자바카드 기반 공개키 암호 API를 위한 임의의 정수 클래스 설계 및 구현

김 성 준[†] · 이 희 규[†] · 조 한 진[††] · 이 재 광[†††]

요 약

자바카드 API는 한정된 메모리를 가진 스마트 카드 기반의 프로그램을 개발할 때 많은 이점을 제공한다. 그러나 공개키 암호 알고리즘 구현에 반드시 필요한 연산들인 모듈러 지수 연산, 최대공약수 계산, 그리고 소수 판정과 생성 등의 연산을 지원하지 않는다. 본 논문에서는 자바 카드에서 공개키 암호 알고리즘 구현을 위해서 반드시 필요한 연산들을 지원하는 임의의 정수 클래스의 설계 및 구현하였다.

Design and Implementation of Arbitrary Precision Class for Public Key Crypto API based on Java Card

Sung-Jun Kim[†] · Hee Kyu Lee[†] · Han-Jin Cho[††] · Jae-Kwang Lee[†††]

ABSTRACT

Java Card API porvide benifit for development program based on smart card using limmited resource. This APIs does not support arithmetic operations such as modular arithmetic, greatest common divisor calculation, and generation and certification of prime number, which is necessary arithmetic in PKI algorithm implementation. In this paper, we implement class BigInteger acted in the Java Card platform because that Java Card APIs does not support class BigInteger necessary in implementation of PKI algorithm.

키워드 : 자바카드(Java Card), 정수처리(BigInteger), 스마트 카드(Smart Card)

1. 서 론

실용적이고 효과적인 정보보호 서비스를 제공하기 위해서 스마트 카드의 사용이 급증하고 있으며, 이와 관련한 기술 개발이 활발히 이루어지고 있다[1]. 개인용 컴퓨터나 금융망, 행정망 및 의료망 등에서 정보보호에 스마트 카드를 사용하는 기술이 이미 일부 국가에서는 실용화 단계에 있으나 국내에서는 실용화를 위한 준비단계에 있다.

스마트 카드를 사용하는 가장 큰 목적은 카드 내에 저장된 데이터를 안전하게 보호하는 일이다. 지금까지 스마트 카드는 일반적인 컴퓨터 시스템에 대한 정보보호 예방 기술만을 제공해 왔으나, 안전한 인터넷 언어라고 알려진 자바 언어를 스마트 카드에 적용한 자바 카드는 스마트 카드의 정보보호 특성을 그대로 보존할 뿐만 아니라, 여러 사람에게 스마트 카드를 위한 프로그래밍 기술을 공개해줌으로써 스마트 카드 자체를 인터넷을 위한 하나의 새로운 응용 플랫폼으로 활용할 수 있도록 하였다.

자바 카드를 비롯한 스마트 카드는 인터넷 프로그래머들에게 개인 암호 키와 같은 비밀 정보를 안전하게 생성하고 저장해줄 수 있는 공간을 제공해준다. 즉, 개인 프라이버시 보장을 위한 서명 및 인증 기능과 기존에 서로 만나서 행해야 했던 계약, 양방향 서명 기능 등을 원격으로 그리고 좀 더 안전하게 행할 수 있게 해준다. 뿐만 아니라 이동성 측면에서 볼 때 개인 암호 키와 같은 비밀 정보를 자신의 PC에 저장하는 것보다는 자바 카드나 스마트카드에 저장하는 것이 훨씬 유리하므로 전자상거래 및 암호 기술 발전과 더불어 이들 카드를 PC와 전자상거래로 연결시키려는 움직임이 활성화되고 있다.

자바 카드용 어플리케이션을 개발할 때 사용하는 자바 카드 API는 공개키 암호 알고리즘의 구현에 반드시 필요한 모듈러 연산, 지수 연산과 승산역원 연산 등을 지원하지 않는다. BigInteger 클래스는 JDK에 포함되어 있으며, JDK가

† 준 회 원 : 한남대학교 대학원 컴퓨터공학과
†† 정 회 원 : 극동대학교 정보통신학부 컴퓨터전공 전임강사
††† 종신회원 : 한남대학교 컴퓨터공학과 교수
논문접수 : 2001년 11월 5일, 심사완료 : 2001년 12월 18일

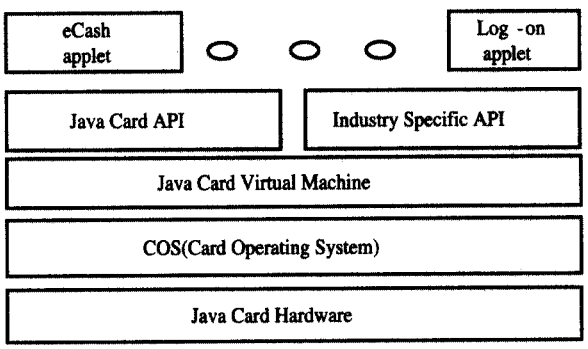
오픈 소스이기 때문에 소스코드가 공개되어 있다. 그러나, JDK1.1에서는 C-native의 형태로 구현되어있으며, JDK1.2 버전 이후부터는 100% 자바로 구현되어 있지만, 자바 카드에서는 적용될 수 없는 다중 상속의 개념을 이용하여 구현되어 있다.

본 논문에서는 자바 카드 상에서 공개키 암호 알고리즘 구현에 반드시 필요한 여러 가지 연산을 지원하는 클래스를 설계하고 구현하였다. 2장에서는 자바카드의 기본 구조와 동작 과정을 살펴보고, 3장에서는 클래스의 설계와 적용된 알고리즘을 설명하고, 그리고 4장에서는 클래스를 구현하고, 구현된 클래스의 동작을 검증하였다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

2.1 자바 카드(JavaCard)

자바 카드란 스마트 카드 기술을 기반으로 하여 자바의 기술을 접목시킨 것으로 (그림 1)과 같이 COS(Card Operating System) 위에 JCVM(Java Card Virtual Machine)이 랩핑(Wrapping)되어 있는 구조의 IC 카드를 말한다[2].



