자바 카드 기반 BigIntger 클래스 구현

이원구, 이재광
한남대학교 컴퓨터공학과
wglee@netwk.hannam.ac.kr

Implementation of BigInteger Class based on Java Card

Won-Goo Lee, Jae-Kwang Lee
Dept. of Computer Engineering, Hannam University

요 약

자바카드 API는 스마트 카드와 같은 작은 메모리를 가진 임베디드 장치에서 실행환경을 최적화하기 위해 구성되었다. 자바카드 API의 목적은 한정된 메모리를 가진 스마트카드 기반의 프로그램을 개발할 때 많은 이점을 제공한다. 그러나 공개키 암호 알고리즘 구현에 필요한 연산들인 모듈러 연산, 최대공약수 계산, 그리고 소수 판정과 생성 등의 연산을 지원하지 않는다. 본 논문에서는 이러한 기능을 제공하는 자바카드 기반 BigInteger 클래스의 설계 및 구현에 목적을 둔다.

I. 서론

실용적이고 효과적인 정보보호 서비스를 제공하기 위해서 IC 카드의 사용이 급증하고 있으며 이와 관련한 기술 개발이 활발히 이루어지고 있다[2]. 개인용 컴퓨터나 무역망, 금융망, 행정망 및 의료망 등에서 정보보호에 IC 카드를 사용하는 기술이 이미 일부 국가에서는 실용화 단계에 있으나 국내에서는 미진한 상태이다. 현재 주로 실용화되고 있는 IC 카드 기술은 스마트 카드 관련 기술과 자바 카드 관련 기술이 주류를 이루고 있다.

이러한 기술 중에 자바 카드 기술은 어떠한 운영체제상에서도 응용프로그램을 작동시켜주는 개방형 플랫폼인 자바언어로 구현됐다는 점에서 주목을 받는다. 따라서 자바 카드는 그 간 특정 플랫폼 의존적인 시스템을 개발해야 만 했던 문제를 해결해준다.

자바 카드 API에는 자바 API의 일부분만을 정의해 놓았다. 때문에 공개키 암호 알고리즘의 구현에 필요한 BigIntger 클래스를 지원하지 않는다. 본 논문에서는 자바 카드에서 동작할 수 있는 BigInteger 클래스를 구현한다. 2장에서는 자바카드와 구현에 적용한 알고리즘을 살펴보고, 3장에서는 구현한 클래스의 내용중 응용연산을 설명하였다. 4장에서는 이를 기반으로 BigInteger 클래스를 구현하고, 5장에서 결론을 맺는다.

II. 연구 배경

1. 자바 카드(Java Card)

자바 카드란 COS(Card Operating System)위에 JCVM(Java Card Virtual Macine)이 랩핑(Wrapping)되어 있는 구조의 스마트 카드를 말한다[1]. 자바 카드 API는 자바 카드 상에서 Java를 이용한 소프트웨어 개발에 필요한 API들을 정의한 것이다.

이것은 스마트 카드의 보안성을 연구하던 Schlumberger사의 연구팀에 의해 1996년에 소개되었다. 이 후 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. 그 후 계속 발전하여 현재 2.1.1버전까지 개발되어 있는 상태이다[6].

자바 카드 API는 전자상거래, 네트워크 접근, 인증을 위한 차세대 네트워크 기술을 제시하였다. Bull, Gemplus, Schulmberger 등 전 세계 스마트 카드 제조 회사의 90%이상이 자바 카드의 개발을 위해 라이센스를 이미 받은 상태이다[8].

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

EMV(Europay/ MasterCard/Visa)와 서로 호환된다. 아래 표 1은 Java와 자바 카드의 자료형을 비교하였다.

