理學碩士 學位論文

타원곡선 암호시스템의 핵심 연산에 대한 효율성의 비교와 분석

Comparison and analysis on efficiency of scalar multiplication for Elliptic Curve Cryptosystem

指導敎授 金 宰 煥

2003年 2月

韓國海洋大學校 大學院

應用數學科 金 建 澔

목 차

ABSTRACT

Ⅰ. 서론 ····· 1

Ⅱ. 타원곡선 암호시스템의 개요 ····· 3
2.1. 타원곡선과 타원곡선 암호의 개념 ····· 3
2.1.1. 일반적인 타원곡선의 정의 및 성질 ····· 3
2.1.2. 유한체 위에서의 타원곡선 암호의 개념 ····· 7
2.2. 타원곡선을 이용한 암호시스템 ····· 10
2.2.1. ECDH(Elliptic Curve Diffie-Hellman) ····· 11
2.2.2. EC-ElGamal ····· 12
2.2.3. EC-Massey-Omura ····· 12
2.2.4. ECDSA(Elliptic Curve DSA) ····· 13
2.3. 기존의 공개키 암호시스템과의 비교 ····· 15
2.3.1. 기존의 공개키 암호시스템의 소개 ····· 16
2.3.2. ECC와 기존 암호시스템과의 비교 ····· 18
2.4. ECC 관련 동향 및 전망 ····· 22
2.4.1. ECC의 표준화 관련 동향 ····· 22
2.4.2. ECC의 적용 분야에 대한 동향 및 전망 ····· 24

Ⅲ. ECC의 핵심 연산 알고리듬 비교 및 분석 ····· 27
3.1. 유한체에서의 연산알고리듬 ····· 27
3.2. 타원곡선군에서의 연산알고리듬 ····· 30
3.3. 주요 연산알고리듬의 비교 및 분석 ····· 34

Ⅳ. 결론 및 향후 과제 ····· 38

참고문헌 ····· 39

부록 ····· 41

그림 · 표 목차

<그림 2-1> 타원곡선 위의 연산 ····· 5
<표 2-1> 타원곡선의 정의체에 따른 도메인 변수와 조건 ····· 8
<그림 2-2> ECDH 키 교환 모델 ····· 11
<그림 2-3> EC-Massey-Omura 메시지 전송 모델 ····· 13
<그림 2-4> ECDSA를 이용한 전자서명 모델 ····· 15
<표 2-2> 유한체와 타원곡선의 비교 ····· 18
<표 2-3> 같은 안전도에 따른 도메인 변수의 크기 비교 ····· 19
<그림 2-5> ECC와 RSA/DSA의 안전도 수준 비교 ····· 20
<표 2-4> DH, ECDH, DSA, ECDSA 알고리듬의 속도 비교 ····· 21
<표 2-5> 전세계 무선인터넷 사용자 전망 ····· 24
<그림 2-6> 국내 Mobile Commerce 시장 규모 전망 ····· 25
<표 2-6> 세계 정보보호 시장 동향 및 전망 ····· 26
<표 2-7> 국내 정보보호 시장 동향 및 전망 ····· 26
<표 3-1> 타원곡선 상의 상수배 연산 알고리듬 수행 속도 비교 ····· 35
<그림 3-1> 타원곡선 상의 상수배 연산 알고리듬 수행 속도 비교 ···· 37

ABSTRACT

In this thesis, we study scalar multiplication algorithms which play an important role in the field of ECC(Elliptic Curve Cryptosystem). They mainly consist in Binary method, Binary NAF method, and Sliding Window method. We compare them in terms of the computing time and analyze their computational efficiency based on the utilization of memory usage. As a result, we notice that Binary NAF method is the most efficient in the computing time among them, and the efficiency of the other methods are not bad to apply in ECC.

Ⅰ. 서론

무선단말기 보급의 증가와 더불어 무선인터넷 사용자가 급속하게 증가하고 있는 추세와 비교해서 현재의 무선인터넷의 보안 수준은 초기 단계에 불과 하다고 할 수 있다. 이는 무선 단말기와 무선인터넷의 특수한 환경과 밀접한 연관이 있다. 즉, 기존의 유선의 장비와 비교해 볼 때 낮은 통신의 대역폭을 가지며, CPU와 메모리의 리소스가 작고, 배터리의 수명이 짧으며, 사용자의 인터페이스가 부족하다는 것 등이다[8]. 그러나 이러한 제약에도 불구하고 무선단말기를 이용한 무선인터넷의 사용이 증가하는 이유는 이와 같은 제약이 계속해서 보완되고 있으며, 또한 기존의 On-Line 시스템에서 제공하지 못하는 이동성과 편의성을 동시에 제공한다는 것이 주요한 요인으로 작용하고 있다. 이러한 무선인터넷의 효용성에 기초한 무선인터넷의 발전과 발맞추어 무선 환경의 보안 문제는 아주 중요한 분야이다. 이와 함께 무선시스템의 IWF 망 개방 정책에 따라, 기존의 On-Line 시스템과의 연계가 이루어지고 있다. 망 개방이 완전히 이루어지게 되면 유선과 무선의 호환성이 보장되는 보안 대책이 강구되어야 한다. 현재 이러한 보안 대책에 대해 여러 학자들과 관련 기업, 연구 기관 등을 통해 계속해서 연구가 이루어지고 있다.

무선인터넷과 망 개방에 따른 유·무선 통합 보안에 대해 현재 여러 가지 방안들이 제시되고 있으나 그중 가장 효율적인 방안으로 주목받고 있는 보안 대책이 바로 ECC(Elliptic Curve Cryptosystem)이다. ECC는 기존의 On-Line 시스템의 보안을 책임지고 있던 RSA/DSA에 비해 수행속도 면에서나 메모리의 효율적인 측면 그리고 보안성 등에서 이미 RSA/DSA를 상당히 능가하는 것으로 여러 논문들이나 관련자료 등은 말하고 있다[6][14][15][24]. 이러한 ECC를 무선 환경에서 실제 상용화에 적용하기 위해서는 아직까지 여러 가지 문제들이 남아있다. 그 중 가장 핵심이 되는 요인이 바로 ECC의 기반이 되는 타원곡선군의 원소들의 효율적인 연산에 대한 것이다.

ECC는 유한체 위에 정의된 타원곡선(elliptic curve) 위의 점들이 덧셈에 대해 군(group)을 이루며, 이러한 타원곡선군의 원소에 대한 덧셈 연산을 기초로 시스템이 수행된다. 이러한 연산은 타원곡선이 정의되는 정의체에 따라서 달라질 수 있으며, 또한 타원곡선의 형태에 따라서도 달라질 수 있다. 본 논문에서는 이러한 여러 가지 정의체와 타원곡선의 형태에 따른 연산에 대해 현재까지 연구된 다양한 연산방법의 효율성에 대해 비교, 분석하고자 한다.

본 논문은 Ⅰ장의 서론에 이어 Ⅱ장에서는 ECC의 기본이 되는 타원곡선의 정의 및 성질과 유한체에서의 타원곡선 암호의 기본 개념과 몇 가지 타원곡선을

이용한 암호시스템들을 소개한다. 그리고 기존의 공개키 암호 시스템인 RSA/DSA에 대해서 간략히 소개하고 ECC와 비교하여 ECC의 효율성을 알아보며, ECC관련 표준화 동향과 ECC의 적용 분야에 대한 동향과 전망에 대해 알아본다. 그리고 Ⅲ장에서는 유한체 상에서의 여러 가지 연산 알고리듬과 타원곡선 상에서의 암호연산 알고리듬에 대해 분석하여 그 중 주요 연산알고리듬을 구현하여 수행속도 등에 대한 비교를 통하여 효율적인 연산 알고리듬을 알아본다. Ⅳ장에서는 본 논문에서 연구한 내용에 대한 결론과 향후의 과제 등에 대해 기술하고자 한다.

Ⅱ. 타원곡선 암호시스템의 개요

타원곡선(Elliptic Curves) 이론은 약 150년 전부터 수학에서 광범위하게 연구되어 왔으며, 최근 Andrew Wiles의 Fermat's Last Theorem 증명에서 중요하게 사용되었다[1]. 이러한 타원곡선 이론은 1985년 Koblitz와 Miller에 의해 타원곡선 암호시스템(ECC : Elliptic Curve Cryptosystem)이 발표된 이후 지금까지 꾸준한 연구의 대상이 되고 있다. 초기에 Koblitz와 Miller가 타원곡선 암호이론을 발표할 때만 해도 타원곡선군에서의 연산의 구현이 어려워 상용화에 대해 회의적이었지만, 그 후 많은 연구들에 의해 구현이 용이하게 되었고 그 결과로 타원곡선을 이용한 암호는 주목을 받기 시작했다. 이러한 구현의 문제가 해결되고 또한 기존의 RSA/DSA 암호시스템과 비교해서 훨씬 빠르고 효율적이며 짧은 키 길이를 갖는 등 여러 가지 장점을 갖고 있어서 타원곡선 암호시스템에 대한 관심은 더욱 증대되고 있다.

본 장에서는 타원곡선 암호알고리듬이 구성되는 내용을 알아보기 위해 타원곡선의 정의 및 타원곡선군에서 정의된 기본적인 연산 등과 유한체에서의 타원곡선 암호의 기본 개념을 알아보며, 타원곡선 이용하는 몇 가지의 암호시스템에 대해 알아본다. 그리고 기존의 공개키 암호시스템과 ECC를 비교하여 ECC의 효율성에 대해 알아보며, ECC의 동향과 전망에 대해 살펴보고자 한다.

2.1. 타원곡선과 타원곡선 암호의 개념

본 절에서는 일반적인 체(field)상에서 정의된 타원곡선에 대해서 알아보고, 타원곡선이 갖는 여러 특징들을 알아보며, 암호에서 사용되는 유한체상에서 정의된 타원곡선에 대해 어떤 방식으로 타원곡선 암호가 형성되는지 알아본다.

2.1.1. 일반적인 타원곡선의 정의 및 성질

타원곡선은 체(field)상에서 주어진 Weierstrass equation을 만족하는 곡선으로 다음과 같이 정의된다[2].

정의 2.1.1

체(field) $\boldsymbol{K}$ 위에서의 Weierstrass equation

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad a_i(1 \leqq i \leqq 6) \in \boldsymbol{K} \quad \cdots\cdots (1)$$

를 만족하는 모든 점 (x, y)들과 무한원점(point at infinity) O를 갖는 집합을 체(field) $\boldsymbol{K}$ 위에서의 타원곡선(Elliptic Curve)이라 하고 $E(\boldsymbol{K})$로 나타낸다. 즉,

$$E(\boldsymbol{K}) = \{(x, y) \in \boldsymbol{K} \times \boldsymbol{K} \mid y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, a_i \in \boldsymbol{K}\} \cup \{O\}$$

이다.

위의 <u>정의 2.1.1</u> 을 만족하는 타원곡선이 주어질 경우 연산은 다음과 같이 정의되며, <그림 2-1>과 같은 의미를 갖는다.

<u>정의 2.1.2</u>

$P_i(x_i, y_i)$, $(i = 1, 2)$를 타원곡선 위의 점이라고 하면 이들 사이의 덧셈을 다음과 같이 정의한다.

(1) 항등원 : O, 모든 $P \in E(\boldsymbol{K})$에 대해서 $P + O = O + P = P$이다.

(2) P_1의 역원 : $-P_1 = (x_1, -y_1 - a_1x_1 - a_3)$,

$$P_1 + (-P_1) = (-P_1) + P_1 = O \ .$$

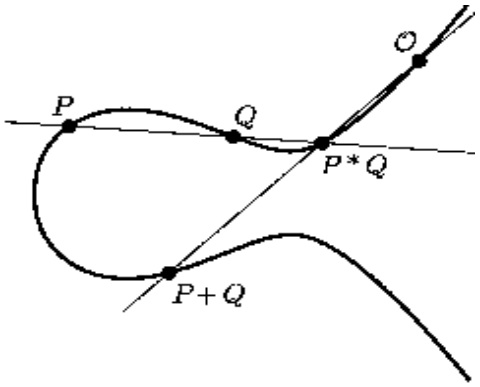
(3) 덧셈 : $P_1 + P_2 = P_3(x_3, y_3)$,

$$P_3 = \begin{cases} O, & \text{if } x_1 = x_2 \text{ and } y_1 + y_2 + a_1x_2 + a_3 = 0 \\ (\lambda^2 + a_1\lambda - a_2 - x_1 - x_2, -(\lambda + a_1)x_3 - \nu - a_3), & otherwise \end{cases}$$

$$\text{여기서, } \lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } x_1 \neq x_2 \\ \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}, & \text{if } x_1 = x_2 \end{cases},$$

$$\nu = \begin{cases} \dfrac{y_1x_2 - y_2x_1}{x_2 - x_1}, & \text{if } x_1 \neq x_2 \\ \dfrac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x_1 + a_3}, & \text{if } x_1 = x_2 \end{cases}$$

위와 같이 연산을 정의하면, 타원곡선 위의 모든 점들은 O를 항등원으로 하는 덧셈에 대한 군(group)을 이루게 된다.



<그림 2-1> 타원곡선 위의 연산

타원곡선을 나타내는 방정식은 **정의 2.1.1** 에서와 같이 일반적으로 표현이 되지만, 방정식이 정의되는 정의체의 표수(characteristic)에 따라 좀 더 간략하게 몇 가지 형태로 변형시킬 수 있다.