(그림 1) 자바 카드 구조

자바 카드 API는 자바 카드 상의 JCVM상에서 자바를 이용한 응용 소프트웨어 개발에 필요한 API들을 정의한 것이다. 이것은 스마트 카드의 보안성을 연구하던 Schlumberger사의 연구팀에 의해 1996년에 소개되었다. 이후 발표된 자바 카드 API 1.0은 단지 명세서의 역할만을 했다. 그러나 1997년 Sun Microsystem사에서 자바API의 일부 제한된 기능을 수행하는 자바 카드 API 2.0을 발표하였다[3]. 그후 계속 발전하여 현재 2.1.2버전까지 개발되어 있는 상태이다. 자바 카드 API는 전자상거래, 네트워크 접근, 인증을 위한 차세대 네트워크 기술을 제시하였다. Bull, Gemplus, Schulmberger 등 전 세계 스마트카드 제조 회사의 90%이상이 자바 카드의 개발을 위해 라이센스를 이미 받은 상태이다[4].

자바 카드 API는 스마트 카드와 같은 작은 메모리를 가진 임베디드 장치를 위한 프로그래밍에 필요한 패키지와 클래스만을 정의하고 있다. 또한 국제 표준인 ISO7816과 산업 명세 표준인 EMV와 서로 호환된다.

자바 카드는 스마트 카드 기술에 자바의 기술을 접목 시켰기 때문에 다음과 같은 특징들을 제공한다[5].

- 플랫폼 독립성 - 자바 카드의 JCAE(Java Card Application Environment)를 기반으로 동작하도록 작성된 애플릿은 서로 다른 업체에서 생산된 자바 카드들에서 동작할 수 있다.
- 복수 응용 프로그램 - 하나 이상의 응용 프로그램이 하나의 카드 상에서 동작할 수 있다. 자바는 상속구조와 코드의 다운로드 가능성을 가지고 있기 때문에, 보다 쉽게 하나의 카드 상에서 복수개의 응용프로그램을 안전하게 실행할 수 있다.
- 응용 프로그램의 갱신 - 카드가 발급된 후에 응용프로그램을 설치 할 수 있다. 사용자는 카드를 발급 받은 이후에 변경되는 응용 프로그램들을 보다 쉽게 갱신 또는 추가 할 수 있다. 다시 말해, 새로운 응용 프로그램들이 설치된 카드를 추가적으로 발급 받는 것이 아니라 이미 존재하는 카드의 응용프로그램만을 변경하거나 추가함으로써 새로운 서비스를 이용할 수 있다.
- 융통성 - 자바 카드 기술의 객체 지향 기술은 스마트 카드 프로그래밍에 융통성을 제공한다.
- 호환성 - 자바 카드는 국제 표준인 ISO7816과 산업 표준인 EMV와 호환된다.

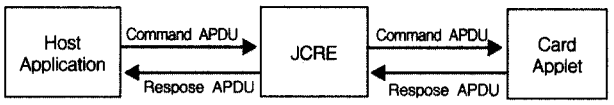
2.2 자바 카드 애플릿

자바 카드 애플릿은 자바 카드 상에서 실행될 수 있는 자바 프로그램이다[3]. 자바 카드 애플릿을 일반 자바 애플릿과 달리 브라우져 환경에서는 실행 될 수 없다. 자바 응용 프로그램과 달리 애플릿은 카드의 ROM에 설치될 필요가 없고, 단지 카드 상에 다운로드 함으로써 사용이 가능하게 된다. 자바 카드 애플릿의 특징은 다음과 같다.

- 자바 카드 런-타임 환경에서 수행된다.
- APDU(Application Program Data Unit)교환을 통해 JCRE와 통신한다.
- AID(Application IDentifier)에 의해 식별된다.
- 카드 상에 동적으로 다운로드 될 수 있다.

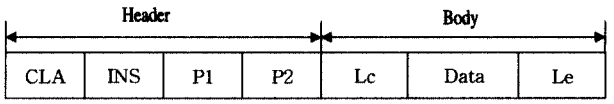
애플릿과 호스트간의 통신은 (그림 2)에서 나타낸 바와 같이 명령어 APDU와 응답 APDU로 구성되는 APDU 교환을 통해서 이루어진다. APDU 교환은 애플릿과 호스트간에 직접 이루어지는 것이 아니라 JCRE를 매개로 하여 이루어지고, JCRE는 애플릿과 호스트간에 교환되는 APDU의 관리와 감독 역할을 수행한다. 따라서 애플릿과 CAD(Card

Access Device) 또는 호스트간의 직접적인 통신은 불가능하며, JCRE를 통한 통신만이 가능하다[3].

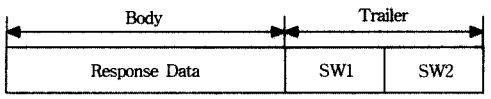


(그림 2) 애플릿 통신

APDU는 카드 상의 통신에서 사용되는 전송 메시지의 형태로 ISO7816에 규정되어 있다. 전송방식은 명령(Command)와 응답(Response)으로 이루어져 있다. 아래 (그림 3)와 (그림 4)는 APDU의 구조를 살펴본 것이다.



(그림 3) 명령(Command) APDU 구조



(그림 4) 응답(Response) ADPU 구조

3. 클래스의 설계

3.1 BigInteger 클래스

JDK 1.1에서 임의의 정수를 표현하기 위해서 java.math.BigInteger라는 클래스를 도입하였다[6]. 이때에는 연산의 핵심 부분이 C-native의 형태로 구현되었으며, JDK1.2에 들어서 기존의 C-native로 구현된 부분을 100% 순수 자바로 구현하였다. 그러나, 순수 자바로 구현하면서 여러 개의 클래스들을 이용하여 다중 상속의 개념을 이용하여 구현되었다. 이 클래스에서 모든 연산들은 2의 보수의 형태로 처리된다. 또한 기본 연산들 이외에 추가적으로 모듈러 연산, 최대공약수 계산, 소수판정 및 생성과 비트 연산 등을 제공한다. BigInteger 클래스가 대부분의 암호 알고리즘의 구현에 필요한 큰 정수들간의 모듈러 연산과 최대공약수 계산 등 유용한 연산을 지원하기 때문에 Java를 이용한 암호알고리즘의 구현이 용이하다. 그러나 자바 카드 API에서는 BigInteger 클래스를 지원하지 않기 때문에 암호알고리즘을 구현하기 위해서는 별도의 라이브러리는 사용해야만 한다.

