mathbb{F}$ to $\mathscr{K}$ as $\mathbb{F}_{p^2}$ by using $x^2 - 2$ polynomial. Creating extension field $\mathscr{K}$ using $x^2 - 2$ will make the algorithm faster when we compare other extension fields. Now it's time to create our elliptic curve over $\mathbb{F}$.

```
repeat
  a := Random(F);
  b := Random(F);
  E1 := EllipticCurve([a, b]);
  n1 := #E1;
  E2 := BaseExtend(E1, K);
  t := p+1-n1;
  ad := K!(u^2)*a;
  bd := K!(u^3)*b;
  E2d := EllipticCurve([ad, bd]);
  assert IsQuadraticTwist(E2,E2d);
  n2d := (p-1)^2 + t^2;
  assert #E2d eq n2d;
  S := Factorization(n2d);
  r := S[#S][1];
  h := n2d div r;
until h le 4;
```

Code 5.2: MAGMA/smul–ec–we–GLS

Let a and b be random scalars in $\mathbb{F}$. We can built elliptic curve $E1$ using coefficients a and b over $\mathbb{F}$. $n1$ is number of points of $E1$ over $\mathbb{F}$ which is represented as $\#E1(\mathbb{F})$. Now we have elliptic curve $E1$ and extension field $\mathscr{K}$. We can extent $E1$ while we go from $\mathbb{F}$ to $\mathscr{K}$.

Let $E2$ be extended of $E1$ over $\mathscr{K}$. t is the trace of Frobenius. Since number of points can calculate with $p - t + 1$ for $E1$ than t satisfies $n1 = p - t + 1$ and we can calculate t with using $t = p - t + 1$ equation. GLS uses curves which are quadretic twist of extended curve over extension field. Now we need to create an elliptic curve using directions from [17] that will give quadratic twist of $E2$. This quadratic twist will be over extension field $\mathscr{K}$. Hence coefficients ad and bd are elements of $\mathscr{K}$ such that $ad = u^2a$ and $bd = u^3b$. Since we have have coefficients we can create $E2d$ over $\mathscr{K}$ which is the quadratic twist of $E2$.

Using points that have prime order is important for security. But it's hard to find a point with prime order randomly. Therefore will find a point that has prime order with the help of a random point. Factors of number of an elliptic curve includes prime order. First we need to calculate number of points of $E2d$ and it can compute with $n2d = (p-1)^2 + t^2$. Because of previous steps we know t, so we can also find $n2d$ using this equation. If we factorize $n2d$, the biggest factor is the prime order r. But taking the biggest r is not enough in this case, we also need have $h \leq 4$ where $n2d = h.r$ [17].

```
R := Random(E2d);
P := h*R;
assert Order(P) eq r;
e := E2d![0,1,0];
assert r*P eq e;
```

Code 5.3: MAGMA/smul–ec–we–GLS

Let R be a random point over $E2d$. If we multiply R with h, the result will be a point P over $E2d$ with prime order. If we check order of P, the result will be r and if multiply P with r, the result will be identity of $E2d$.

```
l := (Modinv(t,r)*(p-1)) mod r;
i := Sqrt(F!-1);
T := E2d![-P[1]^p, i*(P[2]^p)];

//Select the correct square root of -1.
assert T eq l*P or T eq -l*P;
if T ne l*P then
 assert T eq -l*P;
 i:=-i;
end if;
```

Code 5.4: MAGMA/smul–ec–we–GLS

ϕ represents GLS endomorphisms that is $\phi(Q) = [l]Q$ where l is root of $l^2 + 1 \equiv \pmod r$. Since we choose a prime where $p \equiv \pmod 8$ we know there is at least one l and also two i values which are the roots of characteristic equation $i^2 + 1 \equiv \pmod r$. i values are negative of each other. Hence we need a small test to be sure we take proper i. We can learn it easily using contents of ψ map.

```
Psifunc := function(P, E2d, p, i)
 return E2d![-P[1]^p,i*(P[2]^p)];
end function;
```

Code 5.5: MAGMA/smul–ec–we–GLS

ψ is the map that does the main work [17], defined as $\psi(x, y) = (-x^p, iy^p)$. Before the end, the last step is decomposing scalar k.

```
k := Random([0..r]);
B := (p-1)^2 + t^2 ;
b1 := (k*(t))/B;
b2 := (k*(p-1))/B;
v1 := Round(b1)*t + Round(b2)*(p-1);
v2 := Round(b1)*(p-1) - Round(b2)*t;
k1 := k-v1;
k2 := 0-v2;

[p^2, r, k, k1, k2];

assert k*P eq (k1*P+k2*l*P);
Q := Psifunc(P, E2d, p, i);
assert k*P eq (k1*P+k2*Q);
assert l*P eq Q;
```

Code 5.6: MAGMA/smul–ec–we–GLS

Let k be a random scalar in $[0, r]$ and we can decompose it to $k1$ and $k2$ scalars that satisfy $(k1, k2) = (k, 0) - (v1, v2)$ where $v1$ and $v2$ linearly independent vectors. Using $[k1]P + [k2]\psi[P]$ will be faster way to calculate $[k]P$.