정리 2.1.3

체(field) $\boldsymbol{K}$의 표수(characteristic)를 char$\boldsymbol{K}$라 할 때, 그 표수에 따라 타원곡선을 나타내는 방정식은 다음과 같이 표현된다.

(ⅰ) char$\boldsymbol{K}=2$: $y^2+a_3y=x^3+a_4'x+a_6'$ ····· (2)

또는 $y^2+xy=x^3+a_2'x^2+a_6'$ ····· (3)

(ⅱ) char$\boldsymbol{K}=3$: $y^2=x^3+ax^2+bx+c$ ····· (4)

(ⅲ) char$\boldsymbol{K}>3$: $y^2=x^3+bx+c$ ····· (5)

[증명] (ⅰ) $a_1=0$일 때 **정의 2.1.1**에서의 식 (1)은

$$y^2+a_3y=x^3+a_2x^2+a_4x+a_6$$
$$=(x+a_2)^3+(a_2^2+a_4)x+a_2^3+a_6 \text{ 이므로,}$$

변수변환 $(x,\ y)\rightarrow(x+a_2, y)$을 하고 $a_2^{\,2}+a_4=a_4'$과 $a_2^{\,3}+a_6=a_6'$라 놓으면, 식 (2)가 된다. 같은 방법으로 $a_1\neq 0$일 때, 변수변환 $(x,\ y)\rightarrow\left(a_1^{\,2}+\frac{a_3}{a_1},\ a_1^{\,3}+\frac{a_1^{\,3}a_4+a_3^{\,2}}{a_1^{\,3}}\right)$에 의하여 (3)이 된다.

(ii) 변수변환 $(x,\ y)\rightarrow\left(x,\ y-\frac{a_1}{2}x-\frac{a_3}{2}\right)$을 하면

$y^2=x^3+\left(\frac{a_1^{\,2}}{4}+a_2\right)x^2+\left(\frac{a_1a_3}{2}+a_4\right)x+\left(\frac{a_3^{\,2}}{4}+a_6\right)$이므로, (4)를 얻는다.

(iii) (ii)에 의해 변형된 식을 $y^2=x^3+b_2x^2+b_4x+b_6$라 할 때,

변수변환 $(x,\ y)\rightarrow\left(\frac{x-3b_2}{36},\ \frac{y}{216}\right)$에 의하여 (5)를 얻는다.

정의체의 표수에 따라 **정리 2.1.3** 에서와 같이 변형 된 방정식은 Weierstrass 등식의 계수들의 적당한 조합으로 discriminant Δ를 정의할 수 있는데, Δ가 0이 아닐 때 이 방정식은 타원곡선을 이루고, 0일 때는 특이점(singular point)을 갖게 되어 타원곡선이 아니다. 또한 $j-invariant$라는 값을 정의할 수 있는데, 이 값이 같은 두 타원곡선은 정의체 $\boldsymbol{K}$위에서 동형(isomorphic)일 필요충분조건이 된다. 이러한 Δ와 $j-invariant$는 다음과 같은 값으로 정의된다.

$\mathrm{char}\boldsymbol{K}=2$: (2)인 경우 $\Delta=a_3^{\,4},\ j=0$, (3)인 경우 $\Delta=a_6',\ j=1/a_6'$ 이다.

$\mathrm{char}\boldsymbol{K}=3$: (4)식은 다시 $y^2=x^3+a_2x^2+a_6\cdots\cdots(4)'$,
$y^2=x^3+a_4x+a_6\cdots\cdots(4)''$ 처럼 두 가지 형태로 표현되기도 하는데, 여기서 (4)′인 경우 $\Delta=-a_2^{\,3}a_6,\ j=-a_2^{\,3}/a_6$이고, (4)″인 경우 $\Delta=-a_4^{\,3},\ j=0$이 된다.

$\mathrm{char}\boldsymbol{K}>3$: 이 경우 $\Delta=-16(a^3+27b^2),\ j=(1728\times 4a^3)/\Delta$이 된다.

암호시스템에서는 유한체(finite field) 위에 정의된 타원곡선을 주로 사용하게 되는데, 특히, $\mathrm{char}\boldsymbol{K}=2$인 경우와 $\mathrm{char}\boldsymbol{K}>3$인 경우의 타원곡선을 주로 사용

한다. char$\boldsymbol{K}=2$인 경우의 ⑵식을 초특이(supersingular) 타원곡선이라 한다. 초특이 타원곡선의 경우 암호시스템에 적합하지 않으며, 따라서 암호시스템에서는 ⑶식을 사용한다[15].

2.1.2. 유한체 위에서의 타원곡선 암호의 개념

타원곡선 암호알고리듬은 유한체(finite field) 상에 정의된 타원곡선 이산대수문제(elliptic curve discrete logarithm problem)를 기반으로 한다. 타원곡선 이산대수문제의 정의는 다음과 같다[1].

정의 2.1.4

유한체 $\mathbf{F}_q$위에 정의된 타원곡선 E와 그 위의 점 P가 있을 때, 기점을 P로 갖는 E의 이산대수문제란 E 위의 점 Q가 주어졌을 때 $Q=nP$를 만족하는 n을 찾는 문제이다.

실제로 암호에 적용되는 타원곡선의 이산대수문제를 해결하기는 매우 어려우며 이를 해결하기 위한 부지수시간알고리듬(subexponential time algorithm)의 미 발견으로 타원곡선 암호알고리듬은 암호학적으로 안전하다고 할 수 있다.

유한체를 $\mathbf{F}_q$로 나타내면, 타원곡선 암호에서는 $q=p\ \ (p:prime)$와 $q=2^m$ $(m:positive\ \ integer)$ 그리고 $q=p^m\ \ (p:prime,\ m:positive\ \ integer)$인 세 가지의 정의체 위에서 정의된 타원곡선이 사용된다.

각각의 정의체에서 정의된 타원곡선을 암호화에 적용하기 위해서서는 몇 가지 도메인 변수(domain parameters)가 필요하며, 이 도메인 변수는 안전성을 보장하기 위해 여러 가지 조건을 만족해야 한다. 이러한 도메인 변수에 대해 알아보기 위해 먼저 두 가지의 정의와 정리를 언급한다.

정의 2.1.5

타원곡선 $E(\mathbf{F}_q)$ 위의 점 P에 대하여 $rP=O$인 최소의 양의 정수 $r(\in \mathbf{F}_q)$을 점 P의 위수라고 한다.

정리 2.1.6 (Hesse의 정리)

$\#E(\mathbf{F}_q)$를 유한체 $\mathbf{F}_q$에서 정의된 타원곡선 $E(\mathbf{F}_q)$의 원소의 개수라고 하면, 다음을 만족한다.

$\#E(\mathbf{F}_q) = q+1-t$이면, $|t| \leqq 2\sqrt{q}$이다. $q = p^m$, $(p : prime)$

[증명] $E(\mathbf{F}_q)$의 원소의 개수는 유한체 $\mathbf{F}_q$의 원소의 중에 타원곡선 식을 만족하는 원소의 개수와 무한원점 O를 합한 개수이다. 즉, $\mathbf{F}_q$의 원소 x에 대응하는 y의 값이 존재하는 x의 개수를 찾는 문제이다. 이를 위해 $\chi : \mathbf{F}_q^* \rightarrow \{-1, 1\}$를

$$\chi(x) = \begin{cases} 1 : & x\text{가 } \mathbf{F}_q\text{에서 제곱근을 가질 때} \\ -1 : & otherwise \end{cases}$$

으로 정의하면, $\#\{y \in \boldsymbol{F}_q \mid y^2 = u\} = 1 + \chi(u)$이므로 타원곡선군의 점의 개수는

$$\#E(\boldsymbol{F}_q) = 1 + \sum_{x \in \boldsymbol{F}_q} (1 + \chi(x^3 + ax + b))$$

$$= 1 + q + \sum_{x \in \boldsymbol{F}_q} (\chi(x^3 + ax + b))$$

이다. 여기서 $|\sum_{x \in \boldsymbol{F}_q} \chi(x^3 + ax + b)|$는 체 $\mathbf{F}_q$에서 제곱근을 갖는 원소의 개수와 제곱근을 갖지 않는 원소의 개수의 차로 $|\sum_{x \in \boldsymbol{F}_q} \chi(x^3 + ax + b)| \leqq 2\sqrt{q}$ 이므로 정리가 성립한다.

이제 타원곡선 암호에 사용되는 각각의 정의체에 따른 도메인 변수와 조건에 대해 알아보면, <표 2-1>과 같다.

<표 2-1> 타원곡선의 정의체에 따른 도메인 변수와 조건[16]

$\mathbf{F}_q$ / domain parameter	$q = p$	$q = 2^m$	$q = p^m$
p	160비트 이상의 소수	-	3보다 큰 소수
m	-	소수를 권고,	소수를 권고,

		$2^{mB} \neq 1 \mod n$ $(1 \leq B \leq 30)$	$p^{mB} \neq 1 \mod n$ $(1 \leq B \leq 30)$
Weierstrass equation E	$y^2 = x^3 + ax + b$	$y^2 + xy = x^3 + ax^2 + b$	$y^2 = x^3 + ax + b$
$a,\ b \in \mathbf{F}_q$	$4a^3 + 27b^2 \neq 0 \mod p$	$b \neq 0$	$4a^3 + 27b^2 \neq 0$
기본점 G	$G \neq O$	$G \neq O$	$G \neq O$
G의 위수 n	160비트 이상의 소수, $n > 4\sqrt{p}$	160비트 이상의 소수, $n > 4\sqrt{(2^m)}$	160비트 이상의 소수, $n > 4\sqrt{(p^m)}$
$\#E(\mathbf{F}_q)$	$\#E(\mathbf{F}_p) \neq p$	$\#E(\mathbf{F}_{2^m}) \neq 2^m$	$\#E(\mathbf{F}_{p^m}) \neq p^m$
h	$h = \#E(\mathbf{F}_p)/n$	$h = \#E(\mathbf{F}_{2^m})/n$	$h = \#E(\mathbf{F}_{p^m})/n$
감산 다항식 $f(x)$	-	기저의 종류에 따라 생성	$f(x) = x^m - w$

<표 2-1>에서 m을 반드시 소수로 사용해야 하는 것은 아니지만, m을 합성수로 사용하는 경우 일부 곡선에 대해서는 Weil-Descent 공격에 대해 안전하지 않음이 밝혀졌다[16]. 따라서 m을 소수로 사용할 것을 권고하고 있다. 또한 $q = 2^m$과 $q = p^m$에서 각각 $2^{mB} \neq 1 \mod n$과 $p^{mB} \neq 1 \mod n$ $(1 \leq B \leq 30)$의 조건을 주는 것은 MOV공격을 피하기 위해서이다[16].

$q = 2^m$에서 감산 다항식 $f(x)$는 기저의 종류에 따라 삼항 다항식 기저(TPB)에서는 기약인 삼항 다항식 $f(x) = x^m + x^k + 1$의 형태이어야 하는데, 이때 기약성을 만족하는 삼항 다항식 중에서 k가 최소가 되도록 선택한다. 또한 오항 다항식 기저(PPB)는 삼항 다항식 기저를 사용할 수 없을 때, 기약인 오항 다항식 $f(x) = x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$의 형태를 사용한다. 이때 우선 k_3를 최소가 되도록 하고, 차례로 $k_2,\ k_1$이 최소가 되도록 선택한다.

<표 2-1>에서 언급한 기준은 최근까지의 여러 가지 공격에 대해 안전성을 보장하기 위함이다. 타원곡선 도메인 변수를 임의로 생성한 경우에는 <표 2-1>의 사항을 기준으로 반드시 검증을 거치는 것이 보안성에 신뢰를 줄 수 있을 것이다.

지금까지 타원곡선의 도메인 변수의 기준에 대해 알아보았다. 타원곡선 도메인 변수와 함께 타원곡선을 암호에 적용하기 위한 실제적인 타원곡선 이산대수 문제에 대해 알아보기 위해 예제를 통하여 설명하기로 한다. 실제 암호에서 사용하는 도메인 변수의 값이 너무 크므로 계산이 간단한 작은 유한체에서의 예제로 대신 한다.

예제 2.1.1

$\mathbf{F}_{23}$위의 타원곡선 $E: y^2 = x^3 + x + 19$와 그 위의 점 $P = (2,\ 11)$, $Q = (18, 2)$가 주어졌을 때, $Q = mP$인 정수 m을 찾는 것은 타원곡선 이산대수 문제이다. 여기서 몇 가지의 도메인 변수를 검증해 보면, $4 \times 1^3 + 27 \times 19^2 \equiv 22 \neq 0 \ \ mod\ 23$ 을 만족하고, P의 위수는 19이다. $19 \not> 4\sqrt{23}$이지만, 여기서는 19가 소수이므로 초특이 타원곡선이 아니라는 사실만 확인하고 공격에 대해서 고려하지 않기로 한다. 실제로 이 조건을 만족하기 위해서는 위수가 정의체 상에서의 기본점의 위수가 충분히 커야 된다. 이제 m을 찾는 과정을 보자. m을 찾는 과정은 P의 상수배를 통해서 이루어진다.