본 논문에서는 JDK1.3의 BigInteger 클래스를 기반으로 하여 자바 카드에서 동작하는 클래스를 구현하였다. 구현한 클래스의 메소드들은 호환성을 위해 JDK1.3에서 제공되는 클래스와 동일한 이름을 사용하였으며 암호알고리즘의 구현에 불필요한 메소드들과 자바 카드에서 지원되는 자료형에 관련된 연산들은 제외하였다. 구현된 메소드들 중에서 응용연산 중 일부만을 살펴본다.

3.2 산술 연산

임의로 결정되어질 수 있는 수를 표현하기 위해서 모든 값들은 바이트 배열의 형태로 저장되고 부호를 저장하는 변수가 별도로 존재하게 된다. 실제 값이 저장되는 예를 보면 <표 1>과 같다.

〈표 1〉 BigInteger의 수 표현 예제

10진수	16진수 (2 바이트)	BigInteger 표현값	
		부 호	값
1000	03 E8	1	03 E8
-1000	FC 18	-1	03 E8
0	00 00	0	0

앞의 <표 1>과 같이 BigInteger 클래스에서는 표현되는 모든 값은 16진수 바이트 배열로 표현되며, 양수와 음수의 차이는 단지 부호 변수만이 차이가 있고 실제 저장되는 바이트 배열은 동일한 것을 볼 수 있다.

3.2.1 덧 셈

```
알고리즘 구성
INPUT : 두수의 바이트 배열 표현식 a > b
OUTPUT : a와 b의 합
        1. i ← a의 하위 바이트, j ← b의 하위 바이트
        2. while b의 첨자 > 0 do
           2.1 sum ← i + j + (sum >>>8)
           2.2 result ← (sum & 0x00ff)
        3. result를 반환
```

두 수의 덧셈에서 각 수는 바이트 배열의 형태로 저장된다. 이때 두 개의 바이트 배열에서 하위 한 바이트를 short형 변수에 각각 할당한 후에 short형을 덧셈을 취한다. 더한 결과를 오른쪽으로 8비트 쉬프트한 값이 0이 아니면, 자리올림이 발생한 것이다. 그러면 그 자리올림도 덧셈에 반영시킨다.

3.2.2 뺄 셈

```
알고리즘 구성
INPUT : a > b인 두 수
OUTPUT : a와 b의 차
        1. i ← a의 하위 바이트, j ← b의 하위 바이트
        2. while b의 첨자 > 0 do
           2.1 diff ←i - j + (diff >>8)
           2.2 result ← (diff & 0xff)
        3. result를 반환
```

두 수의 뺄셈에서 각 수는 바이트 배열의 형태로 저장된다. 이때 두 개의 바이트 배열에서 하위 한 바이트를 short

형 변수에 각각 할당한 후에 short형을 뺄셈을 취한다. 연산 결과를 오른쪽으로 8비트 쉬프트한 값이 0이 아니면, 빌림이 발생한 것이다. 그러면 그 자리올림도 뺄셈에 반영시킨다.

3.2.3 곱 셈

알고리즘 구성
INPUT : a > b인 두 수 OUTPUT : a와 b의 곱 1. i ← a의 하위 바이트, j ← b의 하위 바이트 2. while b의 첨자 > 0 do 2.1 product ← i × j + carry 2.2 result ← (product & 0xff) 2.3 carry ← (product >>>8) 3. result를 반환

두 수의 곱셈에서 각 수는 바이트 배열의 형태로 저장된다. 이때 두 개의 바이트 배열에서 하위 한 바이트를 short형 변수에 각각 할당한 후에 short형을 곱셈을 취한다. 곱한 결과를 오른쪽으로 8비트 쉬프트한 값이 0이 아니면, 자리올림이 발생한 것이다. 그러면 그 자리올림도 곱셈에 반영시킨다. 곱셈의 결과로 생성된 바이트 배열의 길이는 최대 두 입력 바이트 배열의 길이를 더한 것과 같다.

3.2.4 나눗셈

알고리즘 구성
INPUT : a > b인 두 수 OUTPUT : a를 b로 나눈 몫 1. rem ← a, m ← b 2. while rem > m do 2.1 m >> 1, s>>1 3. while rem >= b do 3.1 while rem < m do 3.1.1 m << 1, s <<1 3.2 rem ← rem - m , retval ← retval + s 4. retval 반환

두 수의 나눗셈은 일반적으로 쉬프트와 뺄셈을 이용해서 수행한다. 이때 두 수의 자릿수가 일치하지 않을 때는 자릿수를 동일하게 한 후에 쉬프트와 뺄셈을 수행한다. 앞의 알고리즘에서는 retval에 나눈 몫이 저장되고 rem에는 나머지가 저장되게 된다.

3.3 응용연산

공개키 암호 알고리즘의 구현에 필요한 모듈러 연산, 지수연산, 최대공약수 계산 및 소수 판정과 생성은 계산량이 사칙연산에 비해 월등히 많다. 뿐만 아니라 계산된 결과 또한 자바 카드에서는 표현할 수 있는 최대 자료형인 정수의 범위를 훨씬 벗어나게 된다. 따라서 응용연산을 구현할 때 미리 구현한 BigInteger 클래스의 사칙연산을 이용하여 구현하였다.

3.3.1 최대공약수 연산

두 수의 최대공약수를 계산하기 위해서 Euclidean 알고리즘을 이용하였다. Euclidean 알고리즘은 정수의 인수분해를 사용하지 않고 두수의 최대공약수를 구하는데 유용하게 사용되는 알고리즘이다.