표 1 : 자바 카드 API에서 지원되는 자료형

자료형	자바 API	자바카드 API
BigInteger	지원	지원 안함
boolean	지원	지원
byte	지원	지원
char	지원	지원 안함
double	지원	지원 안함
float	지원	지원 안함
int	지원	지원
long	지원	지원 안함
short	지원	지원
string	지원	지원 안함

2. 연산 알고리즘

본 논문에서 구현한 BigIntger 클래스에서 사용한 대표적인 알고리즘은 다음과 같다.

1) Binary Exponentiaion 알고리즘

가장 오래 전부터 알려진 멱승 연산 알고리즘은 B.C 200년경에 개발된 binary method이다. 이 알고리즘은 repeated square and multiply이라고도 하는데 이름에서 알 수 있듯이 제곱과 곱셈을 반복하여 멱승을 수행한다[3].

알고리즘 구성

INPUT : g는 $g \in G$ 값 과 $e \geq 1$인 정수 e

OUTPUT : g^e

1. $A \leftarrow 1$, $S \leftarrow g$
2. while $e \neq 0$ do
   2.1 if e가 홀수이면 $A \leftarrow A \cdot S$
   2.2 $e \leftarrow \lfloor e/2 \rfloor$
   2.3 if $e \neq 0$ 이면 $S \leftarrow S \cdot S$
3. A를 반환

e의 이진 표현이 t+1 길이를 갖는다고 하고, wt(e)는 이진표현에서의 1의 개수를 의미한다고 하자. 알고리즘은 t번의 제곱과 wt(e)-1 곱셈을 수행한다. 만약 e가 $0 \leq e < |G|$ = n의 범위에서 임의로 선택되었다면, 대략 $\lfloor \lg n \rfloor$번의 제곱과 $1/2(\lfloor \lg n \rfloor + 1)$번의 곱셈만을 수행할 수 있다. 만약 제곱이 곱셈과 계산량이 거의 같다면, 기대되는 연산은 대략 $2/3 \lfloor \lg n \rfloor$번의 곱셈뿐이다[5].

2) Euclidean 알고리즘

Euclidean 알고리즘은 인수분해를 요구하지 않으면서 두 정수의 최대공약수를 계산하는데 효과적인 알고리즘이다[4].

알고리즘 구성

INPUT : $a > b$인 두 양의 정수

OUTPUT : a 와 b의 최대 공약수

1. while $b \neq 0$ do
   1.1 $r \leftarrow a \bmod b$, $a \leftarrow b$, $b \leftarrow r$
2. a를 반환

만약 a=p1e1p2e2…pkek이고 b=p1f1p2f2…pkfk이고, ei ≥0와 fi≥이면, 두 정수 a 와 b의 최대공약수 gcd는 p1min(e1,f1)p2min(e2,f2)…pkmin(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[4].

3) Extended Euclidean 알고리즘

Extended Euclidean 알고리즘은 승산역원을 구하는데 유용하게 사용되는 알고리즘이다.

알고리즘 구성

INPUT : $a \geq b$인 두 양의 정수

OUTPUT : $\{ (x, y) \mid gcd(a,b) = ax+by \}$

1. $x = v = 1$, $y = u = 0$
2. $\{ (q, r) \mid a = qb + r \} \ (0 \leq r < b)$
   2.1 $a \leftarrow b$, $b \leftarrow r$
   2.2 $x \leftarrow u$, $y \leftarrow v$
   2.3 $u \leftarrow x-qu$, $v \leftarrow y-qv$
3. $b = 0$이 될 때까지 Step 2 반복
4. x와 y를 반환

이 알고리즘은 승산역원을 구하기 위해서 2번의 뺄셈과 2번의 곱셈만을 이용하여 결과를 얻게 되는 알고리즘이다.

III. BigInteger의 내용 및 구성

JDK 1.1은 임의로 결정되어질 수 있는 정수를 표현하기 위해서 java.math.BigInteger라는 클래스를 도입하였다[7]. 이 클래스에서 모든 연산들은 2의 보수의 형태로 처리된다. 또한 기본 연산들 이외에 추가적으로 모듈러 연산, 최대공약수 계산, 소수판정 및 생성과 비트 연산 등을 제공한다[9].