$1P = (2,\ 11)$ $2P = (4,\ 15)$ $3P = (21,\ 20)$ $4P = (18,\ 2)$ $5P = (7,\ 22)$
$6P = (17,\ 2)$ $7P = (20,\ 9)$ $8P = (3,\ 7)$ $9P = (11,\ 2)$ $10P = (11,\ 21)$
$11P = (3,\ 16)$ $2P = (20,\ 14)$ $13P = (17,\ 21)$ $14P = (7,\ 1)$ $15P = (18,\ 21)$
$16P = (21,\ 3)$ $17P = (4,\ 8)$ $18P = (2,\ 12)$ $19P = O$

P의 상수배를 통해 $4P = Q$ 즉, $m = 4$임을 알 수 있다. 본 예제에서는 작은 유한체에서 정의된 타원곡선을 이용하여서 쉽게 m의 값을 찾을 수 있었지만, 암호에 적용되는 정의체의 위수가 160비트를 넘게 되면, 위와 같은 m을 찾는 것은 거의 불가능 하다. 여기서 실제 암호체계의 기반이 되는 타원곡선 이산대수 문제는 Q, P가 주어졌을 때 $Q = mP$를 만족하는 m을 찾는 문제이다.

지금까지 유한체에서 정의된 타원곡선을 이용한 암호의 도메인 변수가 갖추어야 할 조건들과 간단한 예시로 암호화의 기본 개념인 타원곡선의 이산대수 문제가 어떠한 형태로 나타나는지 알아보았다. 실제로 암호학적으로 안전이 보장되는 크기의 유한체에서는 타원곡선군의 원소의 상수배가 전체 알고리듬의 수행속도에 핵심적인 영향을 미친다. 이런 상수배는 유한체에서의 연산이 바탕이 되므로, 유한체에서의 연산 알고리듬의 속도를 개선하는 것은 중요한 문제이다.

2.2. 타원곡선을 이용한 암호시스템

타원곡선을 이용한 암호시스템은 기존의 유한체에서의 이산대수 문제를 기반으로 하는 암호시스템을 타원곡선군의 이산대수 문제로 바꾸어 구현한 것이 많다. 여기서 타원곡선 이산대수문제가 어떤 형식으로 바뀌어 적용 되는지와 각 암

호시스템에 대해 어떤 순서로 개인키와 공개키가 생성되며, 전송되는지 등을 살펴보자.

2.2.1. ECDH(Elliptic Curve Diffie-Hellman)

ECDH는 DH(Diffie-Hellman) 알고리듬을 타원곡선군 위에 그대로 구현한 것으로 유한체 위의 DH 알고리듬에 비해 키 크기가 작고 여러 가지 공격에 강하다[2].

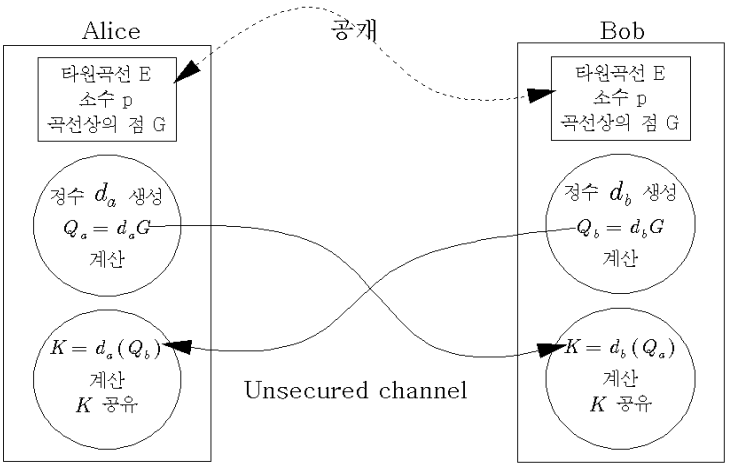
사용자 Alice와 Bob은 서로 동일한 타원곡선 $E(\mathbf{F}_q)$, 생성원 $G \in E(\mathbf{F}_q)$와 생성원의 위수 n을 미리 알고 있다고 가정한다. 그리고 다음의 과정을 거쳐 키 교환이 이루어진다.

단계1) Alice와 Bob은 각각 자신들의 개인키 $d_a, d_b \in \{2, \cdots, n-2\}$를 선택한다.

단계2) Alice와 Bob은 각각 공개키 $Q_a = d_aG$, $Q_b = d_bG$를 계산한다.

단계3) Alice와 Bob은 공개키를 교환한다.

단계4) Alice와 Bob은 각각 상대방의 공개키와 자신의 비밀키로 공유키(shared secret key) $K = d_ad_bG = (x_k, y_k)$를 공유할 수가 있다.



<그림 2-2> ECDH 키 교환 모델

일반적으로, 세션키는 공유키로부터 유도하게 되는데 K의 x, y 좌표를 모두 사용하지 않고 x 좌표인 $x_k \in \mathbf{F}_q$만을 사용하여 유도하게 된다. 이것은 타원곡선을 정의하는 Weierstrass 방정식으로부터 x_k가 결정되면 y의 이차방정식이 되므로 y_k의 부호를 제외하고는 x_k로부터 구할 수 있으므로, 세션키를 유도할 때 y_k를 포함하더라도 세션키의 안전도를 늘려주지는 않기 때문이다. ECDH는다양한 방식(Ephemeral-Ephemeral, Ephemeral-Static, Static-Static, Hybird 등)들이 각 응용(IPSec, S/MIME, WTLS 등)에 따라서 사용되고 있다[15].

2.2.2. EC-ElGamal

ElGamal 암호를 타원곡선 위에서 구현한 것으로 가장 널리 사용되는 타원곡선 암호이다[2]. 타원곡선 위의 ElGamal 암호는 Alice가 Bob에게 메시지 M을 비밀리에 전송하기 위한 암호화와 복호화의 과정이다.

사용자 Alice와 Bob은 타원곡선 $E(\mathbf{F}_q)$, 생성원 $G \in E(\mathbf{F}_q)$와 생성원의 위수 n을 미리 알고 있다고 가정하며, 이는 모두가 공통으로 사용한다. 그리고 다음의 과정을 거쳐 M을 전송한다.

단계1) Alice와 Bob은 각각 자신들의 개인키 $d_a, d_b \in \{2, \cdots, n-2\}$를 선택한다.
단계2) Alice와 Bob은 각각 공개키 $Q_a = d_aG$, $Q_b = d_bG$를 계산한다.
단계3) Alice와 Bob은 공개키를 교환한다.
단계4) Alice는 랜덤한 정수 k를 선택하고 $(kG, M + k(Q_b))$를 계산해 Bob에게 보낸다.
단계5) Bob은 $(M + kQ_b) - d_b(kG) = (M + k(d_bG)) - d_b(kG) = M$을 수행해 M을 얻는다.

표준화 자료에서는 $M + kQ_b$대신 M과 kQ_b의 x좌표를 $\mathbf{F}_q$상에서 곱한 값을 쓴다.

2.2.3. EC-Massey-Omura

Massey-Omura 암호를 타원곡선 위에서 구현한 것으로 Alice가 Bob에게 메

시지 M을 비밀리에 보내기 위한 암호화와 복호화 과정이다.

사용자 Alice와 Bob은 타원곡선 $E(\mathbf{F}_q)$, $E(\mathbf{F}_q)$의 위수 n을 미리 알고 있다고 가정하며, 이는 모두가 공통으로 사용한다. 그리고 다음의 과정을 거쳐 M을 전송한다.

단계1) Alice와 Bob은 각각 랜덤수 e_a, e_b를 선택한 후 $e_a d_a = 1 \mod n$, $e_b d_b = 1 \mod n$ 인 d_a, d_b를 구해 비밀키로 한다.

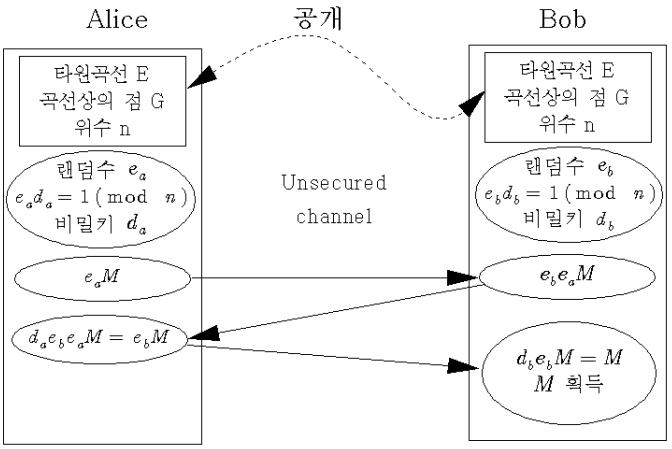
단계2) Alice는 Bob에게 $e_a M$을 보낸다.

단계3) Bob은 Alice에게 $e_b e_a M$을 보낸다.

단계4) Alice는 Bob에게 $d_a e_b e_a M = e_b M$을 보낸다.

단계5) Bob은 $d_b e_b M = M$을 계산해 M을 구한다.

이 시스템은 공개 정보는 단순하나 통신량이 많다는 단점을 갖고 있다[2].



<그림 2-3> EC-Massey-Omura 메시지 전송 모델

2.2.4. ECDSA(Elliptic Curve DSA)

ECDSA는 DSA를 타원곡선으로 변환한 것으로 $X9.62$로 표준화 되었다[15].

ECDSA의 과정은 키 생성 과정과 서명생성 과정, 그리고 서명검증 과정으로 나눌 수 있다. 이들 각각의 과정에 대해 알아보자.

[ECDSA 키 생성] Alice가 키를 생성한다고 가정하자.

단계1) $\boldsymbol{Z}_p$에서 정의된 타원곡선 E를 선택한다. $\#E(\boldsymbol{Z}_p)$는 큰 소수 n에 의해 나누어져야 한다.

단계2) 위수 n인 점 $P \in E(\boldsymbol{Z}_p)$를 선택한다.

단계3) 구간 $[2, n-2]$에서 통계적으로 유일하고 예측 불가능한 정수 d를 선택한다.

단계4) $Q = dP$를 계산한다.

단계5) Alice의 공개키는 (E, P, n, Q)이고, 비밀키는 d이다.

[ECDSA 서명 생성] 메시지 m에 A가 서명한다고 가정하자.

단계1) 구간 $[2, n-2]$에서 통계적으로 유일하고 예측 불가능한 정수 k를 선택한다.

단계2) $kP = (x_1, y_1)$과 $r = x_1 \pmod n$을 계산 한다.(x_1은 정수로 간주)

단계3) $r = 0$이면, step1)로 되돌아간다.

단계4) $k^{-1} \pmod n$을 계산한다.

단계5) $s = k^{-1}\{h(m) + dr\} \pmod n$을 계산한다.

단계6) 메시지 m에 대한 서명은 (r, s)이다.

[ECDSA 서명 검증] Bob이 Alice의 서명 (r, s)을 검증한다고 가정하자.

단계1) Alice의 인증된 공개키 (E, P, n, Q)를 얻는다.

단계2) r과 s가 구간 $[1, n-1]$에 있는 지 확인한다.

단계3) $w = s^{-1} \pmod n$과 $h(m)$을 계산한다.

단계4) $u_1 = h(m)w \pmod n$과 $u_2 = rw \pmod n$을 계산한다.

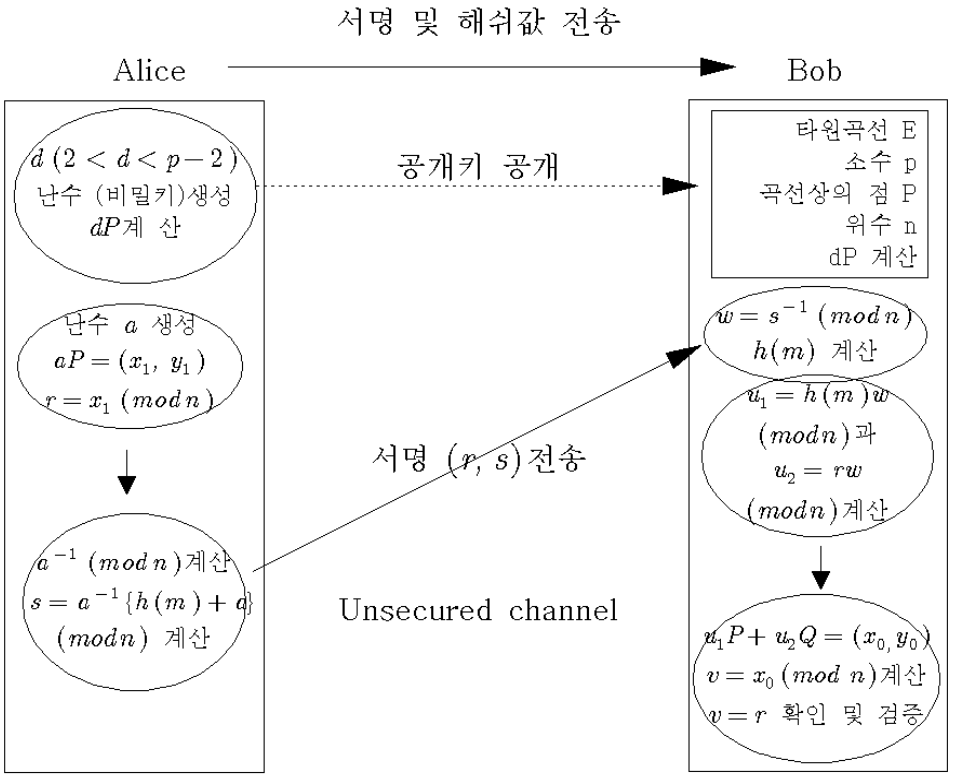
단계5) $u_1P + u_2Q = (x_0, y_0)$와 $v = x_0 \pmod n$을 계산한다.

단계6) $v = r$을 확인한다.