알고리즘 구성
INPUT : 두 정수 a, b $0 < b \le a$ OUTPUT : a와 b의 최대 공약수 1. While $b \le 0$ do 1.1 r ← a mod b, a ← b, b ← r. 2. a를 반환

만약 $a = p_1^{e1}p_2^{e2}\cdots p_k^{ek}$이고 $b = p_1^{f1}p_2^{f2}\cdots p_2^{fk}$이고, $e_i \ge 0$와 $f_i \ge$ 이면, 두 정수 a와 b의 최대공약수 gcd는 $p_1^{\min(e1,f1)}$ $p_2^{\min(e2,f2)}\cdots p_k^{\min(ek,fk)}$로 계산할 수 있다. 그러나 이러한 방법으로 계산을 하는 것은 정수의 인수분해 문제가 상대적으로 어렵기 때문에 효율적인 알고리즘이 아니다. 반면에 Euclidean 알고리즘은 정수의 인수분해를 필요로 하지 않기 때문에 두 정수의 최대공약수를 구하는데 효율적인 알고리즘이다. 이 알고리즘은 만약 a와 b가 양의 정수이고 a > b이면, 최대 공약수 gcd는 gcd(a,b) = gcd(b, a mod b)를 만족한다는 사실에 근거를 둔다[7].

3.3.2 모듈러 지수 연산

모듈러 지수 연산은 Binary Exponentiaition 알고리즘을 이용하여 구현하였다. 이 알고리즘은 B.C 200년경에 개발된 가장 오래 전부터 알려진 멱승 연산 알고리즘이며, repeated square and multiply이라고도 하는데 이름에서 알 수 있듯이 제곱과 곱셈을 반복하여 멱승을 수행한다[8].

알고리즘 구성
INPUT : g는 $g \in G$ 값 과 $e \ge 1$인 정수 e OUTPUT : g^e 1. A ← 1, S ← g 2. while $e \ne 0$ do 2.1 if e가 홀수이면 A ← A×S 2.2 e ← ⌊ e/2 ⌋ 2.3 if $e \ne 0$이면 S ← S×S 3. A를 반환

e의 이진 표현이 t+1 길이를 갖는다고 하고, wt(e)는 이진 표현에서의 1의 개수를 의미한다고 하자. 알고리즘은 t번의 제곱과 wt(e)-1 곱셈을 수행한다. 만약 e가 $0 \le e < |G| = n$(|G|는 G의 절대값)의 범위에서 임의로 선택되었다면, 대략 ⌊log n⌋번의 제곱과 1/2(⌊log n⌋+1)번의 곱셈만을

수행할 수 있다. 만약 제곱이 곱셈과 계산량이 거의 같다면, 기대되는 연산은 대략 2/3 ⌊log n⌋ 번의 곱셈뿐이다[7].

3.3.3 승산 역원 연산

모듈러 n에 대한 곱셈의 역원(a^{-1} mod n)인 승산 역원은 Extended-Euclid Algorithm을 이용하여 구현하였다.

알고리즘 구성
INPUT : 두 정수 a, b $0 < b \leq a$ OUTPUT : d = gcd(a, b) 1. 만약 b = 0 이면, 1.1 $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$ 1.2 d, x, y를 반환 2. $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, $y_2 \leftarrow 0$, $y_1 \leftarrow 1$ 3. While b >0 3.1 $q \leftarrow a = b$, $r \leftarrow a - q \times b$, $x \leftarrow x_2 - q \times x_1$, $y \leftarrow y_2 - q \times y_1$ 3.2 $a \leftarrow b$, $b \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$, $y_2 \leftarrow y_1$, $y_1 \leftarrow y$ 4. $d \leftarrow a$, $x \leftarrow x_2$, $y \leftarrow y_2$ 5. d, x, y를 반환

모듈러 n에 대한 곱셈의 역원은 $ax \equiv 1 \pmod{n}$를 만족하는 x이다. 만약 x가 존재한다면, 그 역원은 유일하다. 이때 a의 역원이 존재한다고 하거나 유일하다고 한다. a의 역원은 a^{-1}로 표현한다. 만약 d > 1이면, a^{-1} mod n이 존재하지 않는다. 그러나 a mod n의 곱셈에 대한 역원은 x가 된다.

4. 클래스의 구현

본 논문의 구현 환경은 <표 2>와 같다. Java Card Development Kit은 본래 Solaris와 Windows NT 4.0 환경에서 동작하지만 windows2000이 NT 기술을 사용하기 때문에 아래와 같은 환경에서 사용하였다.

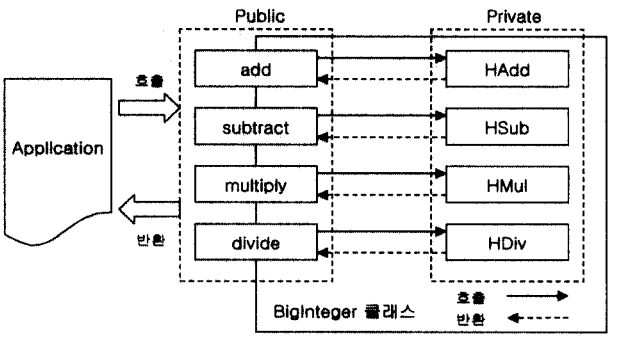
〈표 2〉 구현 환경

운영체제	Windows 2000 Professional	
하드웨어	CPU	Pentium III 800 MHz
	RAM	256 MB
개발도구	JDK 1.3	
	Java Card 2.1.2 Development Kit	

구현한 BigInteger 클래스의 구조는 호환성을 위해서 JDK1.3의 BigInteger 클래스와 가능한 동일하게 하였다. 구현의 목적이 구현된 클래스를 기반으로 하여 RSA, DSA, KCDSA와 같은 공개키 암호 알고리즘의 구현이기 때문에 암호 알고리즘의 구현에 필요하지 않는 메소드들과 Java Card에서 지원되지 않는 자료형을 사용하는 메소드들은 구현에서 제외하였다.

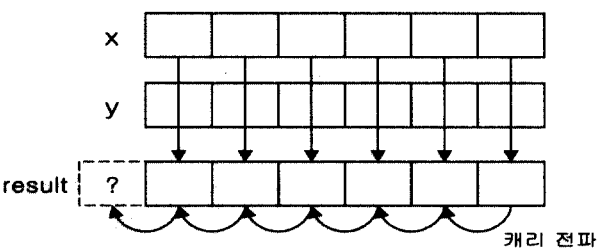
4.1 산술 연산

구현한 산술 연산은 외부에서 호출되는 부분과 실제 연산이 수행되는 부분을 분리하여 구현하였다. 그 이유는 하드웨어적인 구현이 추가되었을 때 실제 연산의 일부만을 변경함으로써 클래스의 성능을 크게 향상시킬 수 있을 뿐만 아니라 코드 상의 변경도 적어지기 때문이다. 카드에 내장된 마이크로프로세서의 성능이 향상되고는 있지만 여전히 16비트 프로세서를 사용하는 것이 많기 때문에 바이트 단위로 연산을 수행하였다. 산술연산의 호출 구조는 아래 (그림 5)과 같다.



