자바 카드 기반 ECC/ECDSA 암호 알고리즘 구현

한진희*, 김영진*, 전성익*, 서창호**

* ETRI IC카드OS연구팀, **공주대학교 응용수학과

Implementation of ECC/ECDSA Cryptography Algorithms based on Java Card

Jin-Hee Han*, Young-Jin Kim*, Seong-Ik Jun* and Chang-Ho Seo*

*IC Card OS Research Team, ETRI.

**Department of Mathematics, Kongju National Univ.

요 약

본 논문에서는 자바 카드용 ECC (Elliptic Curve Cryptosystems) 및 ECDSA (Elliptic Curve Digital Signature Algorithm) 알고리즘 구현 및 시험 결과에 대해 언급하고자 한다. 163비트 타원곡선 암호시스템(ECC)은 현재 많이 사용되고 있는 RSA 1024 비트 이상의 안전성을 보장한다. 또한, 짧은 키 길이를 사용하기 때문에 메모리와 처리능력이 제한된 스마트 카드나 이동 통신 등과 같은 분야에서 매우 유용하게 사용될 수 있으며, ECC나 ECDSA를 자바 카드 상에 구현하여 사용함으로써 사용자들은 보다 강화된 보안성과 안전성을 제공받을 수 있을 것이다.

I. 서론

1. 자바 카드 (Java Card)

자바 카드는 1996년 3월 스마트 카드 (Smart Card) 제조 업체인 Schlumberger 사에 의해 처음 소개되었으며, 이는 자바 기술을 스마트 카드 기술에 접목시킨 것이다. 처음에 출시된 자바 카드는 축소된 기능의 자바 바이트 코드 인터프리터 (Java Bytecode Interpreter)를 COS (Card Operating System)에 탑재하여 변형된 자바 클래스 파일을 다운로드 하여 사용하였으며, 1996년 10월 Sun Microsystems 사가 Schlumberger 사의 개발 경험을 바탕으로 자바 카드의 구현 목표 및 구조와 자바 언어 서브셋 (subset), 그리고 암호 API (Application Programming Interface) 규격 등을 기록한 초기 자바 카드 규격을 공표 하였다.

이후로, 1997년 말에는 보다 구체적이고 실질적인 자바 카드 규격으로 Java Card 2.0 규격이 발표되었고, 1999년에 Java Card 2.1 규격이 발표되었으며, Java Card 2.2 규격이 곧 발표될 예정이다[1][5][8].

자바 카드는 H/W와 밀접하게 연관되는 COS 위에 플랫폼 독립성을 제공해주는 JCVM (Java Card Virtual Machine)이 탑재되며, 그 위에 API가 위치하게 된다. 자바 카드는 개방형 구조로서 다양한 애플릿들이 API 위에 탑재될 수 있다. 그림 1은 자바 카드 구조를 나타낸다.

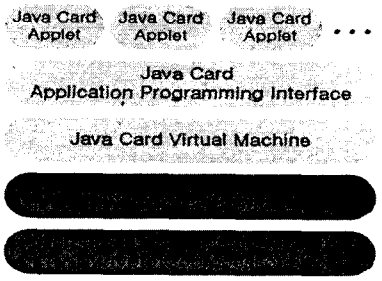


그림 1: 자바 카드 구조

2. 타원곡선 암호시스템(ECC)

유한체 (Finite Fields) 위에서 정의된 타원 곡선 군에서의 이산대수 문제에 기초한 타원 곡선 암호 시스템은 1985년 Koblitz와 Miller에 의해 처음 제안된 이후 활발히 연구되고 있으며, 비트 당

안전도가 타 공개키 시스템보다 효율적이라고 알려져 있다[2].

타원 곡선 암호시스템의 장점은 유한체 연산에 근거한 시스템으로써 다양하고 안전한 암호시스템 설계가 용이하며, 타원 곡선에서의 더하기 연산은 유한체에서의 연산을 포함하므로, H/W와 S/W로 구현하기가 용이하다는 장점을 가지고 있다. 또 다른 이점으로는 비록 모든 사용자가 같은 기저체를 사용한다 할 지라도 각 사용자가 다른 타원 곡선을 선택할 수 있다는 것이다. 즉, 모든 사용자는 유한체 연산을 수행하기 위해 같은 H/W를 사용할 수 있으며, 추가 보안을 위해 주기적으로 타원 곡선을 변경 할 수 있다.