지금까지 ECDSA를 이용한 서명의 생성 및 검증에 관해 알아보았다. 이를 그림으로 표현하면 <그림 2-3>과 같다. ECDSA를 이용한 서명의 서명의 생성 및 검증은 기존의 DSA를 Elliptic Curve를 적용하여 구성한 것이지만, DSA에 비해 더욱 효율적이며, 빠른 알고리듬이다.

국내에서도 ECDSA와 같이 타원곡선을 이용한 전자서명 표준이 제정되었다.

이를 EC-KCDSA라 부르며, 이것의 기본 개념은 ECDSA와 거의 같다.



<그림 2-4> ECDSA를 이용한 전자서명 모델

2.3. 기존의 공개키 암호시스템과의 비교

공개키 암호 시스템에는 소인수분해 문제에 기반한 RSA, Rabin 등이 있으며, 이산대수 문제에 기반한 DSA, ElGamal 그리고 ECC 등이 있다. 여기서 가장 많이 사용되고 있는 RSA/DSA를 중심으로 기존의 공개키 암호시스템에 대해 간략히 알아본다. 그리고 기존의 공개키 방식인 DSA와 DH에 대하여 최근 주목받고 있는 ECDSA와 ECDH의 비교를 통해 ECC의 효율성에 대해 알아본다.

2.3.1. 기존의 공개키 암호시스템의 소개

기존의 공개키 암호시스템은 RSA와 DSA로 대변된다. RSA는 소인수분해 문제에 기반한 암호 알고리듬인 반면, DSA는 유한체에서의 이산대수 문제에 기반한 암호 알고리듬이다. 그러나 이 두 암호 알고리듬은 같은 수준의 보안성을 제공하며, 결과적으로 같은 수준의 키 길이를 갖는다. 그리고 RSA는 암호화 중심의 알고리듬인 반면, DSA는 전자서명을 위한 알고리듬이다. 여기서 이 두 가지의 공개키 암호 시스템에 대해 간략히 살펴본다.

• RSA

RSA는 1978년 Rivest, Shamir, Adleman에 의해 만들어진 것으로 현재 가장 널리 쓰이고 있는 공개키 암호 알고리듬이다[2]. RSA는 소인수분해 문제의 어려움에 기반한 것으로 다음의 Euler 정리가 그 기초가 된다.

정리 2.3.1 (Euler 정리)

주어진 정수 a, n이 서로 소(relatively prime)이면, $a^{\phi(n)} = 1 \ mod \ n$이 성립한다.

정리 2.3.1을 바탕으로 RSA의 암복호화 과정에 대해 살펴보자. RSA의 암복호화 과정은 사전 준비 단계와 암호화 그리고 복호화로 크게 나눌 수 있다. 여기서, B가 A에게 암호문을 전송하고 A는 B로부터 받은 암호문을 복호화하는 것으로 가정한다.

[사전 준비 단계]

단계1) 충분히 크고 크기가 비슷한 두 개의 소수 p와 q를 생성.

단계2) $n = pq$와 $\phi(n) = (p-1)(q-1)$을 계산.

단계3) $\gcd(\phi(n),\ e) = 1$인 랜덤한 정수 $e\ (1 < e < \phi(n))$를 선택.

단계4) 유클리드 알고리듬을 사용하여 $ed = 1 \ mod \ \phi(n)$인 $d\ (1 < d < \phi(n))$를 계산.

여기서 공개키는 $(n,\ e)$이며, 비밀키는 $(p,\ q,\ d)$이다.

[암호화 단계]

단계1) B는 A의 공개키 $(n,\ e)$를 수신.

단계2) 메시지 m을 $[0,\ n-1]$사이의 수로 표현.

단계3) $c = m^e \ mod \ n$을 계산.

단계4) 암호문 c를 A에게 전송.

[복호화 단계]

단계1) A는 자신의 비밀키 d를 이용하여,

$$c^d = m^{ed} = m^{1+k\phi(n)} = m(m^{\phi(n)})^k = m \ mod \ n$$

을 계산하여 m을 획득.

이상 RSA의 암복호화 과정에 대해 알아보았다. 여기서 $\lambda(n) = \mathrm{lcm}(p-1, \ q-1)$을 $\phi(n)$ 대신에 사용하기도 한다. 이 경우 d의 값이 작아져 복호화 과정의 효율을 높일 수 있으나 랜덤한 p, q에 대해서는 두 가지 경우의 값의 차이는 거의 없다[2].

• DSA

DSA는 전자서명 알고리듬으로 1991년 미국의 NIST에서 표준 디지털 서명 알고리즘 DSA를 제안하여, 해쉬함수 SHA-1과 함께 미국 연방 표준 FIPS-186이 되었다[2]. DSA는 키 생성 과정과, 서명 생성 과정 그리고 서명 검증 과정으로 나누어진다.

[DSA 키 생성]

단계1) $2^{159} < q < 2^{160}$인 소수 q 생성.

단계2) $q \mid (p-1)$이고 $2^{1023+64t} < p < 2^{1024+64t}$ $(0 \leqq t \leqq 8)$인 소수 p를 생성. (p가 1024비트인 경우)

단계3) $g \in \mathbf{Z}_p^*$를 선택하여 $g^{(p-1)/q} \neq 1 \ (mod \ p)$이면,

$\alpha = g^{(p-1)/q} \ (mod \ p)$를 계산.

단계4) 사용자 키 생성.

- 비밀키 : $0 < x < q$
- 공개키 : $y = \alpha^x \ (mod \ p)$

[DSA 서명 생성]

단계1) 난수 $0 < k < q$를 생성하여 $r = (\alpha^k \ (mod \ p)) \ (mod \ q)$을 계산.

단계2) 유클리드 알고리듬을 사용하여 $k^{-1} \ (mod \ q)$를 계산.

단계3) $s = k^{-1} \cdot (h(m) + xr) \ (mod \ q)$를 계산.

단계4) 메시지 m과 함께 서명 $(r,\ s)$를 서명 수신자에게 전송.

[DSA 서명 검증]

단계1) $0 < r, s < q$을 확인.

단계2) $w = s^{-1}\ (mod\ \ q)$와 $h(m)$을 계산.

단계3) $u_1 = w \cdot h(m)\ \ (mod\ \ q),\ u_2 = r \cdot w\ \ (mod\ \ q)$를 계산.

단계4) $v = (\alpha^{u_1} \cdot y^{u_2}\ \ (mod\ \ p))\ \ (mod\ \ q)$를 계산.

단계5) $v = r$이면 서명을 받아들임.

유한체에서의 이산대수 문제에 기반한 DSA는 타원곡선상의 이산대수 문제에 기반한 ECDSA나 EC-KCDSA 등과 기본적인 개념이 거의 같다.

2.3.2. ECC와 기존 암호시스템과의 비교

<표 2.2>에서 타원곡선 이산로그문제는 인수분해문제나 유한체의 이산로그문제와는 달리 현재까지 준지수식 시간을 갖는 공격 알고리즘이 발견되지 않았다. 일반적인 타원곡선 이산로그 문제에 대한 공격으로는 sqauare root algorithm인 Pollard-ρ 알고리듬에 바탕을 둔 방식이 현재까지 알려진 가장 좋은 알고리듬이다.[15]

<표 2-2> 유한체와 타원곡선의 비교

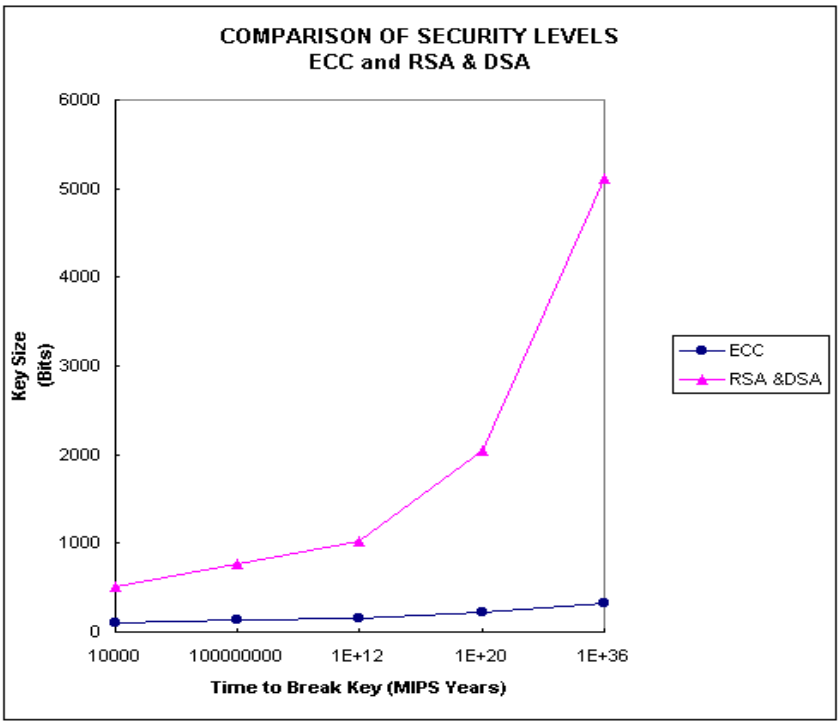
	유한체의 곱셈군 $(\mathbf{F}_p^{\times})$	타원곡선의 덧셈군 $(E(\mathbf{F}_p))$
군의 원소 생성원 위수 연산 개인키 공개키	정수 $\{1,\ \cdots,\ p-1\}$ g q, i.e. $g^q \mod p = 1$ 모듈라 곱셈 $x \in \{2,\ \cdots,\ q-1\}$ $y = g^x \mod p$	방정식을 만족하는 $(x,\ y)$들의 집합 $\cup\ \{O\}$ G n, i.e. $nG = O$ 타원곡선상의 덧셈 $d \in \{2,\ \cdots,\ n-2\}$ $Q = dG = G + \cdots + G\ (d \text{ times})$
이산로그	주어진 $(g,\ y)$로부터 $y = g^x \mod p$를 만족하는 x를 구하는 문제	주어진 $(G,\ Q)$로부터 $Q = dG$를 만족하는 d를 구하는 문제

Pollard-ρ 공격이 square root 공격이므로 일반적인 타원곡선의 안전도는 정의되어 있는 유한체의 크기의 절반정도로 추정한다. 즉, 80 비트의 안전도를 주기 위해서는 약 160비트 이상의 유한체에서 정의된 타원곡선을 사용해야 한다. 이것은 비슷한 안전도를 주기 위해 RSA에서 1024 비트의 합성수 n을 사용해야 하거나, Diffie-Hellman/DSA에서 1024 비트의 소수 p를 사용해야 하는 것과 비교해 볼때 약 1/7밖에 되지 않는다<표 2-3>.

<표 2-3> 같은 안전도에 따른 도메인 변수의 크기 비교[24]
(RSA : 공개키 n, DSA : p, 타원곡선 : 정의체)

Time to break in MIPS year	RSA/DSA (bits)	ECC (bits)	RSA vs. ECC (key size ratio)
10^4	512	106	5:1
10^8	768	132	6:1
10^{12}	1024	160	7:1
10^{20}	2048	210	10:1
10^{36}	5120	320	17:1
10^{168}	120000	1200	100:1

<그림 2-5>에서와 같이 안전도의 증가에 따른 키 크기의 증가가 RSA/DSA는 급격히 증가하는 방면 ECC는 거의 증가하지 않는 것을 볼 수 있다.



<그림 2-5> ECC와 RSA/DSA의 안전도 수준 비교[24]

타원곡선 암호는 일반적으로 RSA/DSA 등에 비해 좋은 보안효율을 갖는다. 그러나 이것은 어디까지나 지금까지 알려진 공격에 대한 안전도라는 것과 이런 알려진 공격에 대해 여러 조건들을 고려하여 암호 시스템이 설계될 때 보안효율이 유지되는 것이다.

일반적인 타원곡선이 아닌 특별한 형태의 타원곡선에 대해서는 공격방법이 있는 것도 있으므로 이러한 형태의 타원곡선을 암호 시스템에 사용하는 것은 위험하다. 안전하지 않은 타원곡선으로는 초특이곡선, Frobenius 사상의 trace가 2인 곡선, 그리고 $\mathbf{F}_p$에서 정의된 비정규 곡선 등이 있다.

DH, ECDH, DSA, ECDSA를 구현한 결과를 비교한 자료를 살펴보자<표 2-4>. 구현환경은 Petium III (450 MHz)에서 C 언어로 프로그램한 것을 MSVC++6.0으로 컴파일한 결과이다[15]. 그리고 각 알고리듬에 적용한 도메인 변수들은 다음과 같다.

• DSA

Random 변수 : 1024 비트의 소수 p와 $p-1$을 나누는 160 비트의 소수 q를

임의로 생성.

• DH

Oakley group #2 : 1024 비트의 소수 p, exponent의 크기는 160비트로 제한.

• ECDH/ECDSA

Oakley group #6 : $\mathbf{F}_{2^{163}}$에서 정의된 일반적인 곡선.

Oakley group #7 : $\mathbf{F}_{2^{163}}$에서 정의된 Koblitz 곡선.

WTLS 곡선 #7 : 160비트 소수 p에 대한 유한체 $\mathbf{F}_p$위에서 정의된 곡선.