(그림 5) 산술 메소드 호출 구조

구현되는 덧셈, 뺄셈, 곱셈은 다음의 (그림 6)에서 처럼 바이트 단위로 처리된다. 하위 바이트에서부터 시작하여 바이트 배열의 끝까지 반복해서 계산하며, 이때 캐리 혹은 빌림이 발생하면 그것을 전파하면서 연산을 계속한다.



(그림 6) 기본 연산 구조도

4.1.1 덧셈

덧셈은 두개의 바이트 배열을 한 바이트 단위로 계산하였다. 두 개의 바이트 배열을 서로 더한 후에 그 결과를 바이트 배열의 형태로 반환된다.

```
private static byte[] HAdd(byte[] x, byte[] y){
  // x_i , y_i는 각 배열의 길이, r : 결과 배열, sum : 임시 합.
  while (y_i > 0){
    // 바이트 단위로 덧셈 수행
    sum = (x[--x_i] & 0xff) +
        (y[--yIndex] & 0xff) + (sum >>> 8) ;
    r[x_i] = (byte)(sum & 0x00ff) ;
  }
```

```
  boolean c = (((sum >>> 8)& 0x00ff) != 0) ;    // 캐리 검사
  // 캐리가 발생하고 x가 남았으면 캐리를 반영하고 x를 결과에
     누적
  while(x_i > 0 && c)
      c = ((r[--x_i] = (byte)(((x[x_i] & 0xff) + 1) & 0x00ff)) == 0) ;
  // 캐리가 발생하지 않고 x가 남았으면 x를 결과에 누적
  while (x_i >0)
      r[--x_i] = (byte)(x[x_i] & 0x00ff) ;
  // 마지막에 캐리가발생하면 자리수를 한자리 늘려서 결과 반영
  return r ;
}
```

4.1.2 뺄셈

두 바이트 배열을 한 바이트씩 short형 변수에 저장한 후에 이 short형 변수를 뺄셈을 하고 그 결과를 다시 바이트 배열에 저장하는 방법을 사용하였다. 두 개의 바이트 배열을 서로 뺄셈을 수행한 후에 그 결과를 바이트 배열의 형태로 반환한다.

```
private static byte[] HSub(byte[] x, byte[] y){
  // x > y 인 수, diff : 뺀 값, b : 빌림, x_i, y_i : 배열의 길이
  // 두 수의 공통 부분에 빼기를 수행
  while(y_i > 0){
    diff = (x[--x_i]&0xff) - (y[--y_i] & 0xff) + (diff >> 8) ;
    result[x_i] = (byte)(diff & 0x00ff) ;
  }
  // 뺄셈의 결과 빌림이 발생했다면, 다음 단계에 반영시킴.
  boolean b = (diff >> 8 != 0) ;
  while(x_i > 0 && b )
    b = ((result[--x_i] = (byte)(((x[x_i] & 0xff) -1) & 0x00ff))
        == -1) ;
  // 연산이 끝난후에 남은 x값의 처리
  return result ;
}
```

4.1.3 곱셈

두 바이트 배열을 한 바이트씩 short 형 변수에 저장 후에 계산을 하고 그 결과를 다시 바이트 배열에 저장한다. 두 개의 바이트 배열을 서로 곱한 후에 그 결과를 바이트 배열의 형태로 반환한다.

```
private byte[] HMul(byte[] x, byte[] y){
  // c : 캐리 , p : 곱셈 결과, x_i, y_j : 배열의 길이, result : 결과
  // 덧셈과 유사하게 연산 진행
  for(short j = y_i, k = (short)(y_i+x_i+1)  ; j >= 0 ; j--, k--){
    int p = (y[j] & 0xff) * (x[xstart] & 0xff) + c ;
    result[k] = (byte)(p & 0x00ff) ;
    c = p >>> 8 ;
  }
  result[x_i] = (byte)(c & 0x00ff) ;      // 캐리 반영
  // 남아 있는 값의 연산
  for(short i = (short)(x_i-1) ; i >= 0  ; i--){
    carry = 0 ;
    for( short j = y_i, k = y_i+1+i  ; j >= 0 ; j--, k--){
      int p = (y[j] & 0xff) * (x[i] & 0xff) + (result[k] & 0xff) + c ;
      result[k] = (byte)(p & 0x00ff) ;
      c = p >>> 8 ;
    }
    result[i]=(byte)(c & 0x00ff) ;
  }
  return result ;
}
```

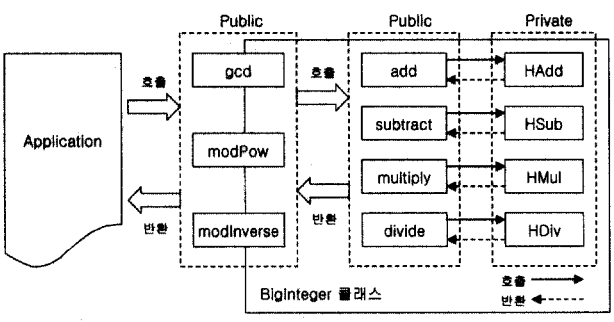
4.1.4 나눗셈

두 수의 나눗셈은 다른 산술 연산과는 다르게 쉬프트와 뺄셈을 이용하여 수행한다. 두 바이트 배열을 x, y를 이미 정의한 연산들을 이용하여 나눗셈을 수행한 후에 그 결과를 바이트 배열의 형태로 반환한다.

```
private BigInteger HDiv(BigInteger rem, BigInteger m){
  // 자리수가 맞을 때까지 쉬프트를 수행
  while(rem.compareTo(m) > 0){
    m = m.shiftLeft(1) ;
    s = s.shiftLeft(1) ;
  }
  // 몫이 0보다 작을 때까지 쉬프트를 수행
  while(rem.compareTo(new BigInt(1, y)) >= 0){
    while (rem.compareTo(m) < 0){
      m = m.shiftRight(1) ;
      s = s.shiftRight(1) ;
    }
    rem = rem.subtract(m) ;       // 몫을 감소 시킴
    result = result.add(s) ;      // 나머지를 증가시킴
  }
  return result ;
}
```

