IC 카드용 타원곡선 암호 알고리즘
(Elliptic Curve Cryptography Algorithms for IC Card)

이 택 희 † 서 창 호 †† 김 영 철 ††† 이 태 훈 †††† 윤 보 현 †††††
(TaegHee-Lee) (Chang-Ho Seo) (Young-Cheol Kim) (Tae-Hun Lee) (Bo-Hyun Yun)

요 약 본 논문에서는 IC 카드 용 ECC(Elliptic Curve Cryptography) 및 ECKCDSA(Elliptic Curve KCDSA) 알고리즘 설계, 구현 및 테스트 결과에 대해 기술하고 있다. 타원곡선 암호는 160 비트의 키 길이를 이용하여 현재 사용되는 공개키 암호 알고리즘(RSA)과 동등한 안전도를 제공해준다. 또한, 짧은 키 길이를 사용하기 때문에 작은 메모리와 처리 능력이 제한된 IC 카드나 이동 통신 등과 같은 분야에서 매우 유용하게 사용될 수 있으며, ECC나 ECKCDSA를 자바 카드 상에 구현하여 사용함으로써 사용자들은 보다 강화된 보안성과 안전성을 제공받을 수 있을 것이다.
키워드 : 타원곡선 암호 알고리즘, IC 카드

Abstract This paper describes implementations and test results of Elliptic Curve Cryptography (ECC) and Elliptic Curve KCDSA(ECKCDSA) algorithms based on Java card. 163-Bit ECC guarantees as secure as 1024-Bit Rivest-Shamir-Adleman (RSA) public key algorithm, which has been frequently used until now. According to our test results, 163-bit ECC processing time is about five times fast compared with 1024-bit RSA and amount of resource usages of ECC is smaller than RSA. Therefore, ECC is more appropriate for use on secure devices such as smart cards and wireless devices with constrained computational power consumption and small memory resources.
Key words : Elliptic Curve Cryptography, IC Card

1. 서 론

최근 전자상거래, 전자화폐, 사용자 키관리 등에 높은 안전도를 제공하는 IC 카드는 실리콘이나 플라스틱에 포함된 전자 회로를 통하여 정보를 저장하고 처리하며, 간편하고 부정조작이 어려운 컴퓨터이다. 그리고 IC 카드내에 저장된 정보를 보호하기 위해 저장된 데이타, 물리적 메모리의 프라이버시를 암호화하는 기능이 필수적이다. 아울러 IC 카드와 외부의 교환 데이타는 안전을 위해 서명되고 암호화를 수행하여야 한다.

· 본 연구는 정보통신부 기초기술연구지원사업 ITA-B1220-0401-0087-0001 연구결과로 수행하였음
† 비 회 원 : 호남대학교 정보통신공학과 교수
thlee@honam.ac.kr
†† 정 회 원 : 공주대학교 응용수학과 교수
chseo@kongju.ac.kr
††† 비 회 원 : 전남대학교 전자컴퓨터정보통신공학과 교수
yckim@chonnam.chonnam.ac.kr
†††† 비 회 원 : 광주대학교 컴퓨터전자통신학부 교수
kehan@kongju.ac.kr
††††† 비 회 원 : 목원대학교 컴퓨터교육학과 교수
ybh@mokwon.ac.kr
논문접수 : 2003년 11월 3일
심사완료 : 2004년 6월 1일

그러나 IC 카드에서는 메모리 용량의 한계로 인하여 연산 능력이 부족하다. IC 카드 JCVM(Java Card Virtural Machine) 에서 볼 수 있는 메모리 공간은 최대 64K 바이트이기 때문에 기존에 검증된 암호 알고리즘(RSA, ECC 등)의 적용이 불가능하다. 따라서 메모리 한계를 극복하여 IC 카드내의 정보를 암호화를 위해서는 소프트웨어 방법과 하드웨어 방법으로 구현하여야 한다.

작은 메모리가 장착된 IC 카드내의 정보를 암호화하기 위한 기존의 연구는 거의 없는 실정이지만, 키 크기가 작은 타원곡선 암호알고리즘을 사용한다면 IC카드용 타원곡선 암호 알고리즘을 구현할 수 있을 것이다. 타원곡선 암호시스템(Elliptic Curve Cryptosystem : ECC)은 1985년에 Neal Koblitz[1]와 Victor Miller[2]에 의해서 각각 독립적으로 제안된 암호시스템이다. 타원 곡선 암호시스템의 장점은 유한체의 곱셈군에 근거한 시스템으로써 다양하고 안전한 암호시스템 설계가 용이하며, 존재하는 다른 공개키 스킴과 같은 안전도를 제공하는 데에 더 작은 키 길이를 가진다. 다른 이점은 타원곡선에서의 더하기 연산은 유한체에서의 연산을 포함하므로, H/W와 S/W로 구현하기가 용이하다. 또 다른 이

점으로는 비록 모든 사용자가 같은 기저체를 사용한다 할지라도 각 사용자가 다른 타원 곡선을 선택할 수 있다는 것이다[1].

IC카드용 타원곡선 암호 알고리즘을 소프트웨어 및 하드웨어 구현시 다음과 같은 사항을 고려하여야 한다. 첫째, 스칼라 곱의 성능을 높이는 것이다. 어떤 기저(basis)를 사용하는가에 상관없이 타원곡선의 한 점 P의 n배인 nP를 어떻게 빠른 시간 안에 구현하느냐가 중요하다. 둘째, 소프트웨어 구현시 지역변수를 최소화하여 최적화하여야 한다. IC 카드의 메모리는 64K 바이트이기 때문에 다양한 지역변수를 이용하여 선언된 암호알고리즘을 최소의 지역변수를 이용하는 암호알고리즘으로 최적화할 수 있어야 한다. 셋째, 하드웨어 구현시 최소의 게이트를 사용하여 IC 카드 칩의 크기를 최소화하여야 한다. 칩 구현시 소용되는 비용을 줄이기 위해서는 최소의 게이트를 사용하여야 한다.