<표 2-4> DH, ECDH, DSA, ECDSA 알고리듬의 속도 비교[15]

(단위 : msec)

		ModP group		EC2N group		ECP group
		DSA(random)	O #2	O #6	O #7	WTLS #7
도메인 변수		$\|p\| = 1024$	$\|p\| = 1024$	$m = 163$	$m = 163$	$\|p\| = 160$
공개키/개인키(비트)		1024/160	1024/160	326/163	326/163	320/160
DH	phase1	-	1.788	-	-	-
	phase2	-	8.148	-	-	-
ECDH	phase1	-	-	0.776	0.735	0.516
	phase2	-	-	2.351	1.216	1.695
DSA	서명	1.931	-	-	-	-
	검증	9.434		-	-	-
ECDSA	서명	-	-	0.899	0.832	0.603
	검증	-	-	3.188	1.923	2.009

DH와 ECDH는 phase 1,2로 구분할 수 있는데, phase 1은 자신의 개인키를 임의로 생성하여 자신의 공개키를 계산하는 과정($y = g^x$, $Q = dG$)이고, phase 2는 상대방의 공개키와 자신의 개인키로부터 shared secret을 계산하는 과정($K = y_b^x$, $K = dQ_b$)이다. 여기서 ECDH는 phase1의 경우 약 2.3~3.5배, phase2의 경우 약 3.5~6.7배 정도 DH보다 빠른 수행속도를 보였다.

DSA와 ECDSA의 경우 키 길이를 비교해 보면, 개인키는 모두 160 비트 정도로 비슷하지만 공개키는 ECDSA는 320 비트로 DSA의 공개키 1024 비트의 약

1/3 수준이다. 또한 수행시간에서도 서명시간은 약 2.1~3.2배, 검증시간은 약 3.0~4.9배 ECDSA가 빠른 것으로 나타났다.

ECDH와 ECDSA를 구현하는데 필요한 타원곡선으로 Oakley group #7과 같은 Koblitz 곡선의 경우 일반적인 곡선보다 ECDH의 phase2나 ECDSA의 검증에서 대략 2배 정도 빠른 수행 결과를 나타낸다. 이와 같은 장점으로 Koblitz 곡선은 IPsec, FIPS186-2, X9.62, SECG, WTLS 등에서 추천 곡선으로 포함하고 있다[15]. 실제로 Koblitz 곡선은 일반적인 곡선을 사용할 때 보다 보안성이 감소하는 것으로 나타나고 있다. 그러나 국내 표준화 자료에서는 보안성의 감소에 대해 현실적으로 문제가 되지 않는다고 밝히고 있다[16].

2.4. ECC 관련 동향 및 전망

본 절에서는 타원곡선 암호시스템과 관련된 여러 국제 표준이나 국내 표준에 대해 현재의 표준화 동향에 대해 알아보며, 표준화에 따라 타원곡선 암호시스템을 적용하기 위한 여러 분야의 시장에 대한 전망을 알아본다.

2.4.1. ECC의 표준화 관련 동향

표준화는 여러 가지 면에서 장점이 있지만, 그중 중요한 3가지 면에서 장점을 말할 수 있다. 첫째로, 다양한 환경의 다른 H/W와 S/W사이의 상호 연동성을 보장하고, 둘째로, 암호학적 관점에서 그 시스템의 안전도를 면밀히 검토하여 재정되므로 신뢰할 수 있으며, 마지막으로 다양하고 폭 넓은 환경에서 시스템을 구현하는 설계자들에게 암호시스템을 설계할 수 있도록 도와준다.

ECC는 현재 세계 여러 표준화 기관에서 표준화가 되었으며, 국내에서도 최근 ECC를 적용한 인증서기반 전자서명이 표준화 되었다. 이러한 여러 가지 현황에 대해 살펴보면, 타원곡선 암호를 지원하는 대표적인 국제 표준으로 IEEE(Institute of Electrical and Electronics Engineers) P1363 working group에서 표준화한 P1363과 현재 표준화 중인 P1363a(Draft ver.9)가 있으며[18], ISO(Interna tional Standards Organization)에서는 ISO/IEC 15946-1, ISO/IEC 15946-2, ISO/IEC 15946-3, ISO/IEC FCD 15946-4 의 각 부분으로 분리하여 현재 표준화 중이다[17]. ISO/IEC 15946 은 “Cryptographic Techniques Based on Elliptic Curves” 이다[17]. ANSI(America National Standards Institute)에서는 타원곡선 전자서명알고리듬(ECDSA, Elliptic Curve Digital

Signature Algori thm)인 X9.62와 타원곡선 암호를 이용한 키 합의 및 키 전송에 관한 내용인 X9.63이 표준화 되었다[20]. SECG(Standards for Efficient Cryptography Group)에서는 SEC 1과 SEC 2에서 각각 타원곡선 암호에 관한 일반적인 내용과 타원곡선의 도메인 변수에 대한 내용을 담고있다[19]. FIPS(Federal Information Processing Standards)에서는 미 연방 서명 표준인 FIPS 180에서 DSA 서명만을 규정하고 있던 것을 개정을 통하여 FIPS 186-2에서는 X9.62의 ECDSA를 포함시키고 있다[21]. RSA Security의 PKCS(Public-Key Cryptography Standards)에서는 타원곡선 암호 표준을 PKCS #13(the elliptic curve cryptography standard)으로 표준화 중에 있다[23]. 여기에는 전자서명, 키 생성, 도메인 변수, 공개키 암호화, 키 합의 등의 내용을 포함하고 있다. 국내에서는 한국정보통신기술협회(TTA : Telecommunications Technology Association)에서 타원곡선을 이용한 전자서명 알고리듬인 EC-KCDSA(Korean Certificate-based DSA using Elliptic Curve s)에 대한 내용을 담고 있는 TTAS.KO-12.0015를 2001년 12월 표준화 완료하였다[22].

지금까지와 같은 표준들 외에도 타원곡선 암호를 지원하는 응용 보안 표준들이 있는데, IPSec, TLS/WTLS, S/MIME, PKIX 등이 그것이다. IPSec(Internet Protocol Security protocol)은 데이터 송신자의 인증을 허용하는 인증 헤더(AH:Authentication Header)와 송신자의 인증 및 데이터 암호화를 함께 지원하는 ESP (Encapsulating Security Payload) 등, 두 종류의 보안 서비스를 제공한다. TLS(Transport Layer Security)는 두 개의 계층으로 나누어 지는데, 데이터의 암호화와 연결에 대해 보장하는 TLS Record Protocol 계층, 그리고 서버와 클라이언트 사이의 인증과 암호화 알고리듬에 대한 협상등의 내용을 갖는 TLS Handshake Protocol 계층이 있다. WTLS(Wireless Transport Layer Security)는 WAP(Wireless Application Protocol)의 보안 계층으로 데이터 무결성과 인증을 제공한다. S/MIME(Secure/MIME)는 MIME(Multipurpose Internet Mail Extensions)의 새로운 버전으로 메시지의 암호화를 제공한다. S/MIME는 RSA의 암호화를 기초로 하고 있지만, RFC 3278 등에 타원곡선을 이용한 암호화의 내용이 포함되어 있다. PKIX(Public-Key Infrastructure based X.509)는 공개키 기반구조의 표준인 X.509를 기초로 하는 내용을 담고 있다.

지금까지 표준화 중이거나 표준으로 제정된 국ㆍ내외 표준화 현황에 대해 알아 보았다. 이 외에도 많은 표준들이 세계의 여러 표준 기관들로부터 제정되고 있다. 표준화가 활발하게 이루어 지고 있는 상황과 함께 타원곡선을 이용한 암호의 적용도 다양한 부분에서 활발하게 이루어지고 있다. 특히, 무선과 같은 특수한 환경에서 적용 사례가 많으며, 기존의 유선 인터넷 상에서도 보안시스템의 효율을 높이기 위해 많은 적용이 이루어 지고 있다. 이와 같이 적용은 향후에 무선

과 유선이 통합되는 환경을 고려하여 보면 앞으로의 타원곡선 암호의 적용 분야는 급격히 증가할 것으로 기대된다. 이러한 적용에 대한 동향과 전망에 대해 다음 절에서 살펴보고자 한다.

2.4.2. ECC의 적용 분야에 대한 동향 및 전망

IMT-2000 등 유·무선 통합화의 가속화에 따라 무선 인터넷은 차세대 정보통신 산업으로 주목받고 있다. 이런 추세와 함께 정보통신부에서는 '무선인터넷 활성화 정책방향(2000.6.)' 및 'M-Commerce 활성화 정책(2001.1)' 등을 수립하여 추진하고 있으며, 이동통신 업계와 관련 업체들은 무선인터넷 시장의 활성화에 주도적 역할을 하고 있다[12]. 이런 상황에서 무선의 보안 문제와 유·무선 통합화에 따른 보안의 문제는 아주 중요한 사안이다. 이와 함께 보안의 중심에 있는 것이 바로 ECC이다. ECC의 적용 분야에 대한 직접적인 조사에 대한 보고는 없으나, ECC의 적용은 무선인터넷, M-Commerce, 보안시장 등의 전망에 부합한다고 할 수 있다.

무선인터넷 사용자 전망은 <표 2-5>에서와 같이 빠른 성장세를 보이고 있다.

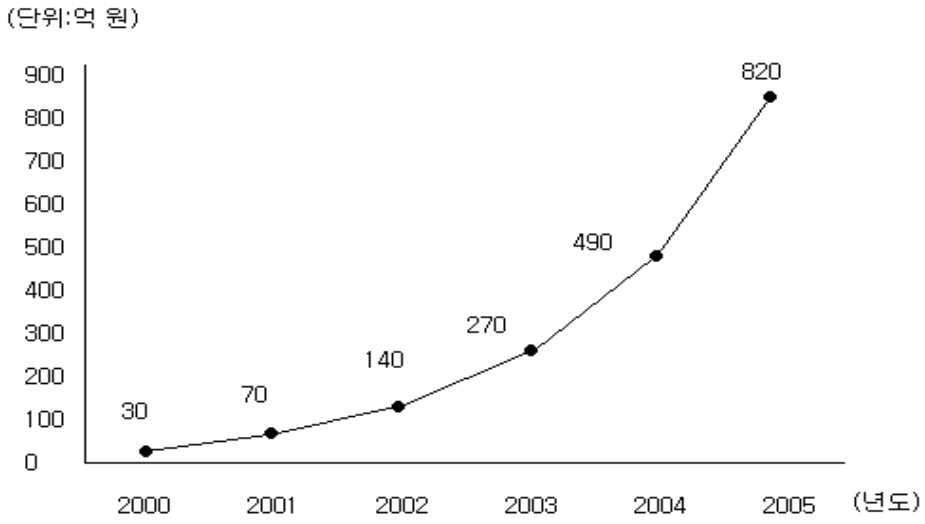
<표 2-5> 전세계 무선인터넷 사용자 전망[5]

(단위 : 백만 명)

구 분	2000년	2001년	2002년	2003년	2004년	2006년
미국	1.9	9.8	32.3	71.9	127.4	167.1
일본	54.0	62.6	80.7	99.2	108.4	115.8
아시아/태평양	8.4	18.6	39.9	77.9	145.9	265.8
서유럽	31.0	93.8	156.4	227.4	290.7	326.0
기타	0.7	6.1	19.2	50.9	101.8	173.8
총 계	96.0	190.9	328.4	527.2	774.2	1,048.5

실제로 과거에 무선인터넷이 SMS 등의 서비스에 집중되었지만, 점차 다양한 부분으로 확산되고 있다. 그 중심에 M-Commerce 가 있다. M-Commerce는 현재의 망개방이 완전히 이루어지지 않은 상태에서 각 이동통신사의 가입자를 중심으로 서비스가 이루어지고 있다. 그러나 향후 1~2년 사이에 망개방이 본격적으로 진행되면, 서로 다른 통신사가 망을 공유하여 이용할 수 있게 되고, 또한 여러 CP(Contents Providor)들이 각종 서비스를 제공하여 무선전자 상거래가 더욱 활성화 될 것이다.

무선을 통한 전자상거래 시장은 지금의 2~6배 정도의 급격한 성장을 할 것으로 전망되고 있다<그림 2-6>.



<그림 2-6> 국내 Mobile Commerce 시장 규모 전망[13]

M-Commerce는 무선네트워크를 통해 금전적 재화의 거래와 개인 정보의 전송 등의 보안이 요구되는 내용이 포함된다. 따라서 보안의 문제를 해결하기 위해 무선단말기와 무선네트워크에 적합한 방식으로 ECC를 적용한 보안이 각광받고 있는 것이다[10][11]. 여기서 세계와 국내의 정보보호 시장의 동향과 전망에 대해 알아보면, 세계 정보보호 시장은 2001년 168억 달러를 넘어선 것으로 파악되며, 연평균 성장률(CAGR)은 28.8%로 성장하여 2007년 766억 달러에 이를 것으로 전망되고 있다<표 2-6>. 바이러스 백신, 방화벽 등에 의한 1차적 정보보호제품의 시장은 이미 성숙단계에 들어섰으며, VPN, PKI, 컨텐츠보안 등의 시장이 크게 성장할 것으로 예상되고 있다. 이들 제품은 2007년까지 각각 53.8%, 42.7%, 59.7%의 성장률을 보일 것으로 전망된다.