In this algorithm we find $k1$, $k2$, $\psi[P]$ but at the end we have calculated $[k1]P + [k2]\psi[P]$ to MAGMA. Fallowing algorithm also calculates $[k1]P + [k2]\psi[P]$ beside $k1$, $k2$, $\psi[P]$.

```
dec := function(k, p, E)
 n:= #E;
 t := p+1-n;
```

```
  B := (p-1)^2 + t^2 ;
  b1 := (k*(t))/B;
  b2 := (k*(p-1))/B;
  v1 := Round(b1)*t + Round(b2)*(p-1);
  v2 := Round(b1)*(p-1) - Round(b2)*t;
  k1 := k-v1;
  k2 := 0-v2;
  return k1, k2;
end function;

phi := function(P, E, p, j)
  return E![-P[1]^p, j*(P[2]^p)];
end function;

wNaf := function(w, k)
  i := 1;
  t := [];
  while (k ge 1) do
    if (k mod 2) eq 1 then
      t[i] := k mod (2^w);
      k := k - t[i];
    elif (k mod 2) eq 0 then
      t[i] := 0;
    end if;
    k := k div 2;
    i := i + 1;
  end while;
  return t;
end function;

mul := function(w, k, P, E, p, j)
  t1 := [];
  t2 := [];
  s1 := [];
  s2 := [];
  k1, k2 := dec(k, p, E);
  Q := phi(P, E, p, j);
  t1 := Reverse(wNaf(w, k1));
  m1 := #t1;
  t2 := Reverse(wNaf(w, k2));
  m2 := #t2;
  m:=Maximum(m1, m2);
  if m1 eq m then
    for i:=(m2+1) to m do
      t2[i]:=0;
    end for;
  elif m2 eq m then
    for i:=(m1+1) to m do
      t1[i]:=0;
    end for;
  end if;
  if k1 lt 0 then
    for i:=1 to m1 do
      t1[i]:=-t1[i];
    end for;
  end if;
  if k2 lt 0 then
    for i:=1 to m2 do
      t2[i]:=-t2[i];
    end for;
  end if;
  for i:=1 to (2^w-1) by 2 do
    s1[i] := i*P;
    s2[i] := i*Q;
  end for;
  R := E![0,1,0];
  for i:=m to 1 by -1 do
    R := 2*R;
    if t1[i] ne 0 then
      if t1[i] gt 0 then
        R := R + s1[t1[i]];
      else
        R := R - s1[Abs(t1[i])];
      end if;
    end if;
    if t2[i] ne 0 then
      if t2[i] gt 0 then
        R := R + s2[t2[i]];
      else
        R := R - s2[Abs(t2[i])];
      end if;
    end if;
  end for;
  return R, k1, k2, Q;
end function;

p := NextPrime(Random([4..2^32]));
repeat
  p := NextPrime(p);
  F := GF(p);
until (p mod 8) eq 5;
_<x> := PolynomialRing(F);
K<u> := ext<F|x^2-2>;

repeat
  a := Random(F);
  b := Random(F);
  E1 := EllipticCurve([a, b]);
  n1 := #E1;
  E2 := BaseExtend(E1, K);
  t := p+1-n1;
  ad := K!(u^2)*a;
  bd := K!(u^3)*b;
  E2d := EllipticCurve([ad, bd]);
  assert IsQuadraticTwist(E2,E2d);
  n2d := (p-1)^2 + t^2;
  assert #E2d eq n2d;
  S := Factorization(n2d);
  r := S[#S][1];
  h := n2d div r;
until h le 4;

R := Random(E2d);
P := h*R;
assert Order(P) eq r;
e := E2d![0,1,0];
assert r*P eq e;
i := Sqrt(F!-1);
k := Random([0..2^50]);
w := 3;

R, k1, k2, Q := mul(w, k, P, E2d, p, i);
R;
assert k*P eq R;
```

Code 5.7: MAGMA/gls–point–mul

Following table compare scalar multiplication algorithms, we eximine in this thesis. Let k is 256-bit scalar. Multiplication counts scalar multiplication operations that is used by algorithms, square counts taking square of a point, doubling counts point doubling operations, addition counts point addition operations and total part gives the sum of equivalents all operations to multiplication.