본 논문에서는 위의 세가지 고려사항을 만족할 수 있는 IC 카드용 타원곡선암호시스템을 설계 및 구현한다. 스칼라 곱의 성능을 높이기 위해서 Modified Almost Inverse Algorithm을 이용하고, 중복된 지역변수를 최소화하며, 연산 모듈에서 가장 중요한 부분인 역수 연산기 설계시 10,000개의 게이트만을 이용하여 구현한다. 본 논문은 다음과 같이 구성된다. 2장에서는 자바 카드 환경과 타원곡선 암호 시스템에 관하여 기술한다. 그리고 3장에서는 IC 카드용 암호 알고리즘 연산 모듈 프로세서 설계 및 타원곡선 암호 알고리즘 구현에 관하여 설명한다. 마지막으로, 4장에서 결론을 맺는다.

2. 타원곡선 암호 시스템

타원곡선 암호 알고리즘은 Diffie-Hellman 키 교환, DSS, 등에서 구현된 작은 키 길이로 현재 사용되는 RSA 공개키 스킴과 동등한 안전도를 제공한다. 짧은 키 길이를 갖는다는 것은 대역폭과 메모리가 작아짐을 뜻하므로 메모리와 처리 능력이 제한된 IC 카드나 이동통신 같은 응용에서 매우 유용하게 된다.

타원곡선 암호 알고리즘은 Prime field 와 Binary field와 같은 유한체 위에서 구현되는데, 어떠한 유한체를 사용하느냐에 따라서 연산의 많은 부분이 달라지게 된다. 하지만 어떤 유한체를 사용하든지 상관없이 타원곡선 암호에서 가장 중요한 문제는 타원곡선 상의 한 점 P에서 nP(n은 임의의 정수)를 빨리 구하는 것이다. 이 때, nP 값은 완벽한 타원곡선 군을 이루어야 하면, 그러기 위해 각 유한체를 위한 타원곡선과 타원곡선상의 덧셈연산이 정의되어야 한다.

타원곡선의 이산 대수 문제는 유한체 위에 정의된 타원곡선 E가 정의되어있을 때, 정수 a 배의 관계에 있는 타원곡선 위의 두 점 P, Q를 모두 안다고 하더라도 정수 a를 계산하는 것이 어렵다는 점을 이용한 것이다.

$$a*P=Q\ (\ P,Q \in E(F_q)) \qquad (1)$$

수식 (1)의 P와 Q는 타원곡선 위의 좌표 점을 의미하며 소문자로 표기된 a는 타원곡선의 점이 아닌 일반적인 정수를 의미한다. 연산자 "*"는 일반적인 사칙연산이 아니라 타원곡선 위의 점의 배수 연산이다. 타원곡선 암호의 주연산은 타원곡선 위의 점 P를 a번 곱하는 배수 연산이다. 이 배수 연산은 서로 다른 두 점을 더하는 덧셈과 한 점을 두 배하는 더블 연산으로 구할 수 있다.

일반적인 이산대수 문제와 비교하였을 때 매우 유사하게 보인다. 일반적인 이산 대수 문제는 유한체 F_q와 g, h $\in$ F_q가 주어졌을 때 g^a = h를 만족하는 정수 a를 구하는 문제로서 g를 a번 곱하는 곱셈의 문제이고, 타원곡선 이산대수 문제는 타원곡선 상의 점 P를 a번 더하는 덧셈의 문제이다. 하지만 덧셈과 곱셈의 차이를 제외시킨다면 두 문제는 추상적으로 유사하게 보이고 타원곡선 이산대수 문제가 해결하기 더 쉬워 보이나 실질적으로는 타원곡선 이산대수 문제의 해결이 더 어렵다. 일반적인 이산대수문제는 두 개의 대수적 연산인 덧셈과 곱셈을 가지고 있다. 이것은 덧셈을 이용한 Index-calculus method로서 부분지수 복잡도(Sub-exponential time complexity)를 가지는 알고리즘을 찾을 수 있다. 하지만 타원곡선 이산대수 문제는 타원곡선상의 두 점에 대한 덧셈이라는 연산만을 가지고 있으므로 초특이 타원곡선(Super-singular elliptic curve)을 제외하고는 Index-calculus method를 사용할 수 없다. 그러므로 타원곡선 암호 시스템에서 초특이 타원곡선을 제외하면 완전지수 복잡도(Full-exponential time complexity)를 가지는 시스템을 구현할 수 있다. 현재 공개키 암호 시스템 중에서 가장 널리 사용되고 있는 RSA 시스템은 sub-exponential time 알고리즘이 존재한다[1].

일반적으로, 공개키 암호시스템으로는 소인수 분해의 어려움에 근거한 시스템(RSA)[4]과 이산대수 문제의 어려움에 근거한 시스템(DSA), 그리고 타원곡선상의 이산대수 문제에 근거한 시스템(ECC)이 주로 사용된다. 이 중 타원곡선 암호시스템 비트당 안전도가 가장 높은 암호 시스템으로 작은 사이즈의 키 값만으로도 높은 안전성을 보장한다. 표 1은 타원곡선 암호시스템(ECC), RSA, DSA의 비트당 안전성을 비교한 표이다[3].

모듈러 지수승 연산이 RSA 암호시스템의 성능을 좌우하듯이 타원곡선 암호시스템의 성능은 스칼라 곱셈 연산에 의하여 좌우된다. 스칼라 곱셈 연산은 임의의 랜덤수 k와 타원 곡선 위의 한점 P의 곱셈 연산으로 정의되며, 타원곡선 위의 점 P의 k번 덧셈연산으로 계산

표 1 RSA, DSA, ECC 안전성 비교

Time to break in MIPS years	RSA/DSA key size	ECC key size	RSA/ECC key size ratio
104	512	106	5 : 1
108	768	132	6 : 1
1011	1,024	160	7 : 1
1020	2,048	210	10 : 1
1078	21,000	600	35 : 1

된다. 이때 타원곡선 의 덧셈연산은 결과값이 다시 타원곡선 위의 점이 되도록 [알고리즘 1]과 같이 정의되어야 한다. (단, Polynomial 기반 유한체를 위한 타원곡선 식은 y2 + xy = x3 +ax2 + b과 같이 주어지며, P1과 P2이 타원곡선 상의 존재한다.)