II. 자바 카드용 ECC/ECDSA 구현

1. 카드 환경

ECC 및 ECDSA가 구현된 카드는 현재 개발 단계에 있는 차세대 IC 카드로써 Java Card 2.1 규격을 따르고 있으며, 32-bit 마이크로프로세서인 ARM7TDMI를 사용한다. 또한, 다양한 암호 알고리즘을 처리할 수 있도록 설계되었으며, 그림 2는 카드에 구현된 암호 알고리즘의 종류를 보여준다.

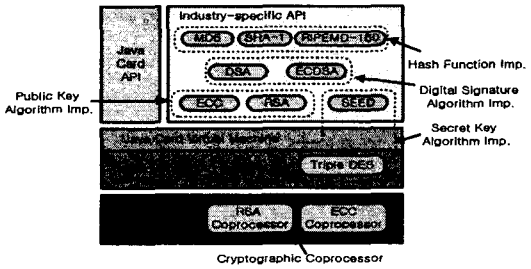


그림 2: 차세대 IC카드 내 암호 알고리즘

그림 2에서 보는 바와 같이 개발 중인 차세대 IC 카드는 해쉬 함수로 MD5, SHA-1, RIPEMD -160, 대칭 키 알고리즘으로 SEED, T-DES, 공개키 알고리즘으로 RSA, ECC, 그리고 서명 알고리즘으로 RSA 서명, ECDSA, DSA를 구현하고있다. 공개 키 알고리즘인 RSA와 ECC는 보다 신속한 처리를 위해 H/W 상에 코프로세서 형태로 구현[주] [9] 되어 있으며, RSA 및 ECC 코프로세서를 애플릿이 이용할 수 있도록 API-JCVM-COS 간의 인터페이스가 구현되어 있다. 따라서, API 상위에 탑재되는 애플릿은 RSA 및 ECC 암호 API 인터페이스를 이용하여 RSA와 ECC 등 암호 알고리즘들을 이용할 수 있다.

주) RSA, ECC 코프로세서는 ETRI에서 진행중인 차세대 IC카드 과제 결과물로서 IC카드 구조팀에서 개발하였다.

2. ECC 구현

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 argument 및 비밀키, 공개키 등에 대한 인터페이스를 지원할 수 있는 ECC 암호 API 및 API-VM-COS간의 인터페이스를 설계하였다. (그림 3)은 ECC 알고리즘을 카드 상에서 수행하기 위해 설계된 인터페이스 및 각 layer 간에 입·출력되는 파라미터를 보여준다[4][6][7].

2) ECC 암·복호화

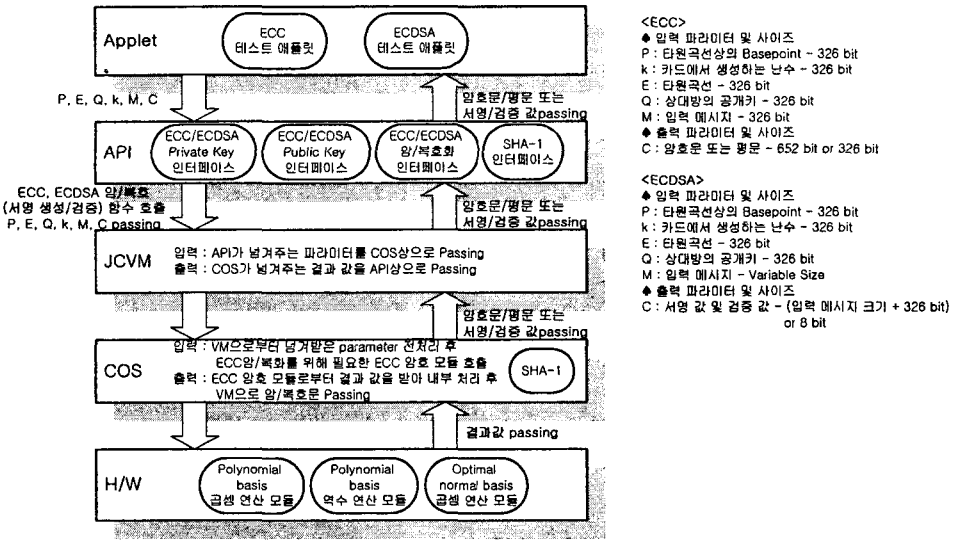
ECC 암호화의 경우, 카드는 상대방의 공개키 ($Q=k_BP$)를 가져와서 자신이 생성한 난수 k_A를 이용하여 k_Ak_BP를 계산한다. 이때, 상수 배(scalar multiplication) 연산은 ECC 암호 모듈이 담당한다. P는 타원 곡선상의 x, y 좌표로 이루어진 점이므로, 입력 메시지를 k_Ak_BP와 더하기 위해서는 입력 메시지도 타원 곡선상의 한 점으로 매핑(P_M)되어야 한다. 즉, 이러한 모든 작업은 OS와 H/W 상의 암호모듈에서 수행되며, 모든 연산이 끝난 후 OS는 암호화 메시지 (k_AP, $P_M+k_A(k_BP)$)를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