Comparisons of Scalar Multiplication Algorithms

Algorithm	Multiplication	Square	Addition	Total
Binary Right to Left	2543	1527	4458	26741M
Binary Left to Right	2541	1527	4457	26734M
Binary NAF	2036	1358	4159	24528M

Chapter 6

Conclusion

In this thesis we studied efficient scalar multiplication techniques Elliptic Curve Cryptography and their implementations. We gave algebraic background for elliptic curves and some fundamental elliptic curve background. With all these tools we aimed to reach the fastest scalar multiplication algorithm ever known. We made mathematical and computational analysis of fundamental scalar multiplication methods and their algorithms from the slowest to faster.

We gave mathematical analysis of the fastest scalar multiplication method deeply. We made computational analysis of it, compared it with other scalar multiplication method. We discussed prons and cons of each method. Then we talked about efficiently computable endomorphism which improves the fastest scalar multiplication method and eliminates performance disadvantages of the methods.

The implementation of all algorithms made using by MAGMA high level language and put to the appendix.

Appendix A

```
AddPoint := function(X1, Y1, Z1, X2, Y2, Z2, a, cntM, cntS, cntD, cntA, E)
 if (Z2 eq 0) then
  X3 := X1; Y3 := Y1; Z3 := Z1;
 elif (Z1 eq 0) then
  X3 := X2; Y3 := Y2; Z3 := Z2;
 elif (X1*Z2^2 eq X2*Z1^2) and (Y1*Z2^3 ne Y2*Z1^3) then
  X3 := 0; Y3 := 0; Z3 := 0;
 elif (X1*Z2^2 eq X2*Z1^2) and (Y1*Z2^3 eq Y2*Z1^3) then //doubling
  if Y1 eq 0 then
   X3 := 0; Y3 := 0; Z3 := 0;
  else
   A := Y1^2;          cntS+:=1;
   B := 4*X1*A;        cntM+:=1; cntA+:=2;
   C := 8*(A^2);       cntS+:=1; cntA+:=3;
   if a eq 0 then
    D := 3*(X1^2);        cntS+:=1; cntA+:=2;
   elif a eq -3 then
    D := 3*(X1-Z1^2)*(X1+Z1^2);   cntM+:=1; cntS+:=1; cntA+:=4;
   else
    D := 3*(X1^2)+a*Z1^4;     cntS+:=3; cntA+:=3; cntD+:=1;
   end if;
   X3 := D^2-2*B;       cntS+:=1; cntA+:=2;
   Y3 := D*(B-X3)-C;      cntM+:=1; cntA+:=2;
   Z3 := (2*Y1)*Z1;      cntA+:=1; cntM+:=1;
  end if;
 elif (X1*Z2^2 ne X2*Z1^2) then //addition
  A := Z1^2;           cntS+:=1;
  B := Z2^2;           cntS+:=1;
  C := X1*B;           cntM+:=1;
  D := X2*A;           cntM+:=1;
  E := Y1*Z2*B;        cntM+:=2;
  F := Y2*Z1*A;        cntM+:=2;
  G := D-C;           cntA+:=1;
  H := G^2;           cntS+:=1;
  I := G*H;           cntM+:=1;
  J := F-E;           cntA+:=1;
  K := C*H;           cntM+:=1;
  X3 := J^2-I-2*K;       cntS+:=1; cntA+:=3;
  Y3 := J*(K-X3)-E*I;       cntM+:=2; cntA+:=2;
  Z3 := Z1*Z2*G;        cntM+:=2;
 end if;
 return X3, Y3, Z3, cntM, cntS, cntD, cntA;
end function;
```

Code A.1: MAGMA/smul–ec–we–point–add

```
RTLPointMul := function(k, X1, Y1, a, E)
 cntM := 0; cntS :=0; cntD:=0; cntA:=0;
 X2 := 0; Y2 := 0; Z2 := 0; Z1 := 1;
 k:=IntegerToSequence(k, 2);
 t := #k;
 for i := 1 to t do
  if (k[i] eq 1) then
   X2, Y2, Z2, cntM, cntS, cntD, cntA := AddPoint(X1, Y1, Z1, X2, Y2, Z2, a, cntM, cntS, cntD,
       cntA, E);
  end if;
  X1, Y1, Z1, cntM, cntS, cntD, cntA := AddPoint(X1, Y1, Z1, X1, Y1, Z1, a, cntM, cntS, cntD,
      cntA, E);
  end for;
  if Z2 eq 0 then
    return 0,0,0,0,0,0;
  else
    return X2/Z2^2, Y2/Z2^3, cntM, cntS, cntD, cntA;
  end if;
end function;
```

