F_{p^m}에서 정의된 타원곡선 암호시스템의 효율적인 구현*

김 덕 수†, 이 은 정‡, 심 상 규†, 이 필 중†

†포항공과대학교 전자전기공학과

‡포항공과대학교 정보통신 연구소

Efficient Implementation of Elliptic Curve Cryptosystem over F_{p^m}

Duk Soo Kim†, Eun Jeong Lee‡, Sang Gyoo Sim†, Pil Joong Lee†

†Dept. of Electronic and Electrical Eng., POSTECH, Pohang, Korea

‡POSTECH Information Research Laboratories, Pohang, Korea

요 약

본 논문에서는 타원곡선 암호시스템의 효율적인 소프트웨어 구현 방법을 제안하였다. 타원곡선과 유한체 F_{p^m}의 선택 방법을 제안하고, 선택한 타원곡선에서 생성자 G를 찾는 방법을 제시하였다. 타원곡선 위의 점에 대한 상수배(scalar multiplication)를 효율적으로 구현하기 위해서 덧셈/뺄셈 사슬을 사용한 윈도우 방식을 채택하여 타원곡선에서의 KCDSA(EC-KCDSA)를 구현하고 수행 성능과 수치 예를 보였다.

1. 서 론

타원곡선 암호시스템(ECC, Elliptic Curve Cryptosystem)은 Koblitz[6]와 Miller[12]에 의해서 독립적으로 제안된 공개키 암호시스템이다. 이 암호시스템의 중요한 장점 중의 하나는 유한체에서의 이산대수 문제(DLP, Discrete Logarithm Problem)와 비교하여 타원곡선에서 정의된 이산대수 문제(ECDLP, Elliptic Curve DLP)가 매우 어렵다고 알려져 있어서, 현재 널리 사용되고 있는 RSA

*본 연구는 정보통신연구관리단의 국책기술 개발사업(타원곡선 암호시스템을 이용한 차세대정보보호 기술의 연구 개발)의 지원에 의해 이루어졌다.

에 비해서 같은 암호학적 안전도를 유지하기 위한 키길이가 현저히 작게 되고, 저장 용량 및 대역폭(bandwidth)의 제한이 있는 smart card나 무선통신에도 유용하게 사용될 수 있다는 것이다. 이는 아직 일반적인 ECDLP에 적용될 수 있는 부지수(subexponential) 알고리즘이 없기 때문이다.

타원곡선에서의 연산은 여러 번의 유한체 연산으로 수행되기 때문에 ECC의 성능은 유한체 연산에 의해 좌우된다. 이제까지 유한체의 효율적인 연산에 대한 많은 연구가 이루어져 왔지만, F_p또는 F_{2^m}에 관한 연구들이 대부분이었다. 하지만, 요즘의 마이크로 프로세서들은 "word"라 불리우는 데이타의 단위로 연산을 수행하기 때문에 암호시스템에 사용될 수 있는 크기의 유한체 F_{2^m}에서의 비트별 연산과 F_p에서의 큰 수 연산은 소프트웨어상의 구현에는 부담이 된다. 이러한 이유로 본 논문에서는 마이크로 프로세서가 정보를 다루는 단위인 word를 최대한 이용하여 효율적인 소프트웨어상의 구현이 되도록, [1, 9]에서와 같이, word 사이즈의 최대값보다는 작지만 가까운 $p = 2^n - c$를 선택하여 유한체 F_{p^m}상의 타원곡선 암호시스템을 구현할 것이다. 여기서 n은 word의 크기이고 c는 p가 n비트 소수가 되도록 하는 작은 자연수이다.

§2에서는 유한체 F_{p^m}에서 정의된 타원곡선를 소개하고, ECDLP와 암호학적으로 안전한 타원곡선의 선택시 ECDLP와 관련하여 고려해야할 문제들을 살펴본다. 또한, 타원곡선에서의 KCDSA(EC-KCDSA)를 소개한다. §3에서는 ECC구현을 위한 타원곡선과 유한체의 선택 방법을 제안하고, §4에서는 선택한 타원곡선에서의 생성자 G를 찾는 방법을 제시한 후, 그 결과의 예를 보여준다. §5에서는 ECC에서 사용하는 타원곡선위의 점에 대한 정수배(scalar multiplication)을 효과적으로 수행하는 방법을 소개하고 구현을 위한 유사코드를 제시한다. §6에서는 구현 결과로서 EC-KCDSA의 수치예와, 서명의 생성 및 검증의 수행 시간을 보인다.

2. 타원곡선 암호시스템 소개

소수 p와 양의 정수 m에 대하여, 유한체 F_{p^m}에서 정의된 타원 곡선은 $p > 3$인 경우, 다음과 같이 정의된다.

$$E(F_{p^m}) = \{(x, y) \in F_{p^m} \times F_{p^m} \mid y^2 = x^3 + ax + b,\ \ a, b \in F_{p^m},\ 4a^2 + 27b^2 \neq 0\} \cup \{\mathcal{O}\} \quad (1)$$

타원곡선 $E(F_{p^m})$상의 점들은 [10]에서와 같이 정의된 덧셈 연산에 대하여 가환 군(abelian group)을 이루게 되고, 식 (1)의 $\mathcal{O}$은 무한대 점으로 이 덧셈 연산의 항등원(identity)에 해당한다. 이하 논문에서 타원곡선 $E(F_{p^m})$를 E, E의 위수(order)를 $N = \#E(F_{p^m})$라 하고, N의 가장 큰 소인수를 q라고 하겠다.