<표 2-6> 세계 정보보호 시장 동향 및 전망[9]

(단위 : 백만 달러)

분 류		2001	2002	2003	2004	2005	2006	2007	CAGR
제품	바이러스백신	1,779	2,148	2,573	3,062	3,606	4,245	4,997	18.8%
	암호	255	305	362	436	525	632	760	19.6%
	인증	790	950	1,235	1,564	1,935	2,360	2,878	24.0%
	침입차단시스템	1,500	2,017	2,651	3,390	4,198	5,195	6,428	27.4%
	IDS	303	388	489	601	733	892	1,085	23.7%
	VPN	654	1,082	1,732	2,658	3,949	5,850	8,666	53.8%
	컨텐츠 보안	249	456	742	1,130	1,551	2,530	4,126	59.7%
	PKI	678	1,037	1,467	1,994	2,637	3,885	5,723	42.7%
	보안관리도구	1,745	2,050	2,520	3,010	3,615	4,248	4,991	21.9%
	생체인식	580	748	1,014	1,372	1,857	2,745	3,568	35.4%
	소 계	8,533	11,181	14,785	19,217	24,606	32,582	43,222	30.8%
서비스	인증서비스	1,812	2,290	2,860	3,750	4,578	5,711	7,124	25.6%
	보안관제	1,526	1,902	2,382	2,970	3,722	4,640	5,784	24.9%
	정보보호컨설팅	2,401	2,960	3,627	4,621	5,785	7,010	8,494	23.4%
	소 계	5,739	7,152	8,869	11,341	14,085	17,361	21,402	24.5%
기타		2,532	3,285	4,270	5,554	7,118	9,230	11,968	29.5%
합 계		16,804	21,618	27,924	36,112	45,809	59,173	76,592	28.8%

국내 정보보호시장은 2001년 3,755억 원 규모에 이른 것으로 파악되며 CAGR은 36.2%로 성장하여, 2007년에는 2조 4,000억 원 규모에 이를 것으로 전망되고 있다. 또한 세계 정보보호 시장에서 차지하는 국내 정보보호 시장의 비율은 2001년에는 1.4%에서 2007에는 1.7%로 CAGR은 4.0%로 전망된다<표2 -7>.

<표 2-7> 국내 정보보호 시장 동향 및 전망[9]

(단위 : 억 원)

분 류		2001	2002	2003	2004	2005	2006	2007	CAGR
제품	바이러스백신	297	372	509	620	635	711	796	17.9%
	암호	112	134	158	190	228	260	296	17.6%
	인증	345	412	520	655	839	1015	1,227	23.6%
	침입차단시스템	585	700	810	1058	1,310	1,580	1,905	21.8%
	IDS	386	546	705	810	1,050	1,260	1512	25.6%
	VPN	285	365	575	921	1,228	1,450	1,712	34.9%
	컨텐츠 보안	209	332	451	668	1,021	1,322	1,711	42.0%
	PKI	382	490	840	1,205	1,721	2,350	3,208	42.6%
	보안관리도구	129	170	235	345	522	730	1,020	41.2%
	생체인식	800	1,200	2,200	3,300	4,700	6,500	8,500	41.2%
	소 계	3,530	4,721	7,003	9,772	13,254	17,178	21,887	35.5%
서비스	인증서비스	40	49	62	85	122	186	283	38.6%
	보안관제	56	84	138	190	266	365	492	43.7%
	정보보호컨설팅	95	142	250	335	523	690	910	45.8%
	소 계	191	275	450	610	911	1,241	1,685	43.8%
기타		34	56	110	195	280	320	365	48.5%
합 계		3,755	5,052	7,563	10,577	14,445	18,739	23,937	36.2%

Ⅲ. ECC의 핵심 연산 알고리듬 비교 및 분석

본 장에서는 타원곡선 알고리듬의 핵심이 되는 연산 알고리듬에 대해 알아본다. 타원곡선에서 연산의 기본은 타원곡선이 정의된 정의체의 연산에 있다. 따라서 정의체의 연산 알고리듬에 대해 살펴본 후 타원곡선군에서의 점(point)의 상수(scalar)배에 대한 알고리듬을 살펴본다. 타원곡선군에서의 점의 상수배는 타원곡선 이산대수문제를 만들어내기 위한 기본적인 연산이다. 따라서 타원곡선 암호시스템에서는 가장 중요한 부분이라고 볼 수 있다. 이런 상수배 연산이 효율적으로 이루어지지 않으면, 제한된 시스템에서의 보안효율이나 수행속도가 떨어지게 된다. 따라서 이러한 연구는 타원곡선 암호시스템에서 가장 중요한 문제이다 [3][4][7].

3.1. 유한체에서의 연산알고리듬

유한체에서의 연산은 타원곡선 상에서의 연산의 기본이 된다. 따라서 유한체에서의 연산을 개선함으로 해서 타원곡선상의 연산에 효율성을 증가시킬 수 있다. 유한체의 연산은 곱셈(Multiplication)과 감소(Reduction), 제곱(Square), 역원(Inversion) 으로 구분하여 생각할 수 있다. 여기서 감소는 일반적인 법(modular)연산 이지만, 유한체 $\mathbf{F}_{2^m}$ 와 같이 다항식의 형태를 취하는 유한체에서는 기약인 m차 감산다항식을 법의 형태로 사용한다. 여기서는 $GF(2^m)$상에서의 각각의 연산에 대해 제시된 알고리듬을 위주로 살펴본다.

- $GF(2^m)$상의 곱셈(Multiplication)

Left-to-Right comb method with windows of width w=4

입력 : 최대 $m-1$차를 가지는 이진 다항식 $a(x)$와 $b(x)$.

출력 : $c(x) = a(x) \cdot b(x)$.

과정 : $t = \lfloor m/n \rfloor$, $s = n \cdot t - m$, $l = n/4$.

단계1) 최대 3차가 되는 모든 다항식 $u(x)$에 대해서 $B_u = u(x) \cdot b(x)$를 계산.

단계2) $C \leftarrow 0$.

단계3) For k from $l-1$ to 0 by -1 do:

단계3.1) For j from 0 to $t-1$ do:

Let $u=(u_3,\ u_2,\ u_1,\ u_0)$. $C\{j\} \leftarrow B_u + C\{j\}$.

단계3.2) If $k \neq 0$ then $C \leftarrow C \cdot x^4$.

단계4) Return C.

- $GF(2^m)$상의 감소(Reduction)

Modular reduction (one bit at a time)

입력 : 최대 $2m-2$의 차수를 가지는 이진 다항식 $c(x)$.

출력 : $c(x)\ \ mod\ f(x)$.

과정 :

사전계산

단계1) $u_k = x^k r(x),\ 0 \leqq k \leqq 31$.

주요루프

단계2) For i from $2m-2$ to m by -1 do:

단계2.1) If $c_i = 1$ then do:

$j = \lfloor (i-m)/32 \rfloor,\ k = (i-m) - 32j$.

$C\{j\} \leftarrow u_k + C\{j\}$.

단계3) Return $(C[t-1], \cdots,\ C[1],\ C[0])$.

Modular reduction (one word at a time)

입력 : 최대 324의 차수를 가지는 이진 다항식 $c(x)$.

출력 : $c(x)\ \ mod\ f(x),\ f(x) = x^{163} + x^7 + x^6 + x^3 + 1$.

과정 :

단계1) For i from 10 to 6 by -1 do:

단계1.1) $T[i] \leftarrow C[i]$.

$C[i-6] \leftarrow C[i-6] \oplus (T \ll 29)$.

$C[i-5] \leftarrow C[i-5] \oplus (T \ll 4) \oplus (T \ll 3) \oplus T \oplus (T \gg 4)$.

$C[i-4] \leftarrow C[i-4] \oplus (T \gg 28) \oplus (T \gg 29)$.

단계2) $T \leftarrow C[5]$ AND 0xFFFFFFF8.

단계3) $C[0] \leftarrow C[0] \oplus (T \ll 4) \oplus (T \ll 3) \oplus T \oplus (T \gg 3)$.

단계4) $C[1] \leftarrow C[1] \oplus (T \gg 28) \oplus (T \gg 29)$.

단계5) $C[5] \leftarrow C[5]$ AND 0x00000007.

단계6) Return $(C[5], C[4], C[3], C[2], C[1], C[0])$.

- $GF(2^m)$상의 제곱(Squaring)

"0" padding에 의한 제곱

입력 : $a \in GF(2^m)$.

출력 : $a^2 \ mod \ f(x)$.

과정 :

사전계산

단계1) 각 바이트 $v = (v_7, \cdots, v_1, v_0)$은 16-비트 양자화.

$T(v) = (0, v_7, \cdots, 0, v_1, 0, v_0)$.

주요계산

단계2) For i from 0 to $t-1$ do:

단계2.1) $A[i] = (u_3, u_2, u_1, u_0)$, (여기서 u_j는 바이트이다).

단계2.2) $C[2i] \leftarrow (T(u_1), T(u_0))$,

$C[2i+1] \leftarrow (T(u_3), T(u_2))$.

단계3) $b(x) = c(x) \ mod \ f(x)$.

단계4) Return b.

- $GF(2^m)$상의 역원(Inversion)

EEA(Extended Euclidean Algorithm)

입력 : $a \in GF(2^m), \ a \neq 0$.

출력 : $a^{-1} \ mod \ f(x)$.

과정 :

단계1) $b \leftarrow 1, \ c \leftarrow 0, \ u \leftarrow a, \ v \leftarrow f$.

단계2) While $\deg(u) \neq 0$ do:

단계2.1) $j \leftarrow \deg(u) - \deg(v)$.

단계2.2) If $j < 0$ then: $u \leftrightarrow v, \ b \leftrightarrow c, \ j \leftrightarrow -j$.

단계2.3) $u \leftarrow u + x^j v,\ b \leftarrow b + x^j c.$

단계3) Return b.

MAIA(Modified Almost Inverse Algorithm)

입력 : $a \in GF(2^m),\ a \neq 0.$

출력 : $a^{-1}\ mod\ f(x).$

과정 :

단계1) $b \leftarrow 1,\ c \leftarrow 0,\ u \leftarrow a,\ v \leftarrow f.$

단계2) While x divides u do:

단계2.1) $u \leftarrow u/x.$

단계2.2) If x divides b then $b \leftarrow b/x$; else $b \leftarrow (b+f)/x.$

단계3) If $u = 1$ then return b.

단계4) If $\deg(u) < \deg(v)$ then: $u \leftrightarrow v,\ b \leftrightarrow c.$

단계5) $u \leftarrow u + v,\ b \leftarrow b + c.$

단계6) 단계 2로 간다.

3.2. 타원곡선군에서의 연산알고리듬

이진 방법(Binary method)

입력 : 점 P, l-bit 정수 $k = \sum_{j=0}^{l-1} k_j 2^j,\ k_j \in \{0,\ 1\}.$

출력 : $Q = [k]P.$

과정 :

단계1) $Q \leftarrow O.$

단계2) For $j = l-1$ to 0 by -1 do:

단계2.1) $Q \leftarrow [2]Q.$

단계2.2) If $k_j = 1$ then $Q \leftarrow Q + P.$

단계3) Return Q.

m진 방법(m-ary method)

입력 : 점 P, 정수 $k=\sum_{j=0}^{d-1} k_j m^j,\ k_j \in \{0,\ 1, \cdots,\ m-1\}$.

출력 : $Q=[k]P$.

과정 :

사전계산

단계1) $P_1 \leftarrow P$.

단계2) For $i=2$ to $m-1$ do:

$P_i \leftarrow P_{i-1}+P$. $(P_i=[i]P)$

단계3) $Q \leftarrow O$.

주요루프

단계4) For $j=d-1$ to 0 by -1 do:

단계4.1) $Q \leftarrow [m]Q$.

단계4.2) $Q \leftarrow Q+P_{k_j}$.

단계5) Return Q.

변형된 m진 방법(Modified m-ary method)

입력 : 점 P, 정수 $k=\sum_{j=0}^{d-1} k_j m^j,\ k_j \in \{0,\ 1, \cdots,\ m-1\}$.

출력 : $Q=[k]P$.

과정 :

사전계산

단계1) $P_1 \leftarrow P$.

단계2) For $i=1$ to $(m-2)/2$ do:

$P_{2i+1} \leftarrow P_{2i-1}+P_2$.

단계3) $Q \leftarrow O$.

주요루프

단계4) For $j=d-1$ to 0 by -1 do:

단계4.1) If $k_j \neq 0$ then do:

단계4.1.1) Let $s_j,\ h_j$ be such that $k_j=2^{s_j}h_j,\ h_j$ odd.

단계4.1.2) $Q \leftarrow [2^{r-s_j}]Q$.

단계4.1.3) $Q \leftarrow Q+P_{k_j}$.

단계4.2) Else $s_j \leftarrow r$.

단계4.3) $Q \leftarrow [2^{s_j}]Q$.

단계5) Return Q.

Sliding window method

입력 : 점 P, 정수 $k = \sum_{j=0}^{l-1} k_j 2^j,\ k_j \in \{0,\ 1\}$.

출력 : $Q = [k]P$.

과정 :

사전계산

단계1) $P_1 \leftarrow P,\ P_2 \leftarrow [2]P$.

단계2) For $i = 1$ to $2^{w-1} - 1$ do $P_{2i+1} \leftarrow P_{2i-1} + P_2$.

단계3) $j \leftarrow l-1,\ Q \leftarrow O$.

주요루프

단계4) While $j \geqq 0$ do:

단계4.1) If $k_j = 0$ then $Q \leftarrow [2]Q,\ j \leftarrow j-1$.

단계4.2) Else do:

단계4.2.1) Let t be the least integer such that $j \leftarrow t+1 \leqq w$ and $k_t = 1$.

단계4.2.2) $h_j \leftarrow (k_j k_{j-1} \cdots k_t)_2$.