Code A.2: MAGMA/smul–ec–we–binary–RtoL

```
LTRPointMul := function(k, X1, Y1, a, E)
  cntM := 0; cntS :=0; cntD:=0; cntA:=0;
  X2 := 0; Y2 := 0; Z2 := 0; Z1 := 1;
  k:=Reverse(IntegerToSequence(k, 2));
  t := #k;
  for i := 1 to t do
    X2, Y2, Z2, cntM, cntS, cntD, cntA := AddPoint(X2, Y2, Z2, X2, Y2, Z2, a, cntM, cntS, cntD,
        cntA, E);
    if (k[i] eq 1) then
      X2, Y2, Z2, cntM, cntS, cntD, cntA := AddPoint(X1, Y1, Z1, X2, Y2, Z2, a, cntM, cntS, cntD,
          cntA, E);
    end if;
  end for;
  if Z2 eq 0 then
    return 0,0,0,0,0,0;
  else
    return X2/Z2^2, Y2/Z2^3, cntM, cntS, cntD, cntA;
  end if;
end function;
```

Code A.3: MAGMA/smul–ec–we–binary–LtoR

```
Naf := function(k)
  i := 1;
  t := [];
  while (k ge 1) do
    if (k mod 2) eq 1 then
      t[i] := 2 - (k mod 4);
      k := k - t[i];
    elif (k mod 2) eq 0 then
      t[i] := 0;
    end if;
    k := k div 2;
    i := i + 1;
  end while;
  return t;
end function;
```

Code A.4: MAGMA/smul–ec–we–computing–NAF

```
NAFMul := function(k, X1, Y1, a, E)
  t := [];
  t := Reverse(Naf(k));
  l := #t;
  X2 := 0; Y2 := 0; Z2 := 0;
  Z1 := 1;
  cntM := 0; cntS :=0; cntD:=0; cntA:=0;
  for i := 1 to l do
    X2, Y2, Z2, cntM, cntS, cntD, cntA := AddPoint(X2, Y2, Z2, X2, Y2, Z2, a, cntM, cntS, cntD,
        cntA, E);
    if t[i] eq 1 then
      X2, Y2, Z2, cntM, cntS, cntD, cntA := AddPoint(X1, Y1, Z1, X2, Y2, Z2, a, cntM, cntS, cntD,
          cntA, E);
    elif t[i] eq -1 then
      X2, Y2, Z2, cntM, cntS, cntD, cntA := AddPoint(X1, -Y1, Z1, X2, Y2, Z2, a, cntM, cntS, cntD
          , cntA, E);
    end if;
  end for;
  if Z2 eq 0 then
    return 0, 0;
  else
    return X2/Z2^2, Y2/Z2^3, cntM, cntS, cntD, cntA;
  end if;
end function;
```

Code A.5: MAGMA/smul–ec–we–binary–NAF

```
wNaf := function(w, k)
 i := 1;
 t := [];
 while (k ge 1) do
  if (k mod 2) eq 1 then
   t[i] := k mod (2^w);
   k := k - t[i];
  else
   t[i] := 0;
  end if;
  k := k div 2;
  i := i + 1;
 end while;
 return t;
end function;
```

Code A.6: MAGMA/smul–ec–we–computing–wNAF

```
wNafMul := function(w, k, X1, Y1, a, E)
 t := []; s:=[];
 cntM := 0; cntS :=0; cntD:=0; cntA:=0;
 t := Reverse(wNaf(w, k));
 l := #t;
 Z1 := 1;
 X := 0; Y :=0; Z:=0;
 for i:= 1 to (2^w) by 2 do
  for j:=1 to i do
   X, Y, Z, cntM, cntS, cntD, cntA := AddPoint(X, Y, Z, X1, Y1, Z1, a, cntM, cntS, cntD, cntA,
         E);
  end for;
  s[i] := <X,Y,Z>;
 end for;
 X2 := 0; Y2 := 0; Z2 := 0;
 for i:=l to 1 by -1 do
  for j:=1 to w do
   X2, Y2, Z2, cntM, cntS, cntD, cntA := AddPoint(X2, Y2, Z2, X2, Y2, Z2, a, cntM, cntS, cntD,
         cntA, E);
  end for;
  if t[i] ne 0 then
   if t[i] gt 0 then
    X2, Y2, Z2, cntM, cntS, cntD, cntA:= AddPoint(X2, Y2, Z2, s[t[i],1], s[t[i],2], s[t[i
         ],3], a, cntM, cntS, cntD, cntA, E);
   elif t[i] lt 0 then
    X2, Y2, Z2, cntM, cntS, cntD, cntA := AddPoint(X2, Y2, Z2, s[-t[i],1], -s[-t[i],2], s[-t[
         i],3], a, cntM, cntS, cntD, cntA, E);
   end if;
  end if;
 end for;
 return X2/Z2^2, Y2/Z2^3, cntM, cntS, cntD, cntA;
end function;
```