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_BP 값을 입력 메시지에서 감소시킨 후, P_M으로부터 실제 입력 메시지 M을 계산해낸다. 여기서도 OS와 ECC 암호 모듈이 각각의 기능을 수행하며, 모든 복호화 연산이 끝난 후 OS는 평문 M을 VM에게 반환해준다.

물론, ECC 암·복호화 수행 시 타원 곡선을 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. ECDSA 구현

ECDSA 역시 카드 상에서 구현하기 위해 ECC와 비슷한 형태로 암호 API 인터페이스를 설계하였으며, 카드에 이미 구현된 SHA-1[2] 알고리즘



<ECC>
♠ 입력 파라미터 및 사이즈
P : 타원곡선상의 Basepoint - 326 bit
k : 카드에서 생성하는 난수 - 326 bit
E : 타원곡선 - 326 bit
Q : 상대방의 공개키 - 326 bit
M : 입력 메시지 - 326 bit
♠ 출력 파라미터 및 사이즈
C : 암호문 또는 평문 - 652 bit or 326 bit

<ECDSA>
♠ 입력 파라미터 및 사이즈
P : 타원곡선상의 Basepoint - 326 bit
k : 카드에서 생성하는 난수 - 326 bit
E : 타원곡선 - 326 bit
Q : 상대방의 공개키 - 326 bit
M : 입력 메시지 - Variable Size
♠ 출력 파라미터 및 사이즈
C : 서명 값 및 검증 값 - (입력 메시지 크기 + 326 bit) or 8 bit

그림 3: ECC 및 ECDSA 인터페이스

을 사용한다. ECDSA 구현을 위한 API-VM-OS 간의 인터페이스 및 입·출력되는 파라미터와 파라미터 크기를 그림 3에 나타내었다[3].

ECDSA 서명 생성의 경우, 카드는 자신의 비밀키(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 생성해낸다. 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 수행하게 된다.

4. 실험 및 결과

테스트를 위해 개발된 에뮬레이터 보드 및 접촉·비접촉 I/O 통신을 위한 단말기를 실험 장비로 사용하여 ECC와 ECDSA를 테스트하였다.

ECC와 ECDSA 테스트 시 사용될 ECC 암호 모듈은 에뮬레이터 보드 상의 Xilinx 칩에 FPGA 코드로 다운로드 되어있으며, 테스트를 위해 사용되는 파라미터와 파라미터 크기는 그림 3의 우측에 제시하였다. 테스트 절차는 아래와 같다.

우선, ECC, ECDSA 테스트 애플릿을 자바 프로그램으로 작성 한 후, Windows NT상에서 ECC/ECDSA 암호 API와 함께 컴파일, converting을 수행 한 후, 결과 파일을 에뮬레이터 보드와 연결된 PC로 이동시킨다. 에뮬레이터 보드 상에 탑재될 테스트 파일을 생성하기 위해, 자바 카드 개발을 위해 구현된 COS와 JCVM 파일들과 함께 결과 파일을 다시 컴파일 한 후, 에뮬레이터 보드에 테스트 파일을 다운로드 하여 ECC 및 ECDSA 테스트를 수행하였다. 그림 4는 이러한 테스트 수행 환경을 간략히 보여준다.