[알고리즘 1] Point Addition Equation

Input : $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$.
Output : $P_3 = P_1 + P_2 = (x_3, y_3)$.

1. If $P_1 = P_2$ (doubling)
   $x_3 = \lambda^2 + \lambda + a$,
   $y_3 = x_1^2 + \lambda(\lambda + 1)\ x_3$
   where $(\lambda = x_1 + y_1 / x_1)$
2. Else if $P_1 \neq P_2$ (point addition)
   $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$,
   $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$
   where $(\lambda = (y_2 + y_1) / (x_2 + x_1))$
3. Return (x_3, y_3)

[알고리즘 1]에서 보는 바와 같이 Prime field에서 두 점의 더하기 연산을 수행하기 위해서는 더하기 연산, 곱하기 연산, 역수 연산, 제곱 연산 등이 필요한데 이를 위한 연산들은 RSA[2]에서 사용되는 기본 연산과 동일하다.

3. 설계 및 구현

3.1 구현 환경

ECC 및 ECKCDSA가 구현된 카드는 현재 IC 카드로써 Java Card 2.1, 2.2 규격을 따르고 있으며, 32-bit 마이크로프로세서인 ARM7TDMI를 사용한다. 또한, 다양한 암호 알고리즘을 처리할 수 있도록 설계되었으며, 그림 1은 카드에 구현된 암호 알고리즘의 종류를 보여준다.

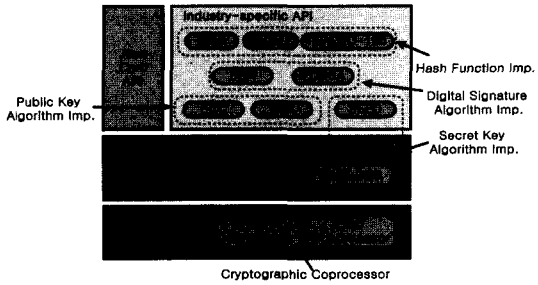


그림 1 IC카드 내 암호 알고리즘 구조

그림 1에서 보는 바와 같이, IC 카드는 해쉬 함수로 MD5, SHA-1, RIPEMD-160, 대칭 키 알고리즘으로 SEED, T-DES, 비대칭 키 알고리즘으로 RSA, ECC, 그리고 서명 알고리즘으로 RSA 서명, ECDSA, DSA를 구현하고 있다. 비대칭 키 알고리즘인 RSA와 ECC는 보다 신속한 처리를 위해 H/W 상에 코프로세서 형태로 구현되어 있으며, RSA 및 ECC 코프로세서를 애플릿이 이용할 수 있도록 API-JCVM-COS 간의 인터페이스가 구현되어 있다. 따라서, API 상위에 탑재되는 애플릿은 ECC 암호 API 인터페이스를 이용하여, ECDSA, ECKCDSA를 이용할 수 있게 된다.

3.2 타원곡선 암호 연산모듈 프로세서 설계

본 논문에서는 SEC2[6]에서 제안된 163비트 다항식 기저(polynomial basis) 타원곡선 암호시스템을 위한 암호 프로세서를 설계하였다. 163비트 타원곡선 암호시스템은 표 1에서 보는 바와 같이 RSA 1024비트 이상의 안정성을 보장하게 된다. 본 타원곡선 암호시스템의 파라미터는 표 2와 같다. 이와 같이 정의된 타원곡선 암호시스템을 위하여 그림 2와 같은 구조를 가지는 암호프로세서를 설계하였다. 이 암호프로세서는 타원곡선의 스

표 2 $GF(2^{163})$ 타원곡선 암호시스템 파라미터

Reduction Polynomial $f(x)$	$x^{163} + x^7 + x^6 + x^3 + 1$
Parameter a	01
Parameter b	02 0A601907 B8C953CA 141EB10 512F7874 4A3205FD
Order of a base point n	04 00000000 00000000 000292FE 77E70C12 A4234C33
x coordinate of base point	03 F0EBA162 86A2D57E A0991168 D4994637 E8343E36
y coordinate of base point	D51FBC6C 71A0094F A2CDD545 B11C5C0C 797424F1

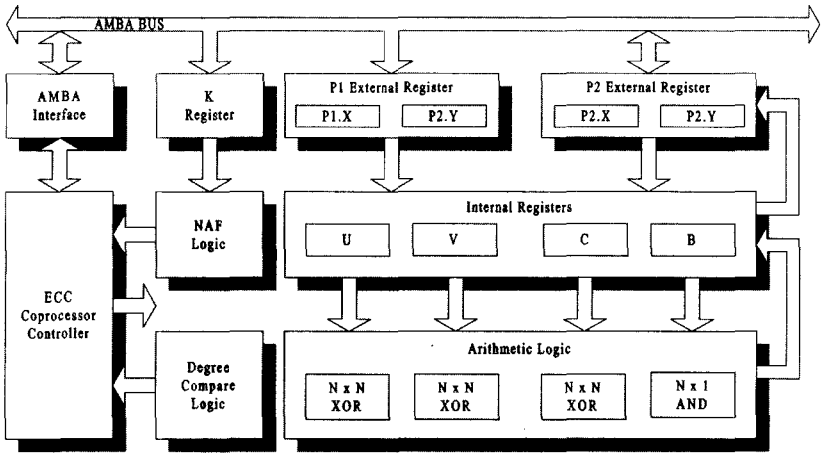


그림 2 타원곡선 암호프로세서 구조

칼라 곱셈(kP 연산)과 타원곡선 덧셈 연산을 수행한다. (엄밀히 말하면 스칼라 곱셈 연산은 타원곡선 덧셈 연산들의 조합이지만, 두 가지 연산의 입출력이 다르므로 서로 구분 지었다.) 실제 상위 프로토콜에서는 이러한 두 가지 연산만을 필요로 하므로, 연산에 필요한 유한체 곱셈, 역승산, 덧셈, 제곱과 같은 연산은 내부적으로만 수행하도록 되어 있다.