Code A.7: MAGMA/smul–ec–we–wNAF

```
p := NextPrime(Random([0..2^256]));
repeat
   p := NextPrime(p);
 F := GF(p);
until IsSquare(F!-1) eq true;
a := Random(F);
b := 0;
E := EllipticCurve([a, b]);
u := Modsqrt(-1, p);
n := #E;
t := p+1-n;
S := Max(Factorization(n));
r := S[1];
h := n div r;
h le 4;
R := Random(E);
P := h*R;
Order(P) eq r;
e := E![0,1,0];
r*P eq e;

EGCD := function(r, u)
 s0 := 1; s1 := 0;
 t0 := 0; t1 := 1;
 r0 := r; r1 := u;
 while (r0^2 ge r) do
  q := r0 div r1;
```

```
   r2 := r0 - q*r1;
   r0 := r1;
   r1 := r2;
   s2 := s0 - q*s1;
   s0 := s1;
   s1 := s2;
   t2 := t0 - q*t1;
   t0 := t1;
   t1 := t2;
  end while;
  v1a := r1;
  v1b := -t1;
  v2a := r0;
  v2b := -t0;
  return v1a, v1b, v2a, v2b, s1, s0;
end function;

r1, t1, r0, t0, s1, s0 := EGCD(r, u);
s1*r - t1*u eq r1;
s0*r - t0*u eq r0;
Log(2, p);
Log(2, h);
Log(2, r);
```

Code A.8: MAGMA/EGCD

```
p := NextPrime(Random([0..2^32]));
repeat
   p := NextPrime(p);
until (p mod 4) eq 1;
F := GF(p);
a := Random(F);
b := 0;
E := EllipticCurve([a, b]);
n := #E;
//t := p+1-n;
S := Max(Factorization(n));
r := S[1];
h := n div r;
h le 4;
R := Random(E);
P := h*R;
Order(P) eq r;
e := E![0,1,0];
r*P eq e;
u := Modsqrt(-1, p);
l := Modsqrt(-1, r);

EGCD := function(r, l)
  /*s0 :=smul-ec-we-GLV-c1-v03.m 1; s1 := 0;*/
  t0 := 0; t1 := 1;
  r0 := r; r1 := l;
  while (r0^2 ge r) do
   q := r0 div r1;
   r2 := r0 -smul-ec-we-GLV-c1-v03.m q*r1;
   r0 := r1;
   r1 := r2;
   /*s2 := s0 - q*s1; s0 := s1; s1 := s2;*/
   t2 := t0 - q*t1;
   t0 := t1;
   t1 := t2;
  end while;
  v1a := r1;
  v1b := -t1;
  v2a := r0;
  v2b := -t0;
  return v1a, v1b, v2a, v2b/*, s1, s0*/;
end function;

Phifunc := function(u, P, E)
  Q := E![-P[1],u*P[2]];
  return Q;
end function;

v1a, v1b, v2a, v2b := EGCD(r, l);
k := Random([0..r]);
B := v1a*v2b - v1b*v2a;

b1 := Round((k*v2b)/B);
b2 := Round((k*v1b)/-B);

va := b1*v1a + b2*v2a;
```

```
vb := b1*v1b + b2*v2b;
k1 := k-va;
k2 := 0-vb;

f := function(i, j, l)
 return (i+ l*j) mod r;
end function;

f(v1a, v1b, l);
f(v2a, v2b, l);

Q := Phifunc(u, P, E);

if (Q ne l*P) then
 u := (-u) mod p;
 Q := Phifunc(u, P, E);
end if;

k*P eq (k1*P+k2*l*P);
k*P eq (k1*P+k2*Q);

k;
k1;
k2;
```