4.2 응용 연산

공개키 암호 알고리즘의 구현에 필요한 모듈러 연산, 지수연산, 최대공약수 계산 및 소수 판정과 생성은 계산된 결과 또한 자바 카드에서는 표현할 수 있는 최대 자료형인 정수의 범위를 훨씬 벗어나게 된다. 따라서 응용 연산을 구현은 앞에서 구현한 산술 연산을 이용하여 구현하였다. 일부 응용 연산의 호출 구조가 다음 (그림 7)에 있다.



(그림 7) 응용 연산 흐름도

4.2.1 최대공약수

최대공약수를 구하는 연산은 Euclid 알고리즘을 이용하여 다음과 같이 구현하였다.

아래 메소드는 BigInteger형 val1과 val2의 최대공약수를 구해서 result에 반환한다.

```
public BigInteger gcd(BigInteger val) {
  BigInteger ZERO = new BigInteger (new byte[0], 0) ;
  // 어느 한쪽이 0이면 다른 쪽을 반환
  if (val.signum == 0) return this.abs() ;
  else if (this.signum == 0) return val.abs() ;
  else{
    // g ← val, x ← this, y ← g
    // 유클리드 알고리즘 구현
    while(x.compareTo(ZERO) == 1){
        g = x ;
        x = y.remainder(x) ;
        y = g ;
    }
  return g ;
}
```

4.2.2 모듈러 지수 연산

모듈러 지수 연산은 Binary Exponent 알고리즘을 이용하여 다음과 같이 구현하였다. 이 메소드는 result = v^e mod m을 계산하여 결과를 반환한다.

```
public BigInteger modPow(BigInteger e, BigInteger m) {
  // ZERO ← 0 , ONE ← 1
  // 만약 e가 -1 이면 res ← -1
  //          1 이면 res ← 1
  BigInteger Pow2 = this ;
  while(e.compareTo(ZERO) != 0){
    // A <- A * S ;
    if (e.testBit(0))
        res = Pow2.multiply(res).mod(m) ;
    e = e.shiftRight(1) ;
    // S <- S * S ;
    if (e.compareTo(ZERO) != 0)
        Pow2 = Pow2.multiply(Pow2).mod(m) ;
  }
  return res ;
}
```

4.2.3 승산 역원 연산

승산역원 연산은 Extended Euclidean Algorithm 알고리즘을 이용하여 다음과 같이 구현하였다. 이 메소드는 result = v^{-1} mod m을 계산하여 결과를 반환한다.

```
public BigInteger modInverse(BigInteger m)
      throws ArithmeticException {
  // ZERO← 0 , ONE ← 1, j ← 1 , I ← 0, b ← m.
  // x ← 0, y ← 0, c ←this
  while(c.compareTo(ZERO) != 0){
    x = b.divide(c) ;
    y = b.subtract(x.multiply(c)) ;
    b = c ; c = y ; y = j ;
    j = x.multiply(j) ;
    j = i.subtract(j) ;
    i = y ;
  }
  if( i.signum() < 0)
    i = i.add(m) ;
  return i ;
}
```

4.3 실행 결과

본 논문에서 구현한 클래스는 PC환경에서 개발되었다. 그렇기 때문에 일단 JDK1.3을 이용하여 클래스를 컴파일하고, Converter를 이용하여 CAP파일을 생성한다. 그 후 Scriptgen이라는 명령어를 이용하여 CAP파일을 APDU명령어로 구성된 SCR파일로 변환한다. 그 후 JCWDE라는 명령어를 이용하여 JCVM과 작성한 클래스를 메모리 상에 로드하고, SCR파일을 호출하는 별도의 SCR파일을 작성하여 동작을 검증하였다.

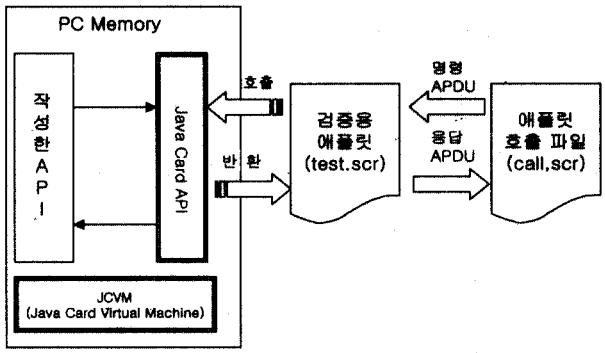
자바 카드 애플릿의 개발과정을 단계별로 보면 다음과 같다.

1 단계 : 소스 파일을 작성한다(test.java).
2 단계 : 소스 파일을 컴파일 한다(test.class).
3 단계 : 컴파일한 프로그램이 카드 명세에 맞는지 검증하기 위해서 Converter라는 명령어를 이용하여 검증한다. 이때 올바르게 검증되면 JCA, EXP와 CAP 파일 등이 생성된다(test.jca, test.exp, test.cap).

다음 단계는 실제 카드 상에 설치하는 단계와 PC 또는 workstation상에서 시뮬레이션 하는 단계로 나누어진다. 본 논문에서는 PC 상의 시뮬레이션에 대해서만 설명하겠다.

4 단계 : 생성된 CAP 파일을 scriptgen이라는 명령어를 이용하여 APDU 명령어로 변환한다(test.scr).
5 단계 : 앞 단계에서 생성된 SCR 파일을 호출하는 별도의 SCR파일을 작성한다. 본래 호스트 응용 프로그램과 상호 대화하면서 동작을 해야 하지만 시뮬레이션 환경 상에서는 불가능하므로 호스트의 요청을 별도의 SCR 파일로 작성한다(call.scr).
6 단계 : 가상의 JCVM을 PC의 메모리 상에 띄워 놓고 이전 단계에 작성한 SCR(call.scr)을 이용하여 서로 통신을 시도한다. 이 단계는 모두 APDU 명령어를 이용하여 통신이 이루어진다.