2.1 타원곡선에서의 이산대수 문제

타원곡선 암호시스템은 이산대수 문제의 어려움, 즉 q를 위수로 갖는 점 $G \in E(F_{p^m})$로 생성된 순환군 $<G>$에서 임의의 점 $Q \in <G>$에 대하여 $Q = lG$되는 l을 찾는 문제가 어렵다는 사실로 부터 안

전성을 얻게 된다. 타원곡선위에서 정의된 이산대수 문제(ECDLP)를 풀기 위하여 Shanks의 baby-step giant-step algorithm과 이것의 랜덤화된 버전인 Pollard-ρ algorithm[17], 그리고 Pohlig-Hellan algorithm[16] 등이 있다. 이 알고리즘들은 q의 이중근에 비례하는 수행 시간을 갖고 있어서 이중근 알고리즘이라고 하고, 타원곡선의 위수가 2^{160}이상의 소수로 나누어 지면, 이산대수 문제가 충분히 어렵다고 알려져 있다[25].

유한체상의 이산대수 문제를 부지수(subexponential) 실행 시간내에 풀 수 있는 index-calculus algorithm은 타원곡선에 적용하기 어려우며[12], 오히려 brute force attack보다 더 많은 cost를 요구하는 것으로 알려졌다[22]. 현재까지 일반적인 ECDLP를 푸는 부지수 실행 시간내 알고리즘은 알려진 것이 없다.

한편, Menezes, Okamoto, Vanstone(MOV attack[14])은 Weil-pairing을 이용하여 $E(F_{p^m})$에서의 이산대수 문제를, 1이상의 어떤 B에 대하여 유한체 $F_{p^{Bm}}$에서의 이산대수 문제로 전환시킬 수 있음을 보였다. 만일 이러한 경우 B의 값이 작다면, 전환된 유한체 $F_{p^{Bm}}$에서의 DLP에 index-calculus algorithm을 적용하여 부지수 시간내에 ECDLP를 풀 수 있게 된다. 타원곡선이 초특이(supersingular)인 경우에는 $B \leq 6$라고 알려져 있어서 ECDLP에 대한 효과적인 MOV attack이 가능하므로, 타원곡선 $E(F_{p^m})$ 선택할 때 $F_{p^{Bm}}$에서의 DLP를 풀기 어렵도록 p^m을 크게 선택해야 한다.

[25, 26]에서는 MOV attack에 대해 안전성을 보장받기 위해서는 $B \geq 20$을 만족해야 한다고 권고하고 있다. $E(F_{p^m})$상의 ECDLP가 $F_{p^{Bm}}$상의 DLP로 전환되기 위해 요구되는 MOV condition의 필요조건은 $p^{Bm} \equiv 1 \bmod N$이므로[10], 임의로 주어진 타원곡선을 이용한 타원곡선 암호시스템이 MOV attack에 안전하려면

$$\text{For all } j \text{ such that } 1 \leq j < 20,\ p^{jm} \not\equiv 1 \bmod q \tag{2}$$

을 만족해야 한다.

이와 함께, F_{p^m}상의 곡선의 점의 개수가 정확히 p^m이 되는 특징을 가지고 있는 변칙 타원곡선(anomalous curve)을 사용하여 암호시스템을 구성할 때에는 이중 특이한 부류에서 ECDLP가 쉽게 풀릴 수 있기 때문에[18, 20, 23], 타원곡선의 선택시 변칙 타원곡선인 지를 확인해야 한다.

2.2 타원곡선 KCDSA(EC-KCDSA)

한국형 전자서명 표준 알고리즘인 KCDSA(Korean Certificate-based Digital Signature Algorithm)[24]를 타원곡선 암호시스템에 적용한 EC-KCDSA[15, 24]에 대해서 간략히 기술하겠다.

시스템 변수들로서 p, m, E, G, q, 그리고 해쉬함수 h를 정하고 이것이 결정되면 이 시스템의 사용자 A는 다음과 같은 사용자 변수들(user parameters)을 갖게 된다.

- 비공개키 x_A : 비공개의 서명키로 $\mathbb{Z}_q^*$에서 랜덤하게 선택된다.
- 공개키 Y_A : 서명 검증용의 공개키로서 $Y_A = \bar{x}_A G$, 여기서 $\bar{x}_A = x_A^{-1} \bmod q$
- z_A : A의 인증정보(식별자, 시스템 변수, Y_A 등)의 해쉬값

서명자 A는 메세지 M에 대한 서명값 (r, s)를 다음의 과정을 통하여 생성한다. 여기서 r은 해쉬함수 h의 해쉬값이고, s는 $\mathbb{Z}_q^*$의 원소이다.

1. k를 [2, n-1]사이에서 랜덤하게 선택한다.
2. 타원곡선 점 $kG = (x_1, y_1)$을 계산한다.
3. 해쉬값 $r = h(kG) = h(x_1||y_1)$을 계산한다.
4. 해쉬값 $H = h(z_A||M)$를 계산한다.
5. $e = (r \oplus H) \bmod q$를 계산한다.
6. $s = x_A(k - e) \bmod q$를 계산한다.

만일 $s = 0$이면 새로운 k를 선택하여 서명생성을 다시 한다. 검증자는 서명자에게 서명값 (r', s')와 메시지 M'를 받아서 다음과 같이 서명 검증을 할 수 있다.

1. r', s'의 bit size를 검사한다. r'은 h의 해쉬값의 bit size가 되어야 하며 s'는 $0 < s' < q$을 만족한다.
2. 해쉬값 $H' = h(z_A||M')$을 계산한다.
3. $e' = (r' \oplus H') \bmod q$를 계산한다.
4. $(x_2, y_2) = s'Y_A + e'G$를 계산한다.
5. $r'' = h(x_2||y_2)$을 계산한다.
6. 받은 r'과 계산한 r''이 같은지 비교함으로써 서명을 검증한다.