타원곡선 덧셈연산을 위한 [알고리즘 1]에서 가장 주요한 연산은 유한체 곱셈연산과 역승산 연산이다. 본 타원곡선 암호프로세서에서는 곱셈연산은 shift-and-add method를 이용하여 구현하였으며, 역승산 연산은 Modified Almost Inverse Algorithm(MAIA)으로 구현하였다. [알고리즘 1]에서 알 수 있듯이 타원곡선 덧셈연산을 위해서는 유한체 덧셈 연산과 제곱 연산도 필요하지만, 다항식 기반 유한체의 특성상 유한체 덧셈 연산은 두 수의 xor 연산으로 쉽게 구현될 수 있다. (실제로 [알고리즘 1]에서 존재하는 덧셈 연산들은 제어기에서 그림 3의 내부 XOR 연산기들과 레지스터들을 적절히 이용하여 수행된다.) 그리고 유한체 제곱 연산은 연산

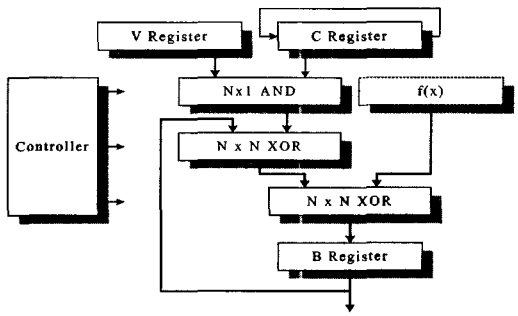


그림 3 유한체 곱셈기 동작 블록도

자체는 유한체 곱셈연산보다 매우 단순하지만 하드웨어적인 구현에서는 별도의 유한체 제곱 연산기를 설계하는 것과 유한체 곱셈연산기의 두 입력에 같은 값을 두 번 입력하여 연산하나 별 차이가 없게 된다. 타원곡선 스칼라 곱셈 연산을 위해서는 binary NAF method가 이용되었다.

3.2.1 유한체 곱셈 연산

다항식 기반 유한체의 곱셈 연산 알고리즘으로는 shift-and-add method, comb method, comb method with windows, partial(parallel) method 등 다양한 알고리즘들이 존재한다[5]. 이중 comb method와 comb method with windows는 쉬프트 연산을 줄이거나, 필요한 연산을 미리 수행하여 메모리에 저장하는 방식으로 연산 속도를 증가시키는 방법으로 소프트웨어적인 구현에서 효율적으로 사용된다.

본 타원곡선 암호프로세서에서 구현알고리즘으로 선택한 shift-and-add method는 가장 일반적인 유한체 곱셈 연산 알고리즘이다. 이는 $a \cdot b \bmod f(x)$를 계산하면 다음과 같다.

$$a \cdot b \bmod f(x) = (a_{m-1} x^{m-1} b + \ldots + a_2 x^2 b + a_1 x b + a_0 b) \bmod f(x) \quad (2)$$

$$= (a_{m-1} x^{m-1} b \bmod f(x)) + \ldots + (a_2 x^2 b \bmod f(x)) + (a_1 x b \bmod f(x)) + (a_0 b \bmod f(x))$$ 이다.

식 (2)와 연산될 수 있음을 이용한 알고리즘으로 덧셈 연산(알고리즘 2의 2.1)과 동시에 reduction 연산(알고리즘 2의 2.2)을 수행할 수 있다. 이러한 구조는 하드웨어로의 구현이 매우 용이하며, 비트 길이만큼의 연산 시간만이 소요된다(만약 163비트의 타원곡선 암호시스템이라면 곱셈연산은 163 클럭이 필요하게 된다. 단, 입출력 시간은 제외되었다.).

알고리즘 2. Shift-and-add field multiplication

Input : Binary Polynomials $a(x)$ and $b(x)$.
Output : $c(x) = a(x) \cdot b(x) \bmod f(x)$.
1. $c \leftarrow 0$.
2. For i from $m-1$ to 1 do
   2.1 If $a_i = 1$ then $c \leftarrow c + b$.
   2.2 $c \leftarrow c \cdot x \bmod f(x)$.
3. If $a_0 = 1$ then $c \leftarrow c + b$.
4. Return (c).

이러한 구조의 유한체 곱셈 연산기는 뒤에서 언급될 역승산기와 쉽게 병합이 가능하며, 임의의 타원곡선을 모두 수용할 수 있도록 확장이 용이하다는 장점이 있다. 본 타원곡선 암호프로세서는 곱셈 연산을 위해서 그림 2의 암호프로세서 구조도에서 v, c, b 레지스터들과 두 개의 NxN XOR 연산기와 Nx1 AND 연산기가 사용된다. 곱셈 연산시 동작 블록도는 그림 3과 같다(그림 3의 블록도는 다른 유한체 연산을 위한 mux 신호는 생략되었다). 앞에서 언급된 바와 같이 본 하드웨어는 변수들이 고정되어 있어서 reduction 함수 f(x) 값을 암호프로세서에 이미 내장하고 있지만, 암호 프로세서가 동작하기 전에 함수 값을 직접 받아들이도록 하면 일정 비트 길이 내의 임의의 타원곡선 암호시스템을 수용할 수 있는 구조로 쉽게 확장할 수 있다.