Code A.9: MAGMA/smul–ec–we–GLV–j1728

```
while (true) do
 p := NextPrime(Random([0..2^32]));
 repeat
  p := NextPrime(p);
 until (p mod 3) eq 1;
 F := GF(p);
 a := 0;
 b := Random(F);
 E := EllipticCurve([a, b]);
 n := #E;
 t := p+1-n;
 S := Max(Factorization(n));
 r := S[1];
 h := n div r;
 h le 4;
 R := Random(E);
 P := h*R;
 assert Order(P) eq r;
 e := E![0,1,0];
 assert r*P eq e;
 x := Modsqrt(-3, p);
 y := Modsqrt(-3, r);
 u := ((-1 + x)*Modinv(2,p)) mod p;
 l := ((-1 + y)*Modinv(2,r)) mod r;
 T := l*P;
 if ((u*P[1]) ne T[1]) or (P[2] ne T[2]) then
  u := ((-1 - x)*Modinv(2,p)) mod p;
  if ((u*P[1]) ne T[1]) or (P[2] ne T[2]) then
   l := ((-1 - y)*Modinv(2,r)) mod r;
   if ((u*P[1]) ne T[1]) or (P[2] ne T[2]) then
    u := ((-1 + x)*Modinv(2,p)) mod p;
   end if;
  end if;
 end if;

 EGCD := function(r, l)
  /*s0 := 1; s1 := 0;*/
  t0 := 0; t1 := 1;
  r0 := r; r1 := l;
  while (r0^2 ge r) do
   q := r0 div r1;
   r2 := r0 - q*r1;
   r0 := r1;
   r1 := r2;
   /*s2 := s0 - q*s1; s0 := s1; s1 := s2;*/
   t2 := t0 - q*t1;
   t0 := t1;
   t1 := t2;
  end while;
  v1a := r1;
  v1b := -t1;
  v2a := r0;
  v2b := -t0;
  return v1a, v1b, v2a, v2b/*, s1, s0*/;
 end function;
```

```

  Phifunc := function(u, P, E)
    Q := E![u*P[1],P[2]];
    return Q;
  end function;

  v1a, v1b, v2a, v2b := EGCD(r, l);
  k := Random([0..r]);
  B := v1a*v2b - v1b*v2a;
  b1 := Round((k*v2b)/B);
  b2 := Round((k*v1b)/-B);
  va := b1*v1a + b2*v2a;
  vb := b1*v1b + b2*v2b;
  k1 := k-va;
  k2 := 0-vb;

  Q := Phifunc(u, P, E);
  assert k*P eq (k1*P+k2*l*P);
  assert k*P eq (k1*P+k2*Q);
end while;
```

Code A.10: MAGMA/smul–ec–we–GLV–j0

```
p := NextPrime(Random([4..2^256]));
repeat
  p := NextPrime(p);
  F := GF(p);
until (p mod 8) eq 5;
_<x> := PolynomialRing(F);
K<u> := ext<F|x^2-2>;

repeat
  a := Random(F);
  b := Random(F);
  E1 := EllipticCurve([a, b]);
  n1 := #E1;
  E2 := BaseExtend(E1, K);
  t := p+1-n1;
  ad := K!(u^2)*a;
  bd := K!(u^3)*b;
  E2d := EllipticCurve([ad, bd]);
  assert IsQuadraticTwist(E2,E2d);
  n2d := (p-1)^2 + t^2;
  assert #E2d eq n2d;
  S := Factorization(n2d);
  r := S[#S][1];
  h := n2d div r;
until h le 4;

R := Random(E2d);
P := h*R;
assert Order(P) eq r;
e := E2d![0,1,0];
assert r*P eq e;
l := (Modinv(t,r)*(p-1)) mod r;
i := Sqrt(F!-1);
T := E2d![-P[1]^p, i*(P[2]^p)];

//Select the correct square root of -1.
assert T eq l*P or T eq -l*P;
if T ne l*P then
  assert T eq -l*P;
  i:=-i;
end if;

Psifunc := function(P, E2d, p, i)
  return E2d![-P[1]^p,i*(P[2]^p)];
end function;

k := Random([0..r]);
B := (p-1)^2 + t^2 ;
b1 := (k*(t))/B;
b2 := (k*(p-1))/B;
v1 := Round(b1)*t + Round(b2)*(p-1);
v2 := Round(b1)*(p-1) - Round(b2)*t;
k1 := k-v1;
k2 := 0-v2;

[p^2, r, k, k1, k2];

assert k*P eq (k1*P+k2*l*P);
Q := Psifunc(P, E2d, p, i);
```

```
assert k*P eq (k1*P+k2*Q);
assert l*P eq Q;
```