3. 타원곡선 및 유한체의 선택

식 1와 같이 정의된 타원곡선의 두 점들의 한 번 연산은 유한체에서의 덧셈, 곱셈, 역수 계산으로 이루어져 있다. 따라서, 효과적인 타원곡선 암호 시스템을 만들기 위해서는 연산을 효과적으로 할 수 있는 유한체를 선택해야 한다. 유한체 F_{p^m}은 차수가 m인 기약 다항식 $f(x)$가 정해질 때 $F_p[x]/(f(x))$와 동치 관계에 있음은 잘 알려진 사실이다. 여기서, $f(x)$를 유한체 F_{p^m}의 minimal polynomial이라고 한다. 따라서, $F_{p^m} = \{\sum_{i=0}^{m-1} a_i x^i | a_i \in F_p\}$와 같이 F_p에서 정의된 차수가 m보다 작은 다항식들의 집합으로 볼 수 있고, 이와 같이 정의된 유한체에서의 효율적인 연산은 [9]에 잘 설명되어 있다.

- p의 선택

현재까지 암호시스템을 위한 유한체로서 F_{2^m}와 F_p가 많이 연구 및 구현되어 왔다. 그러나, F_{2^m}의 경우에 사용되는 많은 비트 단위 연산은 n-bit word단위로 데이타를 처리하는 컴퓨터에서는 비효율적이며 p가 큰 정수일 때에는 다정도 정수 연산을 해야 하므로 word들 사이의 carry 등으로 인하여 비효율적인 면이 발생한다. 본 논문에서는 [1, 9]와 같이 소프트웨어 구현에 적합한 유한체를 위하여 p를 word 크기 최대값의 정수에 가까우면서 적은 값를 사용한다.

• $f(x)$의 선택

$F_p[x]/(f(x))$라 정의된 유한체에서의 연산은 $f(x)$로 모듈라 감소를 매번 수행해야 하므로 $f(x)$가 간단한 형태로 선택되는 것이 좋다. p가 2가 아닌 경우 [11](pp.40)의 정리를 이용하여 다음의 따름정리를 얻을 수 있고, 이를 통하여 항의 갯수가 2개인 binomial polynomial $f(x)$를 찾을 수 있다.

따름정리 1 *w 를 F_p^* 의 생성자라 하고, m 을 $p-1$의 소인수라 하자. 그러면 binomial $x^m - w$ 은 F_p위에서 기약 다항식이다.*

• 타원곡선의 선택

타원곡선 암호 시스템은 타원곡선에서 정의된 이산대수 문제의 어려움을 이용하고 있으므로 알려진 이산 대수 알고리즘에 안전하도록 타원곡선이 선택 되어야 한다. 이를 위하여 먼저 타원곡선의 위수가 큰 소수로 나누어져야 하고, MOV condition(식 (2))을 만족해야 한다. 타원곡선의 위수는

$$\text{(Hasse's theorem[21])} \quad |t = p^m + 1 - \#E(F_{p^m})| \leq 2\sqrt{p^m} \tag{3}$$

이므로, 유한체의 크기를 알 때 타원곡선의 위수를 대략 짐작할 수 있다. 식 (3)에서 p가 t를 나누면 E를 초특이(supersingular) 곡선이라 한다. 따라서, 암호 시스템에 사용하기 위한 타원곡선의 위수의 조건을 다음과 같이 정리할 수 있다.

$$\text{size of } F_{p^m} \approx \text{ order of } E(F_{p^m}) \approx \text{ size of large prime factor } \approx 2^{160}. \tag{4}$$

임의의 타원곡선에 대하여 위수를 구하는 알고리즘으로 Schoof의 알고리즘[19]이 있으나, 실제로 구현하기가 어렵고 수행 시간도 많이 걸리는 것으로 알려져 있다. 그 뒤 위수를 구하기 위한 많은 노력이 있었으며[13], Schoof의 알고리즘을 구현하기 쉽게 제안하는 경우도 발표된 바 있다[4].

Koblitz는 [7]에서 타원곡선의 두 상수 a, b를 부분체 F_p에서 선택하면 Hasse에 의해 증명된 Weil conjecture를 이용하여

$$a_{n+1} = ta_n - pa_{n-1} \text{ for } n \geq 1, a_0 = 2, a_1 = t = p + 1 - \#E(F_p) \tag{5}$$

와 같이 $a_m = \#E(F_{p^m})$을 쉽게 구할 수 있음을 제안하였다. 여기서, p를 작은 수로 선택하면 모든 $x \in F_p$에 대해 $x^3 + ax + b$가 F_p에서 제곱수인가를 검사함으로써 a_1을 계산할 수 있다. 이경우 m의 소인수 m_1에 대하여 $\#E(F_{p^{m_1}})$이 $\#E(F_{p^m})$을 나누므로 조건 (4)을 만족하기 위하여 즉, 타원곡선의 위수가 큰 소수와 작은 소수들의 곱으로 나타내게 하기 위해서는 m을 소수로 선택하는 것이 좋

다. 만약, 이런 조건을 만족하는 소수 m이 없다면, 타원곡선의 위수의 큰 소인수가 2^{160}정도 되게 하는 합성수 m을 신중하게 선택하도록 한다.

다음의 알고리즘은 임의로 선택된 E/F_p에 대하여 타원곡선 암호 시스템에 적합한 타원곡선을 찾기 위한 것이다.

알고리즘 1 (타원곡선 찾기)

Input : p, m

Output : a, b, E의 위수 $\#E$, $\#E$의 가장 큰 소인수 q

1. F_p에서 a와 b를 랜덤하게 선택한다. (식 (1)를 만족하기 위해 a와 b가 동시에 0이면 안된다.)
2. $\#E(F_p)$와 $a_1 = p + 1 - \#E(F_p)$를 계산한다.
3. $\#E(F_{p^m}) = p^m + 1 - a_m$를 계산한다.
4. (Supersingular case) 만약 p가 a_m을 나누면, 단계 1로 간다.
5. $\#E(F_{p^m})$를 소인수 분해한다.
6. $\#E(F_{p^m})$가 큰 소인수($\approx 2^{160}$)를 갖지 않으면, 단계 1로 간다.
7. (MOV condition check) $\#E(F_{p^m})$의 가장 큰 소인수를 q라 할 때, $B = 1, \ldots, 19$에 대하여, $p^{Bm} \equiv 1 \mod q$인 B가 하나라도 존재하면, 단계 1로 간다.