3.2.2 유한체 역승산

다항식 기반 유한체의 역승산 연산 알고리즘으로는 Extended Euclidean Algorithm (EEA)[6]과 Almost Inverse Algorithm (AIA)[7] 등이 있다. 소프트웨어적인 구현에서는 두 가지 알고리즘이 유사한 성능을 보인다. 하지만 하드웨어적인 구현에서는 한 비트씩 연산을 수행하는 AIA가 더 용이하다. 알고리즘 3은 AIA을 수정하여 작성한 Modified Almost Inverse Algorithm (MANIA) 알고리즘이다.

본 타원곡선 암호프로세서는 역승산 연산을 위해서

알고리즘 3. Modified Almost Inverse Algorithm

Input : $a \in F_2^m$, $a \neq 0$.
Output : $a^{-1} \bmod f(x)$.
1. $b \leftarrow 1$, $c \leftarrow 0$, $u \leftarrow a$, $v \leftarrow f$, $k \leftarrow 0$.
2. While x divides u do :
   2.1 $u \leftarrow u / x$.
   2.2 If x divides b then $b \leftarrow b / x$;
   else $b \leftarrow (b + f) / x$.
3. If $u = 1$ the return (b).
4. If $\deg(u) < \deg(v)$ then : u ↔ v, b ↔ c.
5. $u \leftarrow u + v$, $b \leftarrow b + c$.
6. Goto step 2.

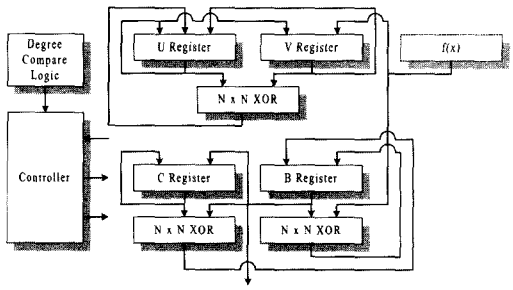


그림 4 유한체 역승산기 동작 블록도

그림 2의 암호프로세서 구조도에서 u, v, c, b 레지스터들과 3개의 NxN XOR 연산기, 그리고 degree compare logic이 이용된다. 역승산기의 동작 블록도는 그림 4와 같다.

본 논문에서 제안하는 IC 카드용 타원곡선암호 알고리즘 구현시 효율성 측면에서 MAIA 알고리즘을 적용하여 하드웨어로 구성하여 효율성을 증가시켰다. 더욱이 이 알고리즘 Modified Almost Inverse Algorithm (MAIA)은 reduction 연산이 따로 필요 없어 매우 효율적이다.

3.2.3 타원곡선 스칼라 곱셈 연산

만약 $k = 1111_{(2)} = 15$이면 이는 $(10000-1)_2 = 16-1$로 다시 표현할 수 있다. 만약 "-"에 대한 연산을 정의할 수 있다면 이런 방식의 표현 기법은 타원곡선 스칼라 곱셈연산의 연산 횟수를 줄일 수 있다(이와 같은 표현을 Non Adjacent Format(NAF)라 한다). 다행히 타원곡선 상에서 $-P = (x, x + y)$로 정의되며, 이러한 NAF방식을 이용한, 스칼라 곱셈 알고리즘은 다음과 같다.

알고리즘 4. Binary NAF method

Input : $\mathrm{NAF}(k) = \sum_{i=0}^{l-1} k_i \cdot 2^i$, $P \in E(F_2^m)$.
Output : kP.
1. $Q \leftarrow 0$.
2. For i from $t-1$ downto 0 do
   2.1 $Q \leftarrow 2Q$.
   2.2 If $k_i = 1$ then $Q \leftarrow Q + P$.
   2.3 If $k_i = 1$ then $Q \leftarrow Q - P$.
3. Return (Q).

본 암호프로세서 NAF 로직에서 NAF 변환을 수행하며, 알고리즘 4의 연산 스케줄링은 제어기에서 담당한다. 실제 NAF 로직은 매우 단순한 형태로 구현이 되었으며, 이와 같은 방법을 이용하여 163비트의 nP(P :

base point, n : P의 order) 계산 시, 20MHz 클럭에서 연산시간을 11.9msec으로 단축할 수 있었다. 이는 단순 binary method로 15.6msec이 소요된다.

3.3 타원곡선 암호 알고리즘 구현

3.3.1 ECC 인터페이스

SUN사에서 제공되는 Java Card 2.1 API 규격에는 ECC와 관련된 인터페이스가 정의되어 있지 않기 때문에, 개발된 ECC 암호 모듈과 상위 애플릿 사이에 주고받는 정보 및 비밀키, 공개키 등에 대한 인터페이스를 지원할 수 있는 ECC 암호 API 및 API-VM-COS간의 인터페이스를 설계하였다. 그림 5는 ECC 알고리즘을 카드 상에서 수행하기 위해 설계된 인터페이스 및 각 단계(layer)간에 입·출력되는 파라미터를 보여준다[9-11].

3.3.2 ECC 암·복호화

ECC 암호화의 경우, 카드는 상대방의 공개키 ($Q=k_BP$)를 가져와서 자신이 생성한 난수 k_A를 이용하여 k_Ak_BP를 계산한다. 이때, 스칼라곱(scalar multiplication)은 ECC 암호 모듈이 담당한다. P는 타원 곡선상의 x, y 좌표로 이루어진 점이므로, 입력 메시지를 k_Ak_BP와 더하기 위해서는 입력 메시지도 타원 곡선상의 한 점으로 매핑(P_M)되어야 한다. 즉, 이러한 모든 작업은 OS와 H/W 상에서 발생되며, 모든 암호화 연산이 끝난 후 OS는 암호화 메시지로 $(k_AP, P_M+k_A(k_BP))$를 VM에게 반환해 준다.