BigIntger 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 앞의 표 1에서 살펴 본바와 같이 BigIntger 클래스를 지원하지 않기 때문에 암호알고리즘을 구현하기 위해서는 별도의 라이브러리는 사용해야만 한다.

본 논문에서는 JDK1.1.8의 BigInteger 클래스를 기반으로 하여 자바 카드에서 동작하는 클래스를 구현하였다. 구현한 클래스의 메소드들은 호환성을 위해 JDK1.1.8에서 제공되는 클래스와 동일한 이름을 사용하였으며 암호알고리즘의 구현에 불필요한 메소드들과 자바 카드에서 지원되지 자료형에 관련된 연산들은 제외하였다. 구현된 메소드들 중에서 산술연산과 응용연산만을 살펴본다.

1. 산술연산

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

실제 값	1000
저장 값	03 E8

즉, 부호를 지시하는 변수는 양수를 의미하는 1의 값을 가지게 된다.

1) 덧셈

알고리즘 구성

INPUT : $a > b$인 두 수
OUTPUT : a 와 b의 합
1. $i \leftarrow a$의 하위 바이트, $j \leftarrow b$의 하위 바이트
2. while b의 첨자 > 0 do
   2.1 *sum* ← *I* + *j* + *(sum >>>8)*
   2.2 *result* ← *(sum & 0xff)*
3. *result*를 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.

2) 뺄셈

알고리즘 구성

INPUT : $a > b$인 두 수
OUTPUT : a 와 b의 차
1. $i \leftarrow a$의 하위 바이트, $j \leftarrow b$의 하위 바이트
2. while b의 첨자 > 0 do
   2.1 *diff*←*i* - *j*+ *(diff >>8)*
   2.2 *result* ← *(diff & 0xff)*
3. *result*를 반환

두 수의 뺄셈에서 각 수는 바이트 배열의 형태로 저장된다. 이때 두 개의 바이트 배열에서 하위 한 바이트를 short형 변수에 각각 할당한 후에 short형을 뺄셈을 취한다. 연산 결과를 오른쪽으로 8비트 쉬프트한 값이 0이 아니면, 빌림이 발생한 것이다. 그러면 그 자리올림도 뺄셈에 반영시킨다.

3) 곱셈

알고리즘 구성

INPUT : $a > b$인 두 수
OUTPUT : a 와 b의 곱
1. $i \leftarrow a$의 하위 바이트, $j \leftarrow b$의 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. *result*를 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.

4) 나눗셈

알고리즘 구성

INPUT : $a > b$인 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*
   3.3 *retval* ← *retval* + *s*
4. *retval* 반환

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

2. 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 범위를 훨씬 벗어나게 된다. 따라서 응용연산을 구현할 때 미리 구현한 BigInteger 클래스의 사칙연산을 이용하여 구현하였다.

Binary Exponentiaion 알고리즘은 지수 연산에 사용하였으며 Eculid 알고리즘은 최대 공약수를 구하는 부분에, 승산역원을 계산하는 부분에서는 Extended Euclidean 알고리즘을 적용하였다.

IV. BigIntger의 구현

본 논문의 구현 환경은 표 2와 같다.

표 2 : 구현 환경

운영체제	Windows 2000 Professional
하드웨어	Pentium III 800 MHz
개발도구	JDK 1.1.8
	Java Card 2.1 Development Kit

1. 구현

본 논문에서 구현한 BigInteger 클래스의 메소드들은 JDK1.1.8의 BigInteger 클래스를 참조해서 구현하였다. 또한 호환성을 위해 메소드들의 이름과 인자형을 동일하게 하였다.

1) 최대공약수

최대공약수를 구하는 연산은 Euclid 알고리즘을 이용하여 다음과 같이 구현하였다.