위의 알고리즘의 단계 5에서 $\#E(F_{p^m})$의 소인수를 모두 구할 필요는 없다. 조건 (4)에 의하여 타원곡선의 위수는 작은 소수들과 큰 소수의 곱으로 표현되어야 하므로 계산된 위수를 작은 소수들로 나누어 보고 그 나머지를 소수 테스트 하는 방법을 사용하면 쉽게 단계 5를 수행할 수 있다.

4. 생성자 G찾기

타원곡선 E에서 위수 q를 갖는 점 G를 찾고자 한다. 이 점 $G = (x, y)$는 시스템 변수로서 모두에게 공개되는 값이 될 것이다. 먼저 G는 타원곡선 E위의 점이므로 x, y가 타원곡선의 식을 만족하도록 선택되어야 한다. 이는 x를 F_{p^m}에서 임의로 선택한 후 $u = x^3 + ax + b$가 $F_{p^m}^*$에서 square root을 갖는다면 선택된 x와 $u = y^2$을 만족하는 y는 타원곡선의 식을 만족함을 알 수 있다. 따라서, 타원곡선의 점을 찾기 위하여 먼저 F_{p^m}에서의 square와 square root에 관한 다음 정리들을 소개한다[8].

전처정리 2 $f(x)$ 가 F_{p^m}의 minimal polynomial 일 때, $\alpha \in F_{p^m}^*$에 대하여, "$\alpha^{(p^m-1)/2} \equiv 1 \mod f(x)$"와 "$\alpha$가 (p^m)에서 square"인 것은 동치이다.

전치정리 3 $p^m - 1 = 2^k s$이라 놓고, n은 F_{p^m} 에서 square가 아닌 원소라 하자. 여기서 s는 홀수이고, k는 2로 나누어지는 최대 수이다. F_{p^m}에서 square인 α에 대하여,

$$b = n^s,\ r = \alpha^{(s+1)/2} \text{ in } F_{p^m}$$

을 정의하자. F_{p^m}에서 $\alpha = x^2$을 만족하는 제곱근 x는 $x = b^j r$으로 얻어진다. 여기서, $j = j_0 + 2j_1 + \cdots + 2^{k-2} j_{k-2}$ 는 다음과 같이 정의된다;

1. 만약 $(r^2/\alpha)^{2^{k-2}} = 1$이면, $j_0 = 0$, 그렇지 않으면 $j_0 = 1$
2. $i = 1, \ldots, k-2$에 대하여
   만약 $\left(\frac{(b^{j_0 + 2j_1 + \cdots + 2^{i-1} j_{i-1}} r)^2}{\alpha} \right)^{2^{k-i-2}} = \begin{cases} 1 \\ -1 \end{cases}$ 이면, $j_i = \begin{cases} 0 \\ 1 \end{cases}$

따름정리 4 *m이 홀수이고, F_{p^m}에서 square인 α에 대하여, α의 제곱근 x를 다음과 같이 쉽게 계산할 수 있다.*

1. *만약 $p \equiv 3 \pmod 4$이면, $x = \alpha^{(p^m+1)/4}$*
2. *만약 $p \equiv 5 \pmod 8$이면, $x = \alpha r(s-1)$, 여기서 $r = (2\alpha)^{(p^m-5)/8}$, $s = 2\alpha r^2$이다.*

다음의 알고리즘은 타원곡선 $E(F_{p^m})$의 위수 N과 찾고자 하는 생성자 G의 위수 q가 주어졌을 때, ECC를 위한 생성자 G를 찾기 위한 것이다.

알고리즘 2 (생성자 G찾기)
Input : $a, b, F_{p^m}, N = \#E(F_{p^m})$, N의 가장 큰 소인수 q
Output : 위수 q를 갖는 생성자 $G \in E(F_{p^m})$

1. *x를 F_{p^m}에서 임의로 선택한다.($x \in_r F_{p^m}$)*
2. *$u = x^3 + ax + b$를 만족하는 u를 계산한다.*
3. *$u^{(p^m-1)/2} \equiv 1 \mod f(x)$을 만족하지 않으면 단계 1로 간다.*
4. *u의 제곱근 y를 계산하여 임의의 점 $Q = (x, y)$를 찾는다.*
5. *$G := \frac{N}{q} Q$를 계산한다.*
6. *만약 $G = \mathcal{O}$이면 단계 1로 간다.*
7. *G를 출력한다.*

알고리즘 1과 알고리즘 2를 구현하여 얻은 타원곡선의 파라미터 a와 b, 생성자 G의 예는 표 1에 보였다. 여기서 사용한 유한체는 $F_{(2^{16}-129)^{11}}$이고, t는 식 (5)의 t이다. 이하 모든 수치 예에서, F_{p^m}의 원소 $\sum_{i=0}^{m} a_i x^i$는 $(a_{m-1}, \ldots, a_1, a_0)$로 표현하겠다.