암호화와 달리 복호화의 경우는 입력메시지가 $(k_BP, P_M+k_B(k_AP))$이기 때문에 카드는 입력메시지의 k_BP를 추출하여 비밀키 k_A와 함께 k_Ak_BP를 계산해내고, 계산된 k_Ak_BP값을 입력 메시지에서 감소시킨 후, P_M으로부터 실제 입력 메시지 M을 계산해낸다. 여기서도 OS와 ECC 암호 모듈이 각각의 기능을 담당하게 되며, 모든 복호화 연산이 끝난 후 OS는 평문 M을 VM에게 반환해준다. 물론, ECC 암·복호화 수행 시 타원곡선을 결정해주는 상수 값, 위수, 기약다항식 등도 파라미터로 사용되어야 하지만, 개발된 ECC 암호 모듈은 타원곡선에 필요한 변수 값을 미리 결정해 놓은 상태에서 암·복호화를 수행하기 때문에 현재 실험 환경에서는 카드 상에서 ECC 테스트 시 외부로부터 변수 값을 입력받지 않도록 설계하였다.

3.3.3 ECKCDSA 구현

ECKCDSA 역시 카드 상에서 구현하기 위해 ECC와 비슷한 형태로 암호 API 인터페이스를 설계하였으며, 카드에 이미 구현된 SHA-1 알고리즘을 사용한다. ECKCDSA 구현을 위한 API-VM-OS 간의 인터페이스 및 입·출력되는 파라미터와 파라미터 크기를 그림 5에 나타내었다[1].

ECKCDSA 서명 생성의 경우, 카드는 자신의 비밀키

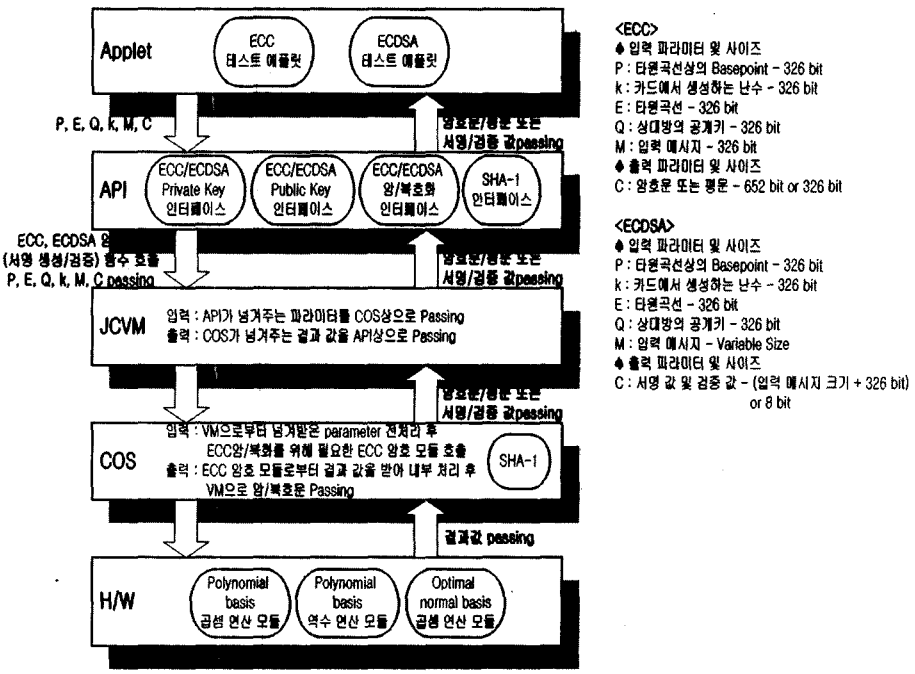


그림 5 타원곡선 암호 알고리즘 인터페이스

(d)를 이용하여 dP를 계산하고, SHA-1 알고리즘을 이용하여 메시지 M을 160 비트 해쉬 값으로 변환한 후, ECC 암호 모듈을 이용하여 서명 값인 r과 s를 생성해 낸다. 생성된 서명 값 ($M \| r \| s$)은 OS를 통해 VM에게 전달한다. 서명 생성과 달리 서명 검증의 경우, 입력메시지가 ($M \| r \| s$)형태이기 때문에 OS는 입력메시지에서 r과 s를 추출해낸 후, 상대방의 공개키 Q를 사용하여 v를 계산해낸다. 계산된 v값과 r값이 일치하면 검증이 성공적으로 이루어진 경우이므로 카드는 0x00이라는 값을 호스트에게 보내준다. 서명 검증 시에도 역시 OS와 ECC 암호 모듈이 각각의 기능을 담당하게 된다.

3.3.4 실험 및 결과

테스트를 위해 개발된 에뮬레이터 보드 및 접촉·비접촉 I/O 통신을 위한 단말기를 실험 장비로 사용하여 ECC와 ECKCDSA를 테스트하였다. ECC와 ECKCDSA 테스트 시 사용될 ECC 암호 모듈은 에뮬레이터 보드 상의 Xilinx 칩에 FPGA 코드로 다운로드 되어있으며, 테스트를 위해 사용되는 파라미터와 파라미터 크기는 그림 5의 우측에 자세히 제시하였다. 테스트 절차는 아래와 같다.

우선, ECC, ECKCDSA 테스트 애플릿을 자바 프로그램으로 작성한 후, Windows NT상에서 ECC/ECKCDSA 암호 API와 함께 컴파일, converting을 수행 한 후, 결과 파일을 에뮬레이터 보드와 연결된 PC로 이동시킨다. 에뮬레이터 보드 상에 탑재 될 테스트 파일을 생성하기 위해, 자바 카드 개발을 위해 구현된 COS와 JCVM 파일들과 함께 결과파일을 다시 컴파일 한 후, 에뮬레이터 보드에 테스트 파일을 다운로드 하여 ECC 및 ECKCDSA 테스트를 수행하였다. 그림 6은 이러한 테스트 수행 환경을 간략히 보여준다.