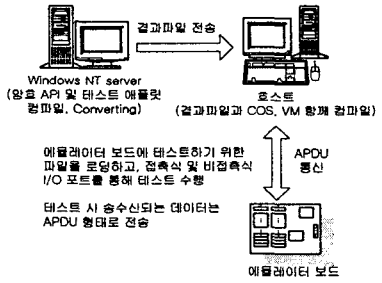


그림 4: 테스트 환경

테스트 수행 시 카드와 호스트 사이에 주고받는 command 및 response APDU 구조는 그림 5와 같으며, 그림 6은 ECC 암·복호화 테스트 절차를 보여준다.

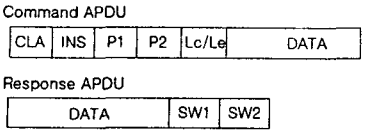


그림 5: APDU 데이터 형태

Command APDU (Application Protocol Data Unit)에서 CLA는 테스트 애플릿에서 사용하는 class 값을, INS는 애플릿에서 수행되는 명령어의 종류를 나타내며, Lc는 데이터의 길이를, Le는 응답 APDU의 데이터 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.

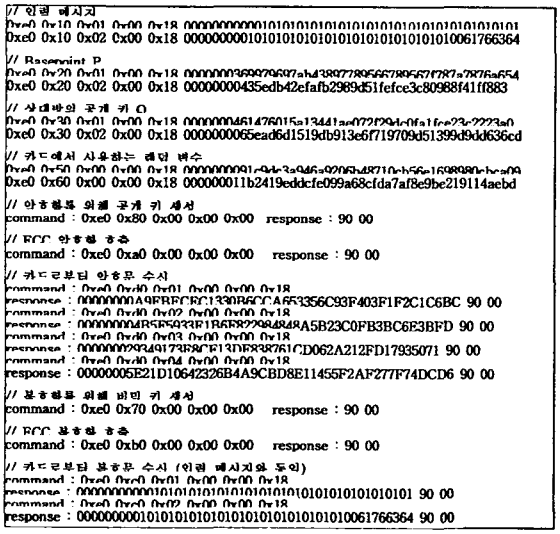


그림 6: ECC 암·복호화 테스트 절차

표 1에는 에뮬레이터 보드 상에서 테스트 한 ECC, ECDSA 알고리즘의 수행 소요 시간을 명시하였다. ECDSA는 ECC에 비해 내부적으로 처리해야 할 작업이 많기 때문에 수행 시간이 ECC 보다 추가됨을 알 수 있다.

현재 ECC 테스트 시, 입력되는 메시지를 타원곡선 상의 점으로 매핑시키는 함수를 구현하지 않고 직접 곡선상의 점을 입력으로 넣어 측정하였는데, 이 부분은 추후 개선할 예정이다.

표 1: 수행 소요 시간 비교

	수행 소요 시간 (msec)	
	암호(서명생성)	복호(서명검증)
ECC	43.244	46.828
ECDSA	101.297	241.526

III. 결론

본 논문에서는 현재 개발중인 차세대 IC 카드 상에 ECC 및 ECDSA 구현을 위한 암호 API 설계, 테스트 절차 및 테스트 결과에 대해 언급하였다.

타원 곡선 암호시스템은 163 비트의 짧은 키를 이용하기 때문에 공개키 방식의 암호 시스템으로서 메모리나 자원이 제한된 스마트 카드에 매우 적합하며, 향후 전자상거래 및 인터넷 보안을 위해 사용될 스마트 카드의 안전성을 보다 더 높여 줄 것이다.

본 논문에서 구현한 ECC 및 ECDSA는 향후 전자 상거래 및 이동통신 서비스에서 이용할 수 있도록 지속적으로 보완·수정되어야 한다.

참고문헌

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

[2] Menezed, A., van Oorschot, P., Vanstone, S., *Handbook of Applied Cryptography*, CRC Press, 1997.

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

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

[5] Chen, Zhiqun, *Java Card Technology for Smart Cards*, Addison-wesley, 2000

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

[7] Sun MicroSystems, Inc. *Java Card™ 2.1.1 Development Kit User's Guide"*, 2001.

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

[9] 최용제, 김호원, 김무섭, 박영수, " IC카드를 위한 Polynomial 기반의 타원곡선 암호시스템 연산기 설계", 대한전자공학회 하계학술대회, pp. 3305-3308, 2001.6.