a, b, t (10진수 사용)	$N = \#E, q, G$ (모두 16진수로 표기하였다.)
a=1, b=12, t=47	N = fa82 e22f7fd1 7cdfe110 3ac0add5 5ba5cb22 e3a9b037 q = fb2e9708 becbea45 063f8029 4a0efbe3 fbfc2507 $G = (G_x, G_y)$ G_x = (3329, 0847, b9d1, 769e, d34f, 8c22, b6f1, ff27, 3d65, 70d6, e151) G_y = (1dfe, 7002, 01fd, bb83, f3d4, ee8f, f14f, f657, e66f, f70b, 8be4)
a=2, b=95, t=67	N = fa82 e22f7fd1 7cdfe110 3b09aaac 8c49dde7 11c2cd9b q = fb4245aa 90bdbd67 26a2a8f0 59c0ea3c 49d34db7 $G = (G_x, G_y)$ G_x = (8c33, 0a99, 7340, 71d3, 9e20, 2b2a, 7179, 256e, f584, 054f, 253e) G_y = (8484, 81bd, f6c7, 3c4f, 61ac, a839, c581, 1a2e, ae9a, cdc5, 2963)
a=3, b=12, t=223	N = fa82 e22f7fd1 7cdfe110 3afded4f 5d59649b bd50fbd9 q = fa260e10 47c6da14 f349f28c eb0426ce fdcb8ec7 $G = (G_x, G_y)$ G_x = (07d1, 86a3, 0dad, 91cf, 21e2, 2164, 9ba3, 4903, 0ae9, 780b, 08d9) G_y = (7842, 6c96, 1add, c75a, cd6b, 47ba, 208b, 2159, 455a, c7cf, c253)
a=3, b=46, t=19	N = fa82 e22f7fd1 7cdfe110 3849e6dc 587443aa aca5b4d5 q = faedb967 70dc8ac6 f7c7b853 625d3a26 05db2af7 $G = (G_x, G_y)$ G_x = (6b21, da92, 8379, a491, 3244, 0722, c3e0, 8212, 6898, 3c2b, d21c) G_y = (d7bf, c390, 97ff, 1fbf, e709, 2ef8, eb1a, fb66, f754, 9b7a, 491d)
a=3, b=63, t=19	N = fa82 e22f7fd1 7cdfe110 3849e6dc 587443aa aca5b4d5 q = faedb967 70dc8ac6 f7c7b853 625d3a26 05db2af7 $G = (G_x, G_y)$ G_x = (5143, 995c, dd5f, aad6, 6ed2, c8e7, 5fba, 558f, 76ac, 10e8, eb1e) G_y = (f3c1, c98c, 2189, f9ba, 7326, 0991, 041b, b014, d56a, d5cc, 8b8f)
a=4, b=14, t=67	N = fa82 e22f7fd1 7cdfe110 3b09aaac 8c49dde7 11c2cd9b q = fb4245aa 90bdbd67 26a2a8f0 59c0ea3c 49d34db7 $G = (G_x, G_y)$ G_x = (9cae, 633d, 8eb0, 8447, 97bf, 3ece, a954, 06b3, 0178, d661, 5109) G_y = (60c0, 23e1, 7cd3, bd74, b1f1, 3e44, eafb, 6e7d, 32ee, 607b, a6d8)

표 1: $F_{(2^{16}-129)^{11}}$에 대하여 타원곡선 파라미터와 생성자의 예

5. 반복적인 덧셈 연산

타원곡선 상의 한 점에 대해서 임의의 정수배(scalar multiplication)를 계산하는 것은 반복적인 덧셈으로 이루어지는데, 이것을 효율적으로 구현하는 것은 타원곡선 암호시스템의 중요한 부분으로서 시스템의 성능에 큰 영향을 미치게 된다. 이 문제를 해결하는 체계적인 방법으로 제시된 것이 덧셈 사슬(addition chain)과 덧셈/뺄셈 사슬(addition/subtraction chain)이다[28]. 이 중 덧셈/뺄셈 사슬의 정의는 다음과 같다.

정의 5 *덧셈/뺄셈 사슬*
임의의 양의 정수 n에 대한 덧셈/뺄셈 사슬은 다음과 같은 성질을 가진 수열 $\{a_i\}_{i=0}^{i=l}$로 정의되고, l을 덧셈/뺄셈 사슬의 길이라 한다.

$$a_0 = 1,\ a_l = n$$
$$a_k = \pm a_j \pm a_i, \quad 0 \le i \le j < k \le l$$

타원곡선 상에서는 $Q = (x, y)$에 대해 $-Q = (x, -y)$이므로, 뺄셈이 덧셈과 거의 같은 시간에 수행되므로, ECC에서는 기존에 많이 사용되었던 덧셈 사슬보다 덧셈/뺄셈 사슬이 더 효율적임을 알 수 있다.

그 동안 이루어진 덧셈 사슬에 대한 연구 중에서 비교적 사전 계산량이 적으나 효율적인 알고리즘으로 윈도우 방식(window method)이 있다[3]. 이것은 윈도우 크기가 w인 경우에 n개의 비트열을 길이가 w이면서 최상위 비트와 최하위 비트가 1인 비트열(윈도우)로 나누어지도록 하는 방식이다. 예를 들어서 $n = 0\text{xdae8bf93}_{(h)}$은 $w = 4$인 경우에 다음과 같이 윈도우들로 나뉘어 진다.

$$\underline{1101}\ \underline{101}\ 0\ \underline{111}\ 0\ \underline{1}\ 000\ \underline{1011}\ \underline{1111}\ \underline{1001}\ 00\ \underline{11}$$

위와 같이 윈도우를 설정한 결과를 덧셈 사슬의 연산 순서로 표현하면 다음과 같다. 단, 여기서 d는 $a_{i+1} = a_i + a_i$가 되는 두배 연산(doubling)을 수행함을 의미하고, 숫자값 $\sharp$은 $a_{i+1} = a_i + \sharp$를 수행함을 의미한다.

$$\begin{aligned} a_0 \;=\; & 1101_{(2)} \to d \to d \to d \to 101_{(2)} \to d \to d \to d \to d \to 111_{(2)} \\ & \to d \to d \to 1_{(2)} \to d \to d \to d \to d \to d \to d \to d \\ & \to 1011_{(2)} \to d \to d \to d \to d \to 1111_{(2)} \to d \to d \to d \to d \\ & \to 1001_{(2)} \to d \to d \to d \to d \to 11_{(2)} \end{aligned} \tag{6}$$