테스트 수행 시 카드와 호스트 사이에 주고받는 command 및 response APDU 구조는 그림 7과 같으며, 그림 8은 ECC 암·복호화 테스트 절차를 보여준다.

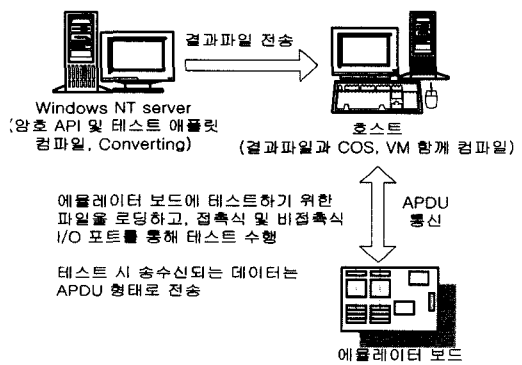


그림 6 테스트 환경

Command APDU

CLA	INS	P1	P2	Lc/Le	DATA

Response APDU

DATA	SW1	SW2

그림 7 APDU 데이타 형태

```
// 입력 메시지
0xe0 0x10 0x01 0x00 0x18 000000000001010101010101010101010101010101010101
0xe0 0x10 0x02 0x00 0x18 000000000101010101010101010101010101010061766364

// Basepoint P
0xe0 0x20 0x01 0x00 0x18 00000000369979697ab43897789566789567f787a7876a654
0xe0 0x20 0x02 0x00 0x18 00000000435edb42efafb2989d51fefce3c80988f41ff883

// 상대방의 공개 키 Q
0xe0 0x30 0x01 0x00 0x18 00000000461476015a13441ae072f29dc0fa1fce23c2223a0
0xe0 0x30 0x02 0x00 0x18 0000000065ead6d1519db913e6f719709d51399d9dd636cd

// 카드에서 사용하는 랜덤 변수
0xe0 0x50 0x00 0x00 0x18 0000000091c9dc3a946a9206b48710cb56e1698980cbca09
0xe0 0x60 0x00 0x00 0x18 000000011b2419eddcfe099a68cfda7af8e9be219114aebd

// 암호화를 위해 공개 키 생성
command : 0xe0 0xb0 0x00 0x00 0x00  response : 90 00

// ECC 암호화 호출
command : 0xe0 0xa0 0x00 0x00 0x00  response : 90 00

// 카드로부터 암호문 수신
command : 0xe0 0xc0 0x01 0x00 0x00 0x18
response : 00000000[illegible]3356C93F403F1F2C1C6BC 90 00
command : 0xe0 0xc0 0x02 0x00 0x00 0x18
response : 00000000[illegible]48A5B23C0FB3BC6E3BFD 90 00
command : 0xe0 0xc0 0x03 0x00 0x00 0x18
response : 00000000[illegible]1CD062A212FD17935071 90 00
command : 0xe0 0xc0 0x04 0x00 0x00 0x18
response : 00000000[illegible]E11455F2AF277F74DCD6 90 00

// 복호화를 위해 비밀 키 생성
command : 0xe0 0x70 0x00 0x00 0x00  response : 90 00

// ECC 복호화 호출
command : 0xe0 0xb0 0x00 0x00 0x00  response : 90 00

// 카드로부터 복호문 수신 (입력 메시지와 동일)
command : 0xe0 0xc0 0x01 0x00 0x00 0x18
response : 000000000001010101010101010101010101010101010101 90 00
command : 0xe0 0xc0 0x02 0x00 0x00 0x18
response : 00000000010101010101010101010101010101010061766364 90 00
```

그림 8 ECC 암·복호화 테스트 절차

Command APDU(Application Protocol Data Unit)에서 CLA는 테스트 애플릿에서 사용하는 class 값을, INS는 애플릿에서 수행되는 명령어의 종류를 나타내며, Lc는 데이타의 길이를, Le는 응답 APDU의 데이타 길이를 의미한다. Command APDU는 호스트가 카드에게 명령 요청을 할 때 사용된다. 반면에 Response APDU의 SW(Status Word)는 카드가 명령어를 수행 한 후 결과 상태를 호스트에게 알려주는 데 사용된다. 보통 90 00은 에러 없이 완벽하게 카드가 수행되었음을 나타낸다.

표 3에는 에뮬레이터 보드 상에서 테스트한 ECC, ECKCDSA 알고리즘의 수행 소요 시간을 명시하였다. ECKCDSA는 ECC에 비해 내부적으로 처리해야 할 작업이 많기 때문에 수행 시간이 ECC보다 추가됨을 알 수 있다. 현재 ECC 테스트 시, 입력되는 메시지를 타원곡선 상의 점으로 매핑시키는 함수를 구현하지 않고 직접 곡선상의 점을 입력으로 넣어 측정하였다.

표 3 수행 소요 시간 비교

	수행 소요 시간 (msec)	
	암호(서명생성)	복호(서명검증)
ECC	43.244	46.828
ECKCDSA	101.297	211.126

본 논문에서 제안하는 IC카드용 타원곡선 암호 알고리즘의 설계 및 구현 방법은 다음과 같다. 첫째, 스칼라 곱의 성능을 높이는 것이다. 어떤 기저(basis)를 사용하는가에 상관없이 타원곡선의 한 점 P의 n배인 nP를 어떻게 빠른 시간 안에 구현할 수 있다. 둘째, 소프트웨어 구현시 지역변수를 최소화하여 최적화하여야 한다. IC 카드의 메모리는 64K 바이트이기 때문에 다양한 지역변수를 이용하여 선언된 암호알고리즘을 최소의 지역변수를 이용하는 암호알고리즘으로 최적화할 수 있다. 셋째, 하드웨어 구현시 최소의 게이트를 사용하여 IC 카드 칩의 크기를 최소화한다. 칩 구현시 소용되는 비용을 줄이기 위해서는 최소의 게이트를 사용한다.