Code A.11: MAGMA/smul–ec–we–GLS

```
dec := function(k, p, E)
 n:= #E;
 t := p+1-n;
 B := (p-1)^2 + t^2 ;
 b1 := (k*(t))/B;
 b2 := (k*(p-1))/B;
 v1 := Round(b1)*t + Round(b2)*(p-1);
 v2 := Round(b1)*(p-1) - Round(b2)*t;
 k1 := k-v1;
 k2 := 0-v2;
 return k1, k2;
end function;

phi := function(P, E, p, j)
 return E![-P[1]^p, j*(P[2]^p)];
end function;

wNaf := function(w, k)
 i := 1;
 t := [];
 while (k ge 1) do
  if (k mod 2) eq 1 then
   t[i] := k mod (2^w);
   k := k - t[i];
  elif (k mod 2) eq 0 then
   t[i] := 0;
  end if;
  k := k div 2;
  i := i + 1;
 end while;
 return t;
end function;

mul := function(w, k, P, E, p, j)
 t1 := [];
 t2 := [];
 s1 := [];
 s2 := [];
 k1, k2 := dec(k, p, E);
 Q := phi(P, E, p, j);
 t1 := Reverse(wNaf(w, k1));
 m1 := #t1;
 t2 := Reverse(wNaf(w, k2));
 m2 := #t2;
 m:=Maximum(m1, m2);
 if m1 eq m then
  for i:=(m2+1) to m do
   t2[i]:=0;
  end for;
 elif m2 eq m then
  for i:=(m1+1) to m do
   t1[i]:=0;
  end for;
 end if;
 if k1 lt 0 then
  for i:=1 to m1 do
   t1[i]:=-t1[i];
  end for;
 end if;
 if k2 lt 0 then
  for i:=1 to m2 do
   t2[i]:=-t2[i];
  end for;
 end if;
 for i:=1 to (2^w-1) by 2 do
  s1[i] := i*P;
  s2[i] := i*Q;
 end for;
 R := E![0,1,0];
 for i:=m to 1 by -1 do
  R := 2*R;
  if t1[i] ne 0 then
   if t1[i] gt 0 then
    R := R + s1[t1[i]];
   else
    R := R - s1[Abs(t1[i])];
   end if;
```

```
    end if;
    if t2[i] ne 0 then
      if t2[i] gt 0 then
        R := R + s2[t2[i]];
      else
        R := R - s2[Abs(t2[i])];
      end if;
    end if;
  end for;
  return R, k1, k2, Q;
end function;

p := NextPrime(Random([4..2^32]));
repeat
  p := NextPrime(p);
  F := GF(p);
until (p mod 8) eq 5;
_<x> := PolynomialRing(F);
K<u> := ext<F|x^2-2>;

repeat
  a := Random(F);
  b := Random(F);
  E1 := EllipticCurve([a, b]);
  n1 := #E1;
  E2 := BaseExtend(E1, K);
  t := p+1-n1;
  ad := K!(u^2)*a;
  bd := K!(u^3)*b;
  E2d := EllipticCurve([ad, bd]);
  assert IsQuadraticTwist(E2,E2d);
  n2d := (p-1)^2 + t^2;
  assert #E2d eq n2d;
  S := Factorization(n2d);
  r := S[#S][1];
  h := n2d div r;
until h le 4;

R := Random(E2d);
P := h*R;
assert Order(P) eq r;
e := E2d![0,1,0];
assert r*P eq e;
i := Sqrt(F!-1);
k := Random([0..2^50]);
w := 3;

R, k1, k2, Q := mul(w, k, P, E2d, p, i);
R;
assert k*P eq R;
```

Code A.12: MAGMA/gls–point–mul

```
p := 115792089210356248762697446949407573530086143415290314195533631308867097853951; //
     NextPrime(Random([4..2^256]));
F := GF(p);
a := F!-3;
b := F!0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b;
E := EllipticCurve([a, b]);

TcntM := 0;
TcntS := 0;
TcntD := 0;
TcntA := 0;
if (-16)*(4*(a^3)+27*(b^2)) ne 0 then
  R := Random(E);
  if R ne E!0 then
    for i:=1 to 1000 do
      k := Random(115792089210356248762697446949407573529996955224135760342422259061068512044369)
          ;
      t := k*R;
      x, y, cntM, cntS, cntD, cntA:=RTLPointMul(k, R[1], R[2], a, E);
      if (t[1] ne x) or (t[2] ne y) then
        t[1] eq x;
        t[2] eq y;
        Order(R);
        print a, b, k, R;
      else
        print x, y, cntM, cntS, cntD, cntA;
        print "--------------";
        TcntM := TcntM+cntM;
        TcntS := TcntS+cntS;
      TcntD := TcntD+cntD;
      TcntA := TcntA+cntA;
    end if;
  end for;
 end if;
end if;
print"END";
TcntM/1000 + 0.0; TcntS/1000 + 0.0; TcntD/1000 + 0.0; TcntA/1000 + 0.0;
```

Code A.13: MAGMA/test–code

Bibliography

[1] Maple 12, Waterloo Maple Inc., 2008. `http://www.maplesoft.com/`.