위의 알고리즘을 의사 코드(pseudo code)로 정리하면 다음과 같다. n은 L_n비트의 정수이고, n_i를 하위 $n+1$번째 비트의 값이라고 할 때, $n[i:j]$는 $\sum_{k=j}^{i} n_k 2^{k-j}$를 의미한다. 이때에 다음 알고리즘은 식 6과 같은 덧셈 사슬의 연산 순서 a를 출력한다.

```
function Window1(n, w)
i = L_n − 1; j = 0;
while (i ≥ 0) do
    if (i+1<w) then w = i+1;
    temp = n[ i : i−w+1 ];
    i=i-w;
    for (zeros=0; temp is even; zeros=zeros+1 ) do temp = temp >> 1;
    if (j>0) then
        for (k=temp; k>0; k=k>>1) do
            a[ j ]='d'; j=j+1;
    a[ j ]=temp; j=j+1;
    for (; zeros>0; zeros=zeros−1) do
        a[ j ]='d'; j=j+1;
    while (n[ i ]==subtract) do
        a[ j ]='d'; j=j+1;
        if (i==0) then return a;
        i=i−1;
return a;
```

이러한 덧셈 사슬을 덧셈/뺄셈 사슬로 만들어 window의 개수를 줄일 수 있다[28]. 앞서 설명한 $n = 0\text{xdae8bf93}_{(h)}$의 예에서 1번째 윈도우와 2번째 윈도우가 서로 붙어 있다. 두 개의 윈도에만 보면 은 그 값은 $\underline{1101}\,\underline{101} = 1101000 + 101 = 1101000 + 1000 - 1000 + 101 = 1110000 - 11 = \underline{111}00/\underline{11}$ 로 표시할 수 있다. (/의 기호가 앞에 붙는 윈도우는 뺄셈 연산을 수행함을 의미한다.) 이와 마찬가지로 새롭게 분할된 두번째 윈도우와 세번째 윈도우에 대해서도 이와 유사한 방법을 적용할 수 있다.

$$\begin{aligned} /\underline{11}\,0\,\underline{111} &= -110000 + 111 = -110000 + 1000 - 1000 + 111 = -101000 - 1 \\ &= /\underline{101}\,00\,/\underline{1} \end{aligned}$$

이러한 방법으로 다음과 같은 덧셈/뺄셈 윈도우 분할을 얻을 수 있다.

$$\underline{111}\,00\,/\underline{101}\,000\,/\underline{11}\,000\;\;\underline{11}\,000000\,/\underline{111}\,000\,\underline{11}$$

이것을 덧셈/뺄셈 사슬의 연산 순서로 나타내면 다음과 같다.

$$\begin{aligned} a_0 = \; & 111_{(2)} \to d \to d \to d \to d \to d \to -101_{(2)} \to d \to d \to d \\ & \to d \to d \to -11_{(2)} \to d \to d \to d \to d \to d \to 11_{(2)} \to d \\ & \to d \to d \to d \to d \to d \to d \to d \to d \to -111_{(2)} \to d \to d \\ & \to d \to d \to d \to 11_{(2)} \end{aligned} \tag{7}$$

식 (6)과 (7)를 비교하면, 식 (7)의 경우에 사슬의 길이가 더 짧은 것을 알 수 있다. [28]의 의사 코드가 오류가 있을 뿐 아니라, 실제 구현하기에도 어려운 점이 많아서 다음과 같은 새로운 의사 코드를 제안한다.

```
function Window2(n, w)
i = L_n − 1; j = 0; subtract = 0;
while (i≥0) do
    if (i+1<w) then
        w = i+1;
        n[ −1 ]=1−subtract;
    if (subtract==1) then temp = bitwise−complement(n[ i : i-w+1 ]);
    else temp = n[ i : i−w+1 ];
    i=i−w;
    if (n[ i ]==1−subtract) then
        temp = temp +1;
        subtract = 1 − subtract;
    for (zeros=0; temp is even; zeros= zeros+1 ) do temp = temp >> 1;
    if (j>0) then
        for (k=temp;k>0;k=k>>1) do
            a[ j ]='d'; j=j+1;
    if (subtract==1) then a[ j ]= −temp;
    else a[ j ]= temp;
    j=j+1;
    for (; zeros>0; zeros=zeros−1) do
        a[ j ]='d'; j=j+1;
    while (n[ i ]==subtract) do
        a[ j ]='d'; j=j+1;
        if (i==0) then return a;
        i=i−1;
return a;
```

위의 의사 코드는 앞서 설명한 **Window1()**과 같은 역할을 하지만, n에 대한 덧셈/뺄셈 사슬을 구성하여 줌으로써 타원곡선에서 더욱 효율적인 연산이 가능하도록 해준다.

6. 구현 및 결과

이 절에서는 펜티엄을 장착한 PC를 이용한 소프트웨어 구현 결과를 제시한다. 구현을 위해 VC++ 5.0 컴파일러를 사용하였고, C언어만으로 구현하였다.

ECC를 위한 유한체는 [9]에서와 같이 16비트 $p = (2^{16} - 129)$에 대해서 유한체 $F_{p^{11}}$를 선택하고, 이를 위한 minimal polynomial은 $x^{11} - 3$를 사용하였다. 타원곡선의 위수의 가장 큰 소인수 q가 160비트가 되도록 하는 식 (1)의 a와 b는 1과 12를 선택하였다. $F_{(2^{16}-129)^{11}}$에서 정의된 타원곡선 $y^2 = x^3 + x + 12$에 대하여 N과 q는 다음과 같다. 모든 수치 예는 16진수로 표현하였다.

$$\begin{aligned} N &= \text{fa82 e22f7fd1 7cdfe110 3ac0add5 5ba5cb22 e3a9b037} \\ &= 3 \times 551\text{b} \times q \\ q &= \text{fb2e9708 becbea45 063f8029 4a0efbe3 fbfc2507} \end{aligned}$$

소수 q를 위수로 갖는 생성자 G는 알고리즘 2를 통하여 다음과 같이 구해진다.