단계4.2.3) $Q \leftarrow [2^{j-t+1}]Q + P_{k_j}$.

단계4.2.4) $j \leftarrow t-1$.

단계5) Return Q.

SD(Signed digital) representations

입력 : 정수 $k = \sum_{j=0}^{l-1} k_j 2^j,\ k_j \in \{0,\ 1\}$.

출력 : $NAF(k),\ k = \sum_{j=0}^{l} s_j 2^j,\ s_j \in \{-1,\ 0,\ 1\}$.

과정 :

단계1) $c_0 \leftarrow 0$.

단계2) For $j=0$ to do:

단계2.1) $c_{j+1} \leftarrow \lfloor (k_{j+1}+k_j+c_j)/2 \rfloor$ (assume $k_i=0$ for $i \geqq l$).

단계2.2) $s_j \leftarrow k_j+c_j-2c_{j+1}$.

단계3) Return $(s_l\, s_{l-1} \cdots s_0)$.

Binary NAF(Non-Adjacent Form) method

입력 : 점 P, 정수 n.

출력 : $Q=nP$.

과정 : $S=NAF(n)$.

단계1) $Q \leftarrow P$.

단계2) For i from $l-2$ to 0 do:

단계2.1) Set $\leftarrow 2Q$.

단계2.2) If $e_i=1$ then $Q \leftarrow Q+P$.

단계2.3) If $e_i=-1$ then $Q \leftarrow Q-P$.

단계3) Return Q.

Non-redundant signed m-ary representation algorithm

입력 : 정수 $k=\sum_{j=0}^{l} k_j 2^j,\ k_j \in \{0,\ 1\},\ k_l=0$.

출력 : 쌍 $\{(b_i,\ e_i)\}_{i=0}^{d-1}$의 순열.

과정 :

단계1) $d \leftarrow 0,\ j \leftarrow 0$.

단계2) While $j \leqq l$ do:

단계2.1) If $k_j=0$ then $j \leftarrow j+1$.

단계2.2) Else do:

단계2.2.1) $t \leftarrow min\{l,\ j+r-1\}$,

$h_d \leftarrow (k_t\, k_{t-1} \cdots k_j)_2$.

단계2.2.2) If $h_d > 2^{r-1}$ then do:

$b_d \leftarrow h_d-2^r$, $(k_l\, k_{l-1} \cdots k_{t+1})$을 1씩 증가.

단계2.2.3) Else $b_d \leftarrow h_d$.

단계2.2.4) $e_d \leftarrow j,\ d \leftarrow d+1,\ j \leftarrow t+1$.

단계3) Return 수열 $(b_0,\ e_0),\ (b_1,\ e_1),\cdots,\ (b_{d-1},\ e_{d-1})$.

A signed m-ary sliding window algorithm

입력 : 점 P, $\{(b_i,\ e_i)\}_{i=0}^{d-1}$ such that $k=\sum_{i=0}^{d-1} b_i 2^{e_i}$.

출력 : $Q=[k]P$.

과정 :

사전계산

단계1) $P_1 \leftarrow P,\ P_2 \leftarrow [2]P$.

단계2) For $i=1$ to $2^{r-2}-1$ do $P_{2i+1} \leftarrow P_{2i-1}+P_2$.

단계3) $Q \leftarrow P_{b_{d-1}}$.

주요루프

단계4) For $i=d-2$ to 0 by -1 do:

단계4.1) $Q \leftarrow [2^{e_{i+1}-e_i}]Q$.

단계4.2) If $b_i>0$ then $Q \leftarrow Q+P_{b_i}$.

단계4.3) Else $Q \leftarrow Q-P_{-b_i}$.

단계5) $Q \leftarrow [2^{e_0}]Q$.

단계6) Return Q.

3.3. 주요 연산알고리듬의 비교 및 분석

일반적으로 타원곡선군에서의 상수배는 타원곡선군의 점 G를 k배 한다고 하면, 이것은 G를 k번 타원곡선군에서의 덧셈 연산을 하는 것이다. 점 G를 한 번씩 더해서 k번까지 더하는 이러한 방법을 여기서 General Method(GM)라고 하기로 한다.

GM을 알고리듬의 효율성을 알아보기 위한 기본 척도로 하여 앞 절에서 소개했던 여러 알고리듬 중 몇 가지 알고리듬에 대해 그 효율성을 알아보기로 한다. 알고리듬 상에서 효율적일 것이라고 기대되는 Binary Method(BM), Sliding Window Method(SWM), Binary NAF Method(BNAFM)에 대해 구현하여 그 수행속도를 계산하여 비교 및 분석을 하며, 가장 효율적인 알고리듬에 대해 알아본다.

알고리듬 구현의 환경과 그 내용은 다음과 같다.

구현환경

하드웨어 : Pentium-III(1Ghz), RAM 256MB.
운영체제 : Red Hat Linux release 7.1.9 (Roswell),
Kernel 2.4.7-2 on an i686.
프로그래밍 언어 : C++.
컴파일러 : g++(version 2.96).

구현내용

k를 각 비트별로 랜덤하게 생성하여 타원곡선상의 기본점 G에 대해 kG의 계산을 수행하여 그 수행시간을 microsecond까지 측정하여 결과값을 산출한다. 결과값은 각 20회 이상의 수행을 거쳐 측정한 값들의 평균으로 하였으며, 참고로 GM에 있어서는 20bits를 넘는 k에 대해 수행시간 측정에 어려움이 있어 20비트 이하의 수행결과를 통해 비교 자료로만 이용한다.

구현에 있어서 C++에서 제공하는 정수형 데이터 범위를 넘어서는 부분에 대해서는 NTL 헤더파일을 사용하였다[25]. 그리고 수행속도를 측정하기 위한 도메인 변수는 EC-KCDSA에서 제공하는 검증된 변수중 위수가 160비트 크기인 것을 사용하였다.

이와 같은 여러 사항들을 바탕으로 구현된 타원곡선 상의 연산 알고리듬의 수행 속도 테스트 결과는 <표 3-1>과 같다.

<표 3-1> 타원곡선 상의 상수배 연산 알고리듬 수행 속도 비교

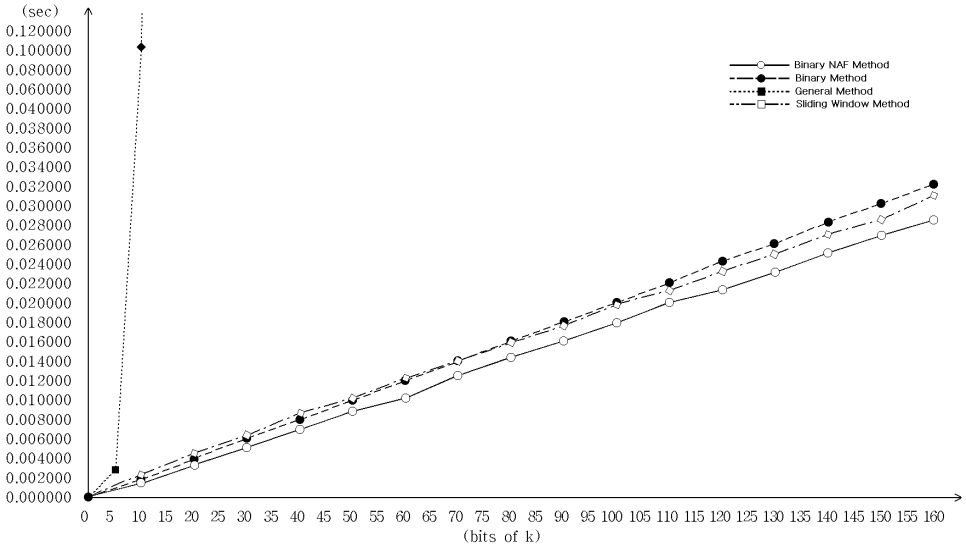
GF(q), q = p			1461501637330902918203684832716283019653785059327			160bit
$y^2 = x^3 + ax + b$			a = 1461501637330902918203684832716283019653785059324, b = 618161358937170673988121756987436237099350920727			
기본점 G = (x, y)			(1168983055804381306122964739888353823030159703303, 591673640518579811944247307252990789873540735386)			
G의 위수 n			1461501637330902918203683057840589624783069764883			160bit
	General Method	Binary Method	Sliding Window Method	Binary NAF Method	k	Bits of k
1	0.002965	0.000779	0.001019	0.000818	$2^4 \sim 2^5 - 1$	5
2	0.102573	0.001825	0.002265	0.001732	$2^9 \sim 2^{10} - 1$	10

3	3.163094	0.002830	0.003234	0.002676	$2^{14} \sim 2^{15}-1$	15
4	101.282056	0.003923	0.004389	0.003569	$2^{19} \sim 2^{20}-1$	20
5	-	0.005941	0.006253	0.005318	$2^{29} \sim 2^{30}-1$	30
6	-	0.008059	0.008564	0.007105	$2^{39} \sim 2^{40}-1$	40
7	-	0.009921	0.010181	0.008878	$2^{49} \sim 2^{50}-1$	50
8	-	0.011883	0.012265	0.010694	$2^{59} \sim 2^{60}-1$	60
9	-	0.013953	0.013946	0.012502	$2^{69} \sim 2^{70}-1$	70
10	-	0.015970	0.015696	0.014400	$2^{79} \sim 2^{80}-1$	80
11	-	0.018000	0.017526	0.016045	$2^{89} \sim 2^{90}-1$	90
12	-	0.019800	0.019460	0.017934	$2^{99} \sim 2^{100}-1$	100
13	-	0.021993	0.021241	0.019650	$2^{109} \sim 2^{110}-1$	110
14	-	0.024169	0.023224	0.021603	$2^{119} \sim 2^{120}-1$	120
15	-	0.025814	0.024899	0.023402	$2^{129} \sim 2^{130}-1$	130
16	-	0.028130	0.026755	0.025280	$2^{139} \sim 2^{140}-1$	140
17	-	0.030140	0.028338	0.026890	$2^{149} \sim 2^{150}-1$	150
18	-	0.032080	0.031293	0.028641	$2^{159} \sim$ n-1	160
비고	-	-	사전계산이 필요	-	-	-

수행속도 측정 결과 GM에서는 k의 비트수가 증가할수록 수행시간이 급격하게 상승하여 20비트 정도의 수준에서는 무려 101초가 넘는 수행시간을 보임으로서 20비트를 넘는 k에 대해서는 측정에 어려움이 있어 이것을 그대로 암호시스템에 적용하기에는 역부족이었다. k가 60비트 이하에서 BM은 SWM보다 대체로 빠른 수행시간을 나타내다가 70비트 정도 수준부터는 SWM에 비해 느린 수행시간을 나타내었다. SWM은 비트수의 증가에 따라서 수행시간의 증가량이 감소하는 것으로 확인 되었다. BNAFM은 작은 비트수의 상수배에서부터 최고 비트수인 160비트까지 그 수행속도가 가장 빠른 것으로 나타났다. 수행속도 측정 결과를 그래프로 나타내면 <그림 3-1>과 같다.

<그림 3-1>에서도 마찬가지로 BNAFM이 가장 수행속도가 빠른 것을 볼 수 있다. 그리고 SWM은 k가 70비트를 넘어서면서 BM의 수행속도를 앞지르는 것을 알 수 있다. 장기적으로 볼 때 암호에서 타원곡선군에서의 유한체의 크기가 커지는 것을 고려하면 SWM이 큰 비트 수에 대해서 비교적 효율적인 연산이 될

수 있을 것으로 기대된다. 그러나 SWM은 부가적인 구현이 필요하며 메모리 효율적인 측면에서 많은 변수를 사용함으로 해서 BNAFM와 비교해 효율성이 낮은 것을 알 수 있다(부록의 “ScalarMult.cpp" 참고).



<그림 3-1> 타원곡선 상의 상수배 연산 알고리듬 수행 속도 비교

지금까지의 수행속도 측정 결과와 구현에서의 효율성에 대해 고찰해 본 결과 BNAFM이 가장 효율적인 연산 알고리듬임을 알 수 있었다. 그러나 SWM이나 BM이 암호에 적용하기에 부적합 하지는 않은 것으로 판명되었다. 실제 수행에서 BM, SWM은 BNAFM과의 수행시간의 차이가 최대 0.005초 미만으로 큰 차이를 보이지 않기 때문이다. 하지만 이러한 문제는 많은 사용자들이 보안 시스템을 이용한는 초대형 시스템에서는 BNAFM과 적지 않은 수행시간의 차이를 나타낼 것으로 보인다.

Ⅳ. 결론 및 향후 과제

유한체에서 정의된 타원곡선군에서의 상수배는 암호시스템의 효율성에 직접적인 영향을 미친다. 이러한 타원곡선군에서의 상수배를 보다 효율적으로 구현하기 위해서는 많은 연구가 필요하다. 상수배에 관한 연구는 현재에도 아주 활발하게 진행되고 있는 부분이다. 본 고에서는 구현에서의 효율성을 고려하여 제시된 여러 알고리듬 중 Binary Method와 Binary NAF Method, Sliding Window Method를 중심으로 연구하였다. 효율성은 주로 상수배에 소요되는 수행시간과 연관되며, 또한 메모리의 효율적인 사용과도 관계된다. 이와 같은 효율성에 근거하여 각각의 알고리듬에 대해 수행시간을 비교, 분석하였다. 그 결과 Binary NAF Method가 연구한 알고리듬 중 가장 효율적임을 알 수 있었으며, 그 외 Binary Method와 Sliding Window Method도 암호 시스템에 적용하기에 무리가 없음이 판명되었다. 그러나 이러한 각 알고리듬이 근소한 차이의 수행시간을 나타내었다는 것을 볼 때, 실제적인 사용에서의 차이는 거의 느낄 수 없을 것으로 기대 되었다. 장기적인 시각으로 본다면, 이러한 상수배 알고리듬은 앞으로도 많은 연구에 의해 더욱 향상된 성능을 갖는 알고리듬이 개발 되어야 할 것이다.