[2] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, New York, NY, USA, 1999.

[3] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag.

[4] W. Bosma, J. Cannon, and C. Playoust. The MAGMA algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[5] W. Bosma and H. W. Lenstra Jr. Complete systems of two addition laws for elliptic curves. *Journal of Number Theory*, 53:229–240, 1995.

[6] E. Brier and M. Joye. Weierstraß elliptic curves and side-channel attacks. In *PKC 2002*, volume 2274 of *LNCS*, pages 335–345. Springer, 2002.

[7] E. Brier and M. Joye. Fast point multiplication on elliptic curves through isogenies. In *AAECC-15*, volume 2643 of *LNCS*, pages 43–50. Springer, 2003.

[8] D. Chudnovsky and G. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Adv. Appl. Math.*, 7(4):385–434, Dec. 1986.

[9] H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[10] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT ’98, pages 51–65, London, UK, UK, 1998. Springer-Verlag.

[11] D. A. Cox, J. B. Little, and D. O’Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, 3rd ed. 2007.

[12] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

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

[14] J. B. Fraleigh. *A first course in abstract algebra (Addison-Wesley series in mathematics)*. Addison-Wesley Pub. Co, July 1976.

[15] S. D. Galbraith, X. Lin, and M. Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 518–535. Springer, 2009.

[16] S. D. Galbraith, X. Lin, and M. Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. Cryptology ePrint Archive, Report 2008/194, extended version, 17-Sep-2009, 2009. http://eprint.iacr.org.

[17] S. D. Galbraith, X. Lin, and M. Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. *J. Cryptology*, 24(3):446–469, 2011.

[18] R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In *CRYPTO*, pages 190–200, 2001.

[19] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[20] Z. Hu, P. Longa, and M. Xu. Implementing the 4-dimensional glv method on gls elliptic curves with j-invariant 0. pages 331–343, 2012.

[21] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[22] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, Jan. 1987.

[23] K. Koyama and Y. Tsuruoka. Speeding up elliptic cryptosystems by using a signed binary window method. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO ’92, pages 345–357, London, UK, UK, 1993. Springer-Verlag.

[24] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, Cambridge, revised edition, 1994.

[25] P. Longa and F. Sica. Four-dimensional gallant-lambert-vanstone scalar multiplication. pages 248–283, 2014.

[26] V. S. Miller. Use of elliptic curves in cryptography. In *Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[27] A. Miyaji, T. Ono, and H. Cohen. Efficient elliptic curve exponentiation. In *Proceedings of the First International Conference on Information and Communication Security*, ICICS ’97, pages 282–291, London, UK, UK, 1997. Springer-Verlag.

[28] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Theoretical Informatics and Applications*, 24:531–543, 1990.

[29] J. A. Muir and D. R. Stinson. Minimality and other properties of the width-w nonadjacent form, 2004.

[30] Y. Nogami and Y. Morikawa. Fast generation of elliptic curves with prime order over extension field of even extension degree. In *Information Theory, 2003. Proceedings. IEEE International Symposium on*, pages 18–18, June 2003.

[31] U. N.S.A. Fact sheet nsa suite b cryptography, u.s. national security agency.

[32] G. W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.

[33] R. Schoof. *Elliptic curves over finite fields and the computation of square roots mod p*. Report. Department of Mathematics. University of Amsterdam. Department, Univ., 1983.

[34] J. H. Silverman. *The arithmetic of elliptic curves*. Graduate texts in mathematics. Springer, New York, Berlin, 1986. 2e tirage corrigé 1992.

[35] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, 1st ed. 1986. Corr. 3rd printing, 1994.

[36] J. A. Solinas. Efficient arithmetic on koblitz curves. *Des. Codes Cryptography*, 19(2-3):195–249, Mar. 2000.

[37] S. S. Wagstaff. *Cryptanalysis of Number Theoretic Ciphers*. CRC Press Inc., October 2002.

[38] L. C. Washington. *Elliptic Curves: Number Theory and Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 2003.

[39] W. WTLS. Wireless application protocol wireless transport layer security specification. Feb. 1999.