생성자 $G = (G_x, G_y)$

G_x = (3329, 0847, b9d1, 769e, d34f, 8c22, b6f1, ff27, 3d65, 70d6, e151)

G_y = (1dfe, 7002, 01fd, bb83, f3d4, ee8f, f14f, f657, e66f, f70b, 8be4)

해쉬함수 h로는 HAS-160[27]을 사용하였다.

위와 같은 시스템 변수를 갖는 사용자 A가 다음과 같은 비공개키 x_A를 랜덤하게 선택하였다고 하자.

비공개키 x_A = ccbe31ff e8014804 6181d7d4 3b64881c ee0a854a

이 때, 공개키 Y_A는

$\bar{x}_A = x_A^{-1} \bmod q$ = c0e60653 439c77fb 48d962ad b47c7243 5cc8a9a1

공개키 $Y_A = \bar{x}_A G = (Y_{Ax}, Y_{Ay})$

Y_{Ax} = (b008, 8f58, 71bf, 4963, 7581, e317, 6814, 79f8, 180f, 9915, 625c)

Y_{By} = (ecad, f37d, 8ce2, 8464, be47, 3014, b31a, 8a43, 9648, 9896, 3a30)

와 같이 계산된다. 또한 A의 인증정보를 해쉬하여 얻은 z_A가 다음과 같다고 하자.

인증 정보의 해쉬값 z_A = a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d

메세지 "This is a test message!"에 대한 서명 생성 과정에 대한 예는 다음과 같다.

$$
\begin{aligned}
&k = \text{e119bf05 5ac6036a 008a02ea fae56a9d 96b2285b} \\
&kG = (x_1, y_1) \qquad (8) \\
&\quad x_1 = \text{(a75c, b17c, f785, acb6, e9b0, 47ad, f90f, b425, 1db4, dc4a, 7170)} \\
&\quad y_1 = \text{(b435, 00b7, 5a48, 1fa3, 5948, f822, 71f7, 47e8, 99e0, 5083, e1f8)} \\
&r = h(x_1 \| y_1) = \text{cd4075b3 d0b6a15f 2cd02fa6 6315766b ba0a4b7f} \\
&\text{메세지 } M = \text{"This is a test message!"} \\
&\quad = \text{546869 73206973 20612074 65737420 6d657373 61676521 (as ASCII form)} \\
&H = h(z_A \| M) = \text{6c9f74e8 ad69f6cd b733e1bc dfbb640f 46177ca6} \\
&e = r \oplus H = \text{a1df015b 7ddf5792 9be3ce1a bcae1264 fc1d37d9} \\
&s = x_A(k - e) \bmod q = \text{30232558 4511c0bd 74ad633f fb83b368 789d05a5}
\end{aligned}
$$

메세지 M'과 서명값 (r', s')를 받은 검증자는 r'이 해쉬함수의 출력 길이에 맞는지 s'가 q보다 작은 값인지 확인하고, Y_A와 z_A를 구하여 확인한 후에 다음과 같은 검증 과정을 수행한다.

$$
\begin{aligned}
&h(z_A \| M') = H' = \text{6c9f74e8 ad69f6cd b733e1bc dfbb640f 46177ca6} \\
&e' = r' \oplus H' = \text{a1df015b 7ddf5792 9be3ce1a bcae1264 fc1d37d9} \\
&s'Y_A + e'G = (x_2, y_2) \qquad (9) \\
&\quad x_2 = \text{(a75c, b17c, f785, acb6, e9b0, 47ad, f90f, b425, 1db4, dc4a, 7170)} \\
&\quad y_2 = \text{(b435, 00b7, 5a48, 1fa3, 5948, f822, 71f7, 47e8, 99e0, 5083, e1f8)} \\
&r' \stackrel{?}{=} h(x_2 \| y_2) = \text{cd4075b3 d0b6a15f 2cd02fa6 6315766b ba0a4b7f}
\end{aligned}
$$

서명 생성을 수행하는 데에는 식 (8)에서 대부분의 시간이 소요되며 서명의 검증은 식 (9)에서 대부분의 시간이 소요된다. 이것은 표 2에서 구현 결과를 살펴 보는 것으로도 확인이 가능하다. 표 2에서 G는 미리 알 수 있는 점으로 윈도우에 대한 사전계산을 미리 저장하여 사용하였다. 수행 시간의 측정을 위해 표 2의 s, e는 모두 160비트의 난수를 사용하였다. binary method란, 가장 일반적인 방법으로 sY_A의 계산시에 s의 최상위 비트에서 부터 차례대로 따라 내려오면서 제곱을 수행하고 비트가 1이면 Y_A를 더해 주는 방식을 말한다[5]. signed chain method는 §5.에서 언급한 덧셈/뺄셈 사슬을 이용한 윈도우 방식을 말한다.

한편, sY_A와 eG를 계산하는 데 걸리는 시간의 합보다 $sY_A + eG$를 계산하는 데 걸리는 시간이 더 짧은 것은, 유한체 상의 곱셈에 대한 군 위에서 다항(multiple-term) 멱승을 윈도우를 이용하여 효과적으로 할 수 있었던 방식[2](pp.13-16)을 타원곡선 상의 다항 상수배에 적용하여 계산한 것이기 때문이다. 이 때 사용한 윈도우의 크기는 2이다. 이 방식을 채용한 2항 상수배는 binary방식으로 한 번 곱셈을 수행하는 것보다 더 빨랐다.