테스트 과정을 그림으로 표현하면 다음과 같다.



(그림 8) 검증 애플릿 실행 구조

다음은 구현한 클래스를 테스트하기 위해서 작성한 자바 카드 애플릿의 내용이다. 구현된 클래스의 연산 중에서 간단히 덧셈, 모듈러 지수승 연산과 멱승 연산만을 테스트하기로 한다.

4.3.1 초기화

실제 연산을 수행하기 전에 각 변수에 값을 할당하는 부분이 필요하다. APDU 명령어로 전달된 바이트 값을 BigInteger형으로 변환 후에 내부 변수에 저장한다.

```
private void init1(APDU apdu){
  byte[] buffer = apdu.getBuffer() ;
                    // 명령 APDU를 버퍼에 저장
  mesLeng = buffer[ISO7816.OFFSET_LC] ;
                    // 명령 APDU에서 데이터의 길이 구하기
  byte[] orMessage = new byte[mesLeng] ;
                    // 명령 APDU에서 데이터를 저장할 배열
  // 명령 APDU에서 배열로 값을 복사
  arrayCopy(buffer, ISO7816.OFFSET_CDATA,
  orMessage, 0, mesLeng) ;
  val1 = new BigInt(1, orMessage) ;
                    // 바이트 배열을 BigInteger형으로 변환
}
```

4.3.2 덧셈 연산 호출 부분

다음의 코드는 초기화과정으로 생성된 두 BigInteger형 변수를 서로 더한 후 val3이라는 변수에 저장하고, 그 결과를 바이트 배열로 변환한 후에 ADPU명령어를 통해서 결과를 전송한다.

```
private void cal_add(APDU apdu){
  val3 = val1.add(val2) ;
                    // 미리 저장된 값들을 서로 더함
  mdMessage = val3.toByteArray() ;
                    // 계산 결과를 바이트 배열로 변환
  short len = (short)mdMessage.length ;
                    // 전송할 데이터의 길이를 설정
  apdu.setOutgoingLength((byte)len) ;

  for (byte i = 0 ; i < len ; i++)
    buffer[i] = mdMessage[i] ;
                    // 전송할 데이터를 버퍼에 복사한다.
  apdu.sendBytes((short)0, (short)len) ;
                    // 버퍼의 값을 전송한다.
}
```

4.3.3 모듈러 지수승 연산 호출 부분

다음의 코드는 a^e mod m을 계산하는 모듈러 지수승 연산을 호출하는 부분이다. 사용되는 변수는 모두 3개이며 결과를 다시 val3에 저장하고, 이를 바이트 배열로 변환한 후에 APDU 명령어를 통해서 전송한다.

```
private void cal_modPow(APDU apdu){
  val3 = val1.modPow(val2,val3) ;
                    // val3 = val1^val2 mod val3을 계산
  mdMessage = val3.toByteArray() ;
                    // 계산 결과를 바이트 배열로 변환
  short len = (short)mdMessage.length ;
                    // 전송할 데이터의 길이 설정
  apdu.setOutgoingLength((byte)len) ;
  for (byte i = 0 ; i < len ; i++)
    buffer[i] = mdMessage[i] ;
                    // 전송할 데이터를 버퍼에 복사한다.
  apdu.sendBytes((short)0, (short)len) ;
                    // 버퍼의 값을 전송한다.
}
```

4.3.4 멱승 연산 호출 부분

다음의 코드는 a^e를 계산하는 지수 연산을 호출하는 부분이다. 초기화과정에서 생성된 두 변수를 이용하며, 결과를 val3에 저장하고 이를 바이트 배열로 변환한 후에 APDU 명령어르 통해서 전송한다.

```
private void cal_pow(APDU apdu){
  val3 = val1.pow(val2.intValue()) ;
                    // val3 = val1^val2를 계산
  mdMessage = val3.toByteArray() ;
                    // 계산 결과를 바이트 배열로 변환
  short len = (short)mdMessage.length ;
                    // 전송할 데이터의 길이를 설정
  apdu.setOutgoingLength((byte)len) ;
                    // 전송할 데이터를 버퍼에 복사한다.
  for (byte i = 0 ; i < len ; i++)
    buffer[i] = mdMessage[i] ;
                    // 버퍼의 값을 전송한다.
  apdu.sendBytes((short)0, (short)len) ;
}
```

4.3.5 애플릿 호출 파일

다음은 검증용 애플릿과 통신을 위한 APDU 명령어로 구성된 파일이다. 이 파일은 카드의 호스트 프로그램에서 요청해야 하는 것이지만, 시뮬레이션 환경에서는 정해진 명령어를 배치 파일의 형태로 구성하고, 애플릿과 통신하

도록 하였다. 실제 호출부분이 아닌 부분은 간략히 표시하겠다.

```
powerup ;
// installer applet을 선택한다.
// Test Applet을 생성한다.
0x80 0xB8 0x00 0x00 0x11 0x0a 0xa0 0x0 0x0 0x0 0x62 0x3
0x1 0xc 0x1 0x1 0x05 0x01 0x02 0x03 0x04 0x05 0x7F ;
// Test Applet을 선택한다.(Select)
0x00 0xa4 0x04 0x00 0x0a 0xa0 0x0 0x0 0x0 0x62 0x3 0x1
0xc 0x1 0x1 0x7f ;
// val1 변수를 초기화 한다. (init1)
0xC0 0x10 0x00 0x00 0x02 0x03 0x38 0x02 ;
// val2 변수를 초기화 한다. (int2)
0xC0 0x11 0x00 0x00 0x01 0x04 0x02 ;
// 덧셈을 호출한다. (add)
0xC0 0x21 0x00 0x00 0x00 0x7f ;
// 모듈러 지수승 연산을 호출한다. (modPow)
0xC0 0x22 0x00 0x00 0x00 0x7f ;
// 지수 연산을 호출한다. (pow)
0xC0 0x23 0x00 0x00 0x00 0x7f ;
//스크립트 종료
powerdown ;
```

위의 스크립트 중에서 밑줄 그은 부분을 살펴보면 0XC0는 테스트용 애플릿을 식별하는 식별자이다. 다음의 0X10은 명령어 코드, 즉 메소드를 의미하며 다음의 2바이트 0X00 0X00은 명령어코드의 파라메터이다. 다음의 0X02는 데이터의 길이를 의미하며, 0X03 0X38은 데이터를 의미하고, 0X02는 이 명령어의 결과로 기대되는 응답 코드의 길이이다.