4. 결 론

본 논문에서는 SEC2에서 제안된 163비트 다항식 기저(polynomial basis) 타원곡선 암호시스템을 위한 암호 알고리즘 연산 모듈 프로세서를 설계하였다. 본 타원곡선 암호 연산 프로세서는 단순 입출력 인터페이스 소프트웨어만으로 타원곡선 스칼라 곱셈 연산과 덧셈 연산을 수행할 수 있다. 본 암호프로세서는 xillinx FPGA에서 20MHz로 동작하며, HYNDAI 0.5m CMOS 공정으로 AMBA 인터페이스까지 포함하여 24k gates의 면적을 가진다.

본 타원곡선 암호시스템은 여러 가지 구현 알고리즘을 비교 분석하여 IC 카드 메모리에 적합한 구조를 택하였다. 현재 IC 카드 상에 ECC 및 ECKCDSA 구현을 위한 암호 API 설계및 테스트 절차, 테스트 결과에 대해 언급하였다. 타원 곡선 암호는 163 비트의 짧은 키를 이용하기 때문에 공개키 방식의 암호 시스템으로서 메모리나 자원이 제한된 IC 카드에 매우 적합하게 구현하였으며, 향후 전자상거래 및 인터넷 보안을 위해 사용될 IC 카드의 안전성을 한층 더 높여줄 것이다.

참 고 문 헌

[1] http://java.sun.com/products/javacard

[2] R.L. Rivest, A. Shamir, and L.M. Adleman, *"A method for obtaining digital ignatures and public-key cryptosystems,"* Communications of the ACM, volume 21, pp.120-126, February 1978.

[3] Certicom research, *The Elliptic Curve Cryptosystem*, Certicom, April 1997.

[4] Certicom research, *"SEC 2 : Recommended Elliptic Curve Domain Parameters,"* October 1999.

[5] Chen, Zhiqun, *Java Card Technology for Smart Cards*, Addison-wesley, 2002.

[6] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, Handbook of Applied Cryptography, CRC press, 1997.

[7] Richard Schroeppel, Hilarie Orman, Sean O'Malley, *"Fast Key Exchange with Elliptic Curve Systems,"* TR-95-03(Tucson, AZ: University of Arizona), Computer Sciences Department, 1995.

[8] Michael Caentsch, "Java Card-From Hype to Reality," IEEE Concurrency, pp.36-43, Oct.-Dec., 1999.

[9] Sun Microsystems, Inc., *Java Card Applet Developers Guide*, August 30, 1999.

[10] Sun Microsystems, Inc., *Java Card™ 2.1 Application Programming Interface*, June 7, 1999.

[11] Sun MicroSystems, Inc. *Java Card™ 2.1.2 Development Kit User's Guide*, 2002.

[12] Tommi Elo, "A Software Implementation of ECDSA on a Java Smart Card," Master's Thesis, Helsinki University of Technology, March 2000.

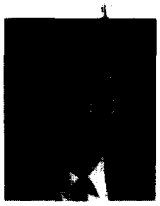
[13] Sun Microsystems, Inc., *Java Card Applet Developers Guide*, August 30, 1999.

[14] IEEE P1363a: Standard Specifications for Public-Key Cryptography: Additional Techniques Draft 9, 2001.

[15] Darrel Hankerson, Julio Lopez Hernandez, Alfred Menezes, *"Software Implementation of Elliptic Curve Cryptography over Binary Fields,"* CHES 2000, pp.1-24. 2000.

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

[17] W. Rnakl, W. Effing, Smart Card Handbook, JOHN WILEY & SONS, 1999.



이 택 희

1981년 전남대학교 제어공학과(학사). 1983년 서울대학교 제어계측공학과(석사). 1984년~1995년 한국전자통신연구원 선임연구원. 1989년~1991년 Belgium Alcatel Bell 연구소 파견 연구원. 1995년~현재 호남대학교 정보통신공학과 부교수. 관심분야는 embedded system, field bus networking, realtime O.S., 통신망 보안 등

서 창 호

1990년 고려대학교 수학과 졸업(학사) 1992년 고려대학교 일반대학원 수학과(이학석사). 1996년 고려대학교 일반대학원 수학과(이학박사). 1996년~1997년 국방과학연구소 선임연구원. 1997년~2000년 한국전자통신연구원 선임연구원, 팀장. 2000년~현재 공주대학교 응용수학과 부교수. 관심분야는 암호 알고리즘, PKI, 시스템 보안, 무선 보안 등

김 영 철

1981년 한양대학교 전자공학과 공학사. 1987년 University of Detroit, EE, 공학석사. 1993년 Michigan State University, EE, 공학박사. 1993년~현재 전남대학교 전자컴퓨터정보통신공학과 부교수. 2000년~현재 전남대학교 반도체설계교육센터 소장. 관심분야는 초고속통신망, ASIC/SoC

이 태 훈

1982년 한국항공대학교 전자공학과 졸업. 1984년 아주대학교 대학원 전자공학과 공학석사. 1999년 아주대학교 대학원 전자공학과 공학박사. 1984년 한국전자통신연구원 연구원. 1993년~현재 광주대학교 컴퓨터전자통신학부 부교수. 관심분야는 멀티미디어 통신 및 서비스, 홈 네트워크 미들웨어

윤 보 현

1999년 고려대학교 컴퓨터학과(이학박사). 1999년~2003년 한국전자통신연구원(ETRI) 선임연구원 및 팀장. 2001년~2003년 한국소프트웨어산업협회 언어정보산업협의회 운영위원. 2002년~2002년 광인터넷 기술정책 자문위원. 2003년~2004년 한국전자통신연구원(ETRI) 초빙연구원. 2004년 KRnet 컨퍼런스 운영위원. 2003년~현재 목원대학교 컴퓨터교육과 교수