실험 플랫폼	Timing (ms)	
	Pentium 133MHz	PentiumII 233MHz
sY_A with binary method	82.08	25.64
sY_A with signed chain method	64.81	21.43
eG with signed chain method	59.60	19.83
$sY_A + eG$	76.90	25.44
서명 생성	59.97	20.03
서명 검증	78.18	25.57

표 2: 타원곡선 점의 반복 덧셈과 EC-KCDSA의 수행 시간

7. 결 론

본 논문에서는 타원곡선 암호시스템의 구현을 위한 타원곡선과 유한체의 선택 방법을 제안하였고, 이러한 방법을 사용하여 타원곡선의 위수를 쉽게 구할 수 있었다. 또한, 선택한 타원곡선에서 생성자 G를 찾는 방법도 살펴보았다. 그 예로서, 타원곡선 E의 파라미터 a와 b, E의 위수 N, N을 나누는 2^{160}비트 크기의 q, 그리고 생성자 G의 값을 보였다. EC-KCDSA의 효율적인 구현을 위해, [9]에서와 같이 소수 p를 $2^{16} - 129$로 두고 $F_{p^{11}}$에서의 유한체 연산을 이용하였다. 타원곡선 위의 점에 대한 효율적인 상수배를 위해서 덧셈/뺄셈 사슬을 이용한 윈도우 방식을 채택하고, 이것의 의사 코드를 제시하였다. 서명 검증 과정의 두 개의 항을 갖는 상수배에 대해서도 윈도우 방식을 적용하여 효율적으로 구현하였다. 133MHz의 Pentium 컴퓨터에서 약 0.060초, 0.078초만에 서명 생성과 검증을 할 수 있었고, 233MHz의 Pentium 컴퓨터에서는 약 0.020초, 0.026초만에 서명의 생성과 검증이 가능하였다.

참 고 문 헌

[1] D. V. Bailey and C. Paar, Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms, *Advances in Cryptology-Crypto'98*, Springer-Verlag, pp.472-485, 1998.

[2] C. H. Lim, *Computational methods for speeding up public key cryptosystem*, Ph. D Thesis, Dept. Electronic and Electrical Engineering, POSTECH, Mar. 1996.

[3] D. M. Gordon, A Survey of Fast Exponentiation Methods, *Journal of Algorithms*, **27**, pp.129-146, 1998.

[4] T. Izu, J. Kogure, M. Noro, and K. Yokoyama, Efficient Implementation of Schoof's Algorithm, *Advances in Cryptology-Asiacrypt'98*, Springer-Verlag, pp.66-79, 1998.

[5] D. Knuth, *The Art of Computer Programming - Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.

[6] N. Koblitz, Elliptic curve cryptosystems, *Mathematics of Computation*, **48**, pp.203-209, 1987.

[7] N. Koblitz, CM-curves with good cryptographic properties, *Advances in Cryptology-Crypto'91*, Springer-Verlag, pp.279-287, 1991.

[8] N. Koblitz, *Algebraic Aspects of Cryptography*, ACM **3**, Springer-Verlag, 1998.

[9] E. J. Lee, D. S. Kim, and P. J. Lee, Speed-up of F_{p^m} Arithmetic for Elliptic Curve Cryptosystem, *Proceeding of ICISC'98*, preprint.

[10] A. Menezes, *Elliptic Curve Public key Cryptosystems*, Kluwer Academic Publishers, 1993.

[11] A. Menezes, editor; Blake et al., *Applications of finite fields*, Kluwer Academic Publishers, 1993.

[12] V. Miller, Uses of elliptic curves in cryptography, *Advances in cryptology - Crypto'85*, Springer-Verlag, pp.417-426, 1986.

[13] F. Morain, Building cyclic elliptic curves modulo large primes, *Advanced in Cryptology - Eurocrypt'91*, Springer-Verlag, pp.328-336, 1991.

[14] A. Menezes, T. Okamoto, and S. A. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field, *Proceedings of the 23rd ACM Symp. on the theory of Computing*, pp.80-89, 1991.

[15] ISO/IEC JTC1/SC27 N2056: Cryptographic Techniques based on Elliptic Curves: Part 2-Digital Signature, 1998.

[16] S. C. Pohlig and M. Hellman, An improved algorithm for computing logarithm over GF(p) and its cryptographic significance, *IEEE Trans. Inform. Theory*, IT-24, pp.106-110, 1978.

[17] J. M. Pollard, Monte Carlo methods for index computations mod p, *Mathematics of Computation*, **32**, pp.918-924, 1978.

[18] T. Satoh and K. Araki, Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, 1997, preprint.

[19] R. Schoof, Elliptic Curves over finite fields and the computation of square root modulo p, *Mathematics of Computation*, **44**, pp.483-494, 1985.

[20] I. Semaev, Evaluation of discrete logarithms on some elliptic curves, to appear in *Mathematics of Computation*, 1997.

[21] J. H. Silverman, *The Arithmetic of Elliptic curves*, Springer-Verlag, 1986.

[22] J. H. Silverman and Joe Suzuki, Elliptic Curve Discrete Logarithms and the Index Calculus, *Advances in Cryptology-Asiacrypt'98*, Springer-Verlag, 1998.

[23] N. Smart, Announcement of an attack on the ECDLP for anomalous elliptic curves, preprint, 1997.

[24] KCDSA Task Force Team, The Korean Certificate-based Digital Signature Algorithm, *http://grouper.ieee.org/groups/1363/addendum.html*, 1998.

[25] American National Standard X9.62-1998, Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm(ECDSA).

[26] American National Standard X9.63-1997, Public Key Cryptography For The Financial Services Industry: Elliptic Curve Key Agreement and Transport Protocols.

[27] 정보보호 표준, 해쉬알고리즘 표준(HAS-160: Hash Algorithm Standard), TTA. IS-10118, 1998.10.27.

[28] 홍성민, 오상엽, 윤현수, 타원곡선 암호시스템에서의 빠른 연산을 위한 새로운 덧셈/뺄셈 사슬 알고리즘, *한국 통신 정보보호 학회 종합 학술발표회 논문집*, **5(1)**, pp.151-162, 1995.