4.3.6 실행 결과

앞에서 작성한 검증용 애플릿과 통신을 위한 APDU 명령어 파일을 이용하여 실행하여 얻은 결과가 다음에 있다.

```
. . . . . ( 생략 ) . . . . .
CLA : c0, INS : 10, P1 : 00, P2 : 00, Lc : 02, 03, 38, Le : 00,
      SW1 : 90, SW2 : 00
CLA : c0, INS : 11, P1 : 00, P2 : 00, Lc : 01, 04, Le : 00, SW1 :
      90, SW2 : 00
CLA : c0, INS : 21, P1 : 00, P2 : 00, Lc : 00, Le : 02, 03, 3c,
      SW1 : 90, SW2 : 00
CLA : c0, INS : 22, P1 : 00, P2 : 00, Lc : 00, Le : 02, 01, 00,
      SW1 : 90, SW2 : 00
CLA : c0, INS : 23, P1 : 00, P2 : 00, Lc : 00, Le : 05, 6b, 56, 3e,
      10, 00, SW1 : 90, SW2 : 00
```

위의 결과에서 CLA, INS, P1, P2등은 이전에 (그림 2)와 (그림 3)에서 설명한 것이다. 밑줄 친 부분이 각 명령어에서 응답으로 넘어온 값이다. 초기 입력이 16진수 338과 4일 때, 덧셈, 모듈러 지수 연산, 멱승 연산의 결과가 출력된 것을 볼 수 있다. 이때 첫 바이트는 결과 값의 길이를 의미한다.

5. 결 론

자바 카드의 안전성과 이식성은 e-비즈니스에서 신뢰성, 편리성과 안전성을 보장하는 효율적인 수단을 제공한다. 그러나 자바 카드는 제한된 자원 상에서 동작을 한다. 즉, 이러한 자바 카드에서 응용프로그램의 설계 및 구현에서 가장 많은 영향을 미치는 요인은 자원의 제한성이다. 자바 카드 API는 자바 API의 일부만을 정의해 놓았다. 그렇기 때문에 이 API는 공개키 암호 알고리즘 구현에 필요한 모듈러 연산, 최대 공약수 계산, 소수의 판정과 생성과 같은 일부 연산은 지원하지 않는다.

본 논문에서는 공개키 암호 알고리즘 구현에 필요한 BigInteger 클래스를 자바 카드 API에서 지원하지 않기 때문에, 자바 카드 환경에서 동작 가능한 BigInteger 클래스를 구현하였다. 이 클래스는 한정된 자료형만을 지원하는 자바 카드 상에서 공개키 암호 시스템을 구현하기 위해서는 반드시 필요한 클래스이다. 만약 이 클래스의 산술 연산 부분이 하드웨어로 지원된다면, 클래스의 성능은 더욱 향상될 것이다.

32비트 프로세서의 처리에 적합하게 변경을 하는 것을 계속 연구해야 하며, 이를 이용한 타원곡선 암호 알고리즘과 같은 공개키 암호 알고리즘의 구현은 향후 연구 과제로 남긴다.

참 고 문 헌

[1] 문상재 외, "차세대 IC 카드를 사용한 정보보호 신기술 시스템 개발", 정보통신부, pp.17, 1997.

[2] 김연선, 이창욱, "자바카드 애플릿 설계 및 검증에 관한 연구", 한국통신정보보호학회 종합학술발표회논문집, Vol. 10, No.1, p.805, 2000.

[3] Chen, Zhiqun, "Java Card Technology for Smart Cards," ADDISON-WESLEY Company, pp.42-72, 2000.

[4] http://java.sun.com/products/javacard/datasheet.html

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

[6] http://java.sun.com/products/jdk/1.3/docs/api/java.math.BigInteger.html.

[7] A. Menezes, P. van Oorschot, S. Vanstone, "Handbook of Applied Cryptography," CRC Press, pp.66-614 1996.

[8] 황효선, 임채훈, 이필중, "멱승 알고리즘의 구현과 분석", 한국통신 정보보호학회 학회지, Vol.5, No.1, p.5, 1995.

김 성 준

e-mail : sjkim@netwk.hannam.ac.kr
2000년 한남대학교 컴퓨터공학과 (공학사)
2002년 한남대학교 대학원 컴퓨터공학과 (공학석사)
2002년~현재 한남대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 컴퓨터네트워크, 정보통신 정보보호

이 희 규

e-mail : june@netwk.hannam.ac.kr
1998년 우송대학교 컴퓨터과학과(공학사)
2000년 한남대학교 대학원 컴퓨터공학과 (공학석사)
2000년~현재 한남대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 컴퓨터네트워크, 정보통신 정보보호

조 한 진

e-mail : hjcho@netwk.hannam.ac.kr
1997년 한남대학교 컴퓨터공학과 (공학사)
1999년 한남대학교 대학원 컴퓨터공학과 (공학석사)
2002년 한남대학교 대학원 컴퓨터공학과 (박사수료)
2002년~현재 극동대학교 정보통신학부 컴퓨터전공 전임강사
관심분야 : 컴퓨터네트워크, 정보통신 정보보호

이 재 광

e-mail : jklee@netwk.hannam.ac.kr
1984년 광운대학교 전자계산학과(이학사)
1986년 광운대학교 대학원 전자계산학과 (이학석사)
1993년 광운대학교 대학원 전자계산학과 (이학박사)
1986년~1993년 군산전문대학 전자계산학과 부교수
1997년~1998년 University of Alabama 객원교수
1993년~현재 한남대학교 컴퓨터공학과 부교수
관심분야 : 컴퓨터 네트워크, 정보통신 정보보호