아래 메소드는 retsult = val1.gcd(val2)와 같이 선언되면 val1과 val2의 최대공약수를 구해서 result에 반환한다.

```
public BigInt gcd(BigInt val) {
  BigInt ZERO = new BigInt(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;
  }
}
```

2) 모듈러 연산

이 절에서 구현한 모듈러 연산은 Binary Exponent 알고리즘을 적용하였으며 result = v.modPow(e,m)과 같이 선언하면 ve mod m을 계산하여 result에 반환한다.

```
public BigInt modPow(BigInt e, BigInt m) {
   // ZERO← 0  , ONE ← 1
   // 지수가 0 이면 1을 반환
   // 만약 e 가 -1 이면 res ← -1
   //            1 이면 res ←  1
   BigInt 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;
}
```

3) 승산역원 연산

승산역원 연산은 Extended Euclidean Algorithm 알고리즘을 적용하였으며 result = v.modInverse(m)과 같이 선언하면 v-1 mod m을 계산하여 result에 반환한다.

```
public BigInt modInverse(BigInt 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;
}
```

2. 성능 평가

본 논문에서 구현한 클래스를 시험하기 위해서 JDK1.1.8을 이용하여 클래스를 호출하여 연산을 수행해 보았다. 또한 연산의 결과를 JDK1.1.8에서 제공하는 BigInteger 클래스를 이용하여 비교하였다. 두 클래스의 수행 성능 비교는 표 3과 같다.

(표 3) 성능 비교 (단위:ms)

메소드	구현한 BigInteger	JDK 1.1.8의 BigInteger
gcd	331	20
modPow	130	71
modInv	140	50

위의 성능평가에서는 동일한 연산을 1000회 반복한 시간이며, 단위는 ms(1/1000초)이다. 처리 가능한 최대 자료형이 JDK1.1.8에서는 8 바이트 길이의 Long형이고 자바 카드에서는 4바이트 길이의 integer형이다. 이러한 제한 때문에 본 논문에서 구현한 BigInteger 클래스가 JDK1.1.8보다 소요 시간이 오래 걸린다.

V. 결론

본 논문에서는 몇 가지 알고리즘을 이용하여 공개키 암호 시스템의 구현에 반드시 필요한 지수연산, 모듈러 연산과 최대공약수 연산 등을 제공하는 BigInteger 클래스를 구현하였다. 구현한 클래스가 속도가 다소 느리다는 단점은 있지만 한정된 자료형을 사용하는 자바 카드 상에서 동작하는 암호 시스템을 구현하기 위해서는 반드시 필요한 클래스이다.

본 논문에서 구현한 클래스를 이용하여 공개키 알고리즘과 전자 서명 알고리즘의 구현하는 것을 향후 연구과제로 남긴다.

참고 문헌

[1] 김연선, 이창욱,"자바카드 애플릿 설계 및 검증에 관한 연구"『한국통신정보보호학회 종합학술발표회 논문집』 2000. Vol.10 No.1 pp.805.

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

[3] 황효선, 임채훈, 이필중, "멱승 알고리즘의 구현과 분석" 『한국통신정보보호학회 학회지』 1995. Vol.5 No.1 pp.5.

[4] A.Menezes, P.van Oorschot, S.Vanstone, "Ha

ndbook of Applied Cryptography", CRC Pres s, 1996. pp.66

[5] A.Menezes, P.van Oorschot, S.Vanstone, "Ha ndbook of Applied Cryptography", CRC Pres s, 1996. pp.614

[6] Chen, Zhiqun, "Java Card Technology for S mart Cards", ADDISON-WESLEY Compan y, 2000 pp.9.

[7] Jonathan Knudsen, "Java Cryptography", O' Reilly 1998. pp.255

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

[9] http://java.sun.com/products/jdk/1.1/docs/api/j ava.math.BigInteger.html