앞으로 무선 단말기나 유선과 무선이 통합된 환경에서의 암호시스템에 적용하기 위해서는 향후의 보안 수준을 고려한다면 더욱 많은 계산량이 요구되므로 상수배 알고리듬의 개선이 불가피해 질 것이다. 전혀 새로운 암호시스템이 나오지 않는 한은 현재의 RSA/DSA에서 ECC로의 전환은 일반적인 생각으로 인식되고 있으며, 또한 표준화 등도 그 동향을 뒷받침 하고 있다. 차후 새로운 암호시스템이 나오더라도 다시 암호시스템의 전환이 일어나기 전까지는 상당시간이 소요될 것이므로 ECC의 상수배 연산 알고리듬의 개선은 중요한 문제이다.

참고문헌

[1] 이민섭, “현대암호학”, 교우사, 2000.

[2] 강주성·김재헌·박상우·박춘식·지성택·하길찬·한재우, “현대암호학”, 경문사, 2000.

[3] 박영호 외5명, “타원곡선에서 스칼라 곱의 고속연산”, 고려대 정보보호 기술연구센터, 2001.

[4] 김영호 외3명, “홀수 표수 확장체위의 타원곡선 고속연산”, 고려대학교, 2000.

[5] 문형돈·이재환, “국내외 무선인터넷 시장 동향”, 전자통신동향분석 제17권 제3호, 2002.

[6] 신균호, “타원곡선 알고리즘이 탑재된 자바기반 무선 전자지갑 구성 방안”, 경희대학교 산업정보대학원 정보통신학과 석사논문, 2001.

[7] 박영호 외4명, “타원곡선상의 고속 곱셈연산을 위한 새로운 분해 알고리즘”, 고려대학교, 2000.

[8] 이석래, “무선보안 기술 동향”, 전자서명인증관리센터, 2000.

[9] “정보보호산업 시장 동향 및 전망”, 주간기술동향 통권1055호, 2002.

[10] 김건우·유형소·문상재, “이동통신에서의 정보보호기술”, Telecomunications Review 제10권 5호 p939~952, 2000.

[11] 정종필, “무선인터넷 보안의 발전방향”, 엠아이시큐리티, 2002.

[12] 박기준, “무선 보안과 M-Commerce", Dream Security 무선보안사업본부, 2002.

[13] 김문구, “무선 전자상거래의 성장동인 및 시장활성화 방향”, ETRI IT정보센터 주간기술동향 통권 1062호 p34~39, 2002.

[14] 이경효, “타원곡선 알고리즘을 이용한 암호 시스템에 관한 연구”, 목포대학교 멀티미디어공학과 석사논문, 2001.

[15] 이동훈·황효선·임채훈, “타원곡선 암호의 기초와 응용”, (주)퓨쳐시스템 암호체계센터 FS-TR01-03, 2001.

[16] 정보통신단체표준 TTAS.KO-12.0015, “부가형 전자서명 방식 표준 - 제3부 : 타원곡선을 이용한 인증서 기반 전자서명 알고리듬”, 한국정보통신기술협회, 2001.

[17] http://www.iso.ch/

[18] http://standards.ieee.org/

[19] http://www.secg.org/

[20] http://www.ansi.org/
[21] http://www.itl.nist.gov/fipspubs/
[22] http://www.tta.or.kr/
[23] http://www.rsasecurity.com/
[24] http://www.certicom.com/
[25] http://www.shoup.net/

부록 (Source Code)

"EcDomain.h"

```
/*=== Heder File for "ScalarMult.cpp" : "EcDomain.h" ===*/
#ifndef _EcDomain_h
#define _EcDomain_h
#endif

/* Header Files Include */
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <NTL/ZZ.h>
#include <sys/time.h>
#include <time.h>
#include <string.h>

/* Define Variables */
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
#ifdef MAX
#undef MAX
#endif
#ifndef MAX
#define MAX 200
#endif

/* Define Macro */
#define IsEqual(x,y) (((x)==(y)) ? TRUE : FALSE)
#define BinaryRepresent(x,i) (((x) >> (i)) & ~(~0 << 1))
```

```
#define MaxNumberValue(x,y) ((x)>(y) ? (x) : (y))

/* Define Structure */
typedef struct{
        ZZ x;
        ZZ y;
}coordinate;

typedef struct{
        int Pos;
        int NumVal;
}NumValPos;

typedef struct timeval ElapsedTimeCheck;

/*=== End Of Header File "EcDomain.h" ===*/
```

"ScalarMult.cpp"

```
/*=== Scalar Multiplication For ECC : "ScalarMult.cpp" ===*/
/* Include Header File */
#include"EcDomain.h"

/* Function Prototype */
coordinate PointAddition(coordinate A, coordinate B, int DISTINCTION);
void ZtoBinaryRepresent(ZZ n, ZZ s[]);
ZZ ModPrimeDiv(ZZ x, ZZ y);

/* Extern Variables Setup */
ZZ p,a,b,n,xG,yG;

main()
{
/* Variables Setup */
        ZZ k,l,t,e[MAX],h[MAX];
        long i,j,index=0,m=1,MaxNumVal,sec,msec,BitsOfK;
        int DISTINCTION=FALSE;
        coordinate G,Q,NegG,P[MAX];
        NumValPos T[30];
        ElapsedTimeCheck start, finish;


/* Domain Parameters Setup */
p=to_ZZ("1461501637330902918203684832716283019653785059327");
a=to_ZZ("1461501637330902918203684832716283019653785059324");
b=to_ZZ("6181613589371706739881217569874362370993509207 27");
n=to_ZZ("1461501637330902918203683057840589624783069764883");
xG=to_ZZ("1168983055804381306122964739888353823030159703303");
yG=to_ZZ("5916736405185798119442473072529907898735407353 86");

        G.x=xG;
        G.y=yG;
        NegG.x=G.x;
```

```
        NegG.y=-G.y;
        Q=G;

        t = time(NULL);
        SetSeed(t);

        cout << "₩n" << "input bits of k = ";
        cin >> BitsOfK;

        while(NumBits(k=RandomBits_ZZ(BitsOfK))==BitsOfK);
        if(k>=n) {
                puts("error : k is larger than n₩n");
                exit(1);
        }
        l=3*k;
        cout << "compute Q = [" << k << "] G" << "₩n";
        cout << "where G = ( " << xG << ", " << yG << " )" << "₩n₩n";


/* General Method */
        gettimeofday (&start, NULL);
        for(i=1;i<k;i++){
                DISTINCTION=IsEqual(G.x,Q.x);
                Q=PointAddition(G,Q,DISTINCTION);
        }
        gettimeofday (&finish, NULL);
        cout << "General Method : ₩n";
        cout << "Q = ( " << Q.x << ", " << Q.y << " )" << "₩n";
        if(finish.tv_usec-start.tv_usec < 0){
                sec=finish.tv_sec-start.tv_sec-1;
                msec=1000000+(finish.tv_usec-start.tv_usec);
        }else{
                sec=finish.tv_sec-start.tv_sec;
                msec=finish.tv_usec-start.tv_usec;
        }
```

```
        printf("Elapsed Time = %0lu.%06lu (sec)₩n₩n",sec,msec);


/* Binary Method */
        gettimeofday (&start, NULL);
        ZtoBinaryRepresent(k, e);
        Q=G;
        for(i=NumBits(k)-2 ; i>=0 ;i--){
                Q=PointAddition(Q,Q,TRUE);
                if(e[i]==1) {
                        DISTINCTION=IsEqual(G.x,Q.x);
                        Q=PointAddition(Q,G,DISTINCTION);
                }
        }
        gettimeofday (&finish, NULL);
        cout << "Binary Method : ₩n";
        cout << "Q = ( " << Q.x << ", " << Q.y << " )" << "₩n";
        if(finish.tv_usec-start.tv_usec < 0){
                sec=finish.tv_sec-start.tv_sec-1;
                msec=1000000+(finish.tv_usec-start.tv_usec);
        }else{
                sec=finish.tv_sec-start.tv_sec;
                msec=finish.tv_usec-start.tv_usec;
        }
        printf("Elapsed Time = %0lu.%06lu (sec)₩n₩n",sec,msec);


/* Sliding Window Method */
        gettimeofday (&start, NULL);
        P[0]=G;
        Q=G;
        for(i=0;i<NumBits(k);i){
                if(e[i]==1){
                        j=i;
                        while(e[i]==1) i++;
```

```
                if(i-j>=2) {
                        T[index].Pos=i-1;
                        T[index++].NumVal=i-j;
                }
        }else while(e[i]==0) i++;
}
MaxNumVal=1;
for(i=0;i<index;i++) MaxNumVal = MaxNumberValue(T[i].NumVal,
                                        MaxNumVal);
for(i=1;i<MaxNumVal;i++) {
        P[i]=PointAddition(P[i-1],P[i-1],TRUE);
        DISTINCTION=IsEqual(P[i].x,G.x);
        P[i]=PointAddition(P[i],G,DISTINCTION);
}
m=index-1;
for(i=NumBits(k)-2; i>=0; i){
        if(i==T[m].Pos){
                for(j=0;j<T[m].NumVal;j++){
                        Q=PointAddition(Q,Q,TRUE);
                        i--;
                }
                DISTINCTION=IsEqual(Q.x,P[T[m].NumVal-1].x);
             Q=PointAddition(Q,P[T[m].NumVal-1],DISTINCTION);
                m--;
        } else if(e[i]==0){
                Q=PointAddition(Q,Q,TRUE);
                i--;
        } else {
                Q=PointAddition(Q,Q,TRUE);
                DISTINCTION=IsEqual(Q.x,G.x);
                Q=PointAddition(Q,G,DISTINCTION);
                i--;
        }

}
```

```
        gettimeofday (&finish, NULL);
        cout << "Sliding Window Method : ₩n";
        cout << "Q = ( " << Q.x << ", " << Q.y << " )" << "₩n";
        if(finish.tv_usec-start.tv_usec < 0){
                sec=finish.tv_sec-start.tv_sec-1;
                msec=1000000+(finish.tv_usec-start.tv_usec);
        }else{
                sec=finish.tv_sec-start.tv_sec;
                msec=finish.tv_usec-start.tv_usec;
        }
        printf("Elapsed Time = %0lu.%06lu (sec)₩n₩n",sec,msec);


/* Binary NAF Method */
        gettimeofday (&start, NULL);
        ZtoBinaryRepresent(l, h);
        Q=G;
        for(i=NumBits(l)-2 ; i>=1 ;i--){
                Q=PointAddition(Q,Q,TRUE);
                if(h[i]==1 && e[i]==0) {
                        DISTINCTION=IsEqual(G.x,Q.x);
                        Q=PointAddition(Q,G,DISTINCTION);
                }
                if(h[i]==0 && e[i]==1) {
                        DISTINCTION=IsEqual(NegG.x,Q.x);
                        Q=PointAddition(Q,NegG,DISTINCTION);
                }
        }
        gettimeofday (&finish, NULL);
        cout << "Binary NAF Method : ₩n";
        cout << "Q = ( " << Q.x << ", " << Q.y << " )" << "₩n";
        if(finish.tv_usec-start.tv_usec < 0){
                sec=finish.tv_sec-start.tv_sec-1;
                msec=1000000+(finish.tv_usec-start.tv_usec);
        }else{
```

```
            sec=finish.tv_sec-start.tv_sec;
            msec=finish.tv_usec-start.tv_usec;
        }
        printf("Elapsed Time = %0lu.%06lu (sec)₩n₩n",sec,msec);
}

/* Point Addition Function */
coordinate PointAddition(coordinate A, coordinate B, int DISTINCTION)
{
        coordinate rslt;

        if(DISTINCTION==FALSE){
            rslt.x=AddMod(SqrMod(ModPrimeDiv(B.y - A.y, B.x -
                A.x), p), AddMod(- A.x, - B.x, p), p);
            rslt.y=AddMod(-A.y, MulMod(ModPrimeDiv(B.y - A.y, B.x
                - A.x), (A.x - rslt.x),p), p);
        }else{
            rslt.x=AddMod(SqrMod(ModPrimeDiv(3 * SqrMod(A.x, p) +
                a, 2*A.y), p), - 2 * A.x, p);
            rslt.y=AddMod(-A.y, ModPrimeDiv(3 * SqrMod(A.x, p) +
                a, 2 * A.y) * (A.x - rslt.x), p);
        }

        rslt.x=rslt.x%p;
        rslt.y=rslt.y%p;
        return rslt;
}

/* Big-Integer to Binary Representation */
void ZtoBinaryRepresent(ZZ n, ZZ s[])
{
        long i=0;

        for(i=0;i<NumBits(n);i++)
            s[i]=BinaryRepresent(n,i);
```

```
}

/* x/y by Mod Prime */
ZZ ModPrimeDiv(ZZ x, ZZ y)
{
        ZZ rslt,i;

        y=y%p;
        y=InvMod(y, p);
        rslt=MulMod(x, y, p);

        return rslt;
}

/*=== End Of File "ScalarMult.cpp" ===*/
```