

# 자바 카드 기반 RSA 알고리즘 구현

김성준\*, 이주영\*\*, 이재광\*

\*한남대학교 컴퓨터공학과

\*\*한국전자통신연구원

e-mail:sjkim@netwk.hannam.ac.kr

## Implementation of RSA Algorithm Based on JavaCard

Sung-Jun Kim\*, Joo-Young Lee\*\*, Jae-Kwang Lee\*

\*Dept. of Computer Engineering, Hannam University

\*\*Electronics and Telecommunications Research Institute

### 요약

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. 본 논문에서는 자바 카드 API를 이용하여 공개키 암호 알고리즘 중에서 대표적인 RSA 알고리즘을 구현하는 것에 목적을 둔다.

### 1. 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 실용화를 위한 준비단계에 있다.

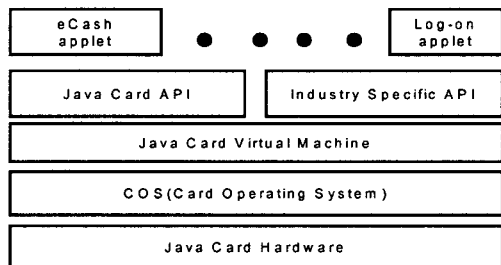
스마트 카드를 사용하는 가장 큰 목적은 카드 내에 저장된 데이터를 안전하게 보호하는 일이다. 지금까지 스마트 카드는 일반적인 컴퓨터 시스템에 대한 정보보호 예방 기술만을 제공해 왔으나, 안전한 인터넷 언어라고 알려진 자바 언어를 스마트 카드에 적용한 자바 카드는 스마트 카드가 가지는 정보보호 특성을 그대로 보존할 뿐만 아니라, 여러 사람에게 스마트 카드를 위한 프로그래밍 기술을 공개해줌으로써 스마트 카드 자체를 인터넷을 위한 하나의 새로운 응용 플랫폼으로 활용할 수 있도록 하였다.

본 논문에서는 자바 카드 API와 호환되는 RSA 알고리즘을 구현한다. 2장에서는 스마트 카드와 자바카드의 기본 구조 그리고 RSA 알고리즘을 살펴보고, 3장에서는 구현하였다. 4장에서는 구현한 클래스를 검증하고, 마지막으로 5장에서 결론을 맺는다.

### 2. 관련 연구

#### 2.1 자바 카드

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



[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. 이 후 발표된 자바 카드 API 1.0은 단지 명

세서의 역할만을 했다. 그러나 1997년 Sun Microsystem사에서 자바API의 일부 제한된 기능을 수행하는 자바 카드 API 2.0을 발표하였다[3]. 그 후 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 라이선스를 이미 받은 상태이다[5].

### 2.2 자바 카드 애플릿

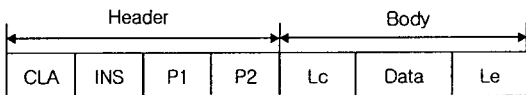
자바 카드 애플릿은 자바 카드 상에서 실행될 수 있는 자바 프로그램이다[3]. 애플릿과 호스트 간의 통신은 [그림 2]에서 나타낸 바와 같이 명령어 APDU와 응답 APDU로 구성되는 APDU 교환을 통해서 이루어진다. APDU 교환은 애플릿과 호스트 간에 직접 이루어지는 것이 아니라 JCRE를 매개로 하여 이루어지고, JCRE는 애플릿과 호스트 간에 교환되는 APDU의 관리와 감독 역할을 수행한다. 따라서 애플릿과 CAD(Card Access Device) 또는 호스트간의 직접적인 통신은 불가능하며, JCRE를 통한 통신만이 가능하다[3].



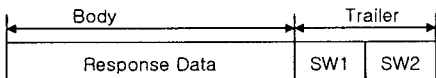
[그림 2] 애플릿과 호스트 응용프로그램간의 통신

### 2.3 APDU(Application Program Data Unit)

APDU는 카드 상의 통신에서 사용되는 전송 메시지의 형태로 ISO7816에 규정되어 있다. 전송방식은 명령(Command)과 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) APDU 구조

### 2.4 RSA 알고리즘

현재 공개키 암호기법들 중에서 가장 많이 사용되고 있는 알고리즘으로써, 1977년 Ron Rivest, Adi Shamir, 그리고 Leonard Adleman이라는 세 명의 수학자들에 의해 제안된 방식이다. 큰 소수의 Exponent Modulo 연산을 이용한 공개키 암호화 알고리즘 랜덤하게 구해진 파라미터인 P, Q로부터 e (encrypt exponent)와 d(decrypt exponent), n (modulus)를 구하고  $M^e \text{ mod } n$ 과  $C^d \text{ mod } n$  연산으로 암호화와 복호화를 수행한다[6].

- 송신 : 송신자 A는 B의 공개키 e를 획득하여  $C = M^e \text{ mod } n$ 를 계산하여 암호화된 메시지 C를 전송한다.
- 수신 : 수신자 B는 자신의 비밀키 d로  $M=C^d \text{ mod } n$ 를 계산하여 원래의 메시지 M을 얻는다.

RSA 알고리즘의 암호화와 복호화 단계를 간단히 표현하면 다음과 같은 수식으로 표현할 수 있다.

암호화	$C = M^e \text{ mod } n$
복호화	$M=C^d \text{ mod } n$

### 3. 구현

본 장에서는 RSA 알고리즘을 자바 카드 명세에 따라 구현하는 방법에 대하여 설명한다. 자바 카드 API에는 모듈러 지수 연산, 최대공약수 계산, 승산역원의 계산등 공개키 암호 알고리즘 구현에 반드시 필요한 연산 메소드를 지원하지 않는다. 따라서, 이미 개발한 자바 카드용 BigInteger 클래스를 이용하여 구현하였다.

#### 3.1 개발 환경

RSA 알고리즘의 구현 환경은 다음의 [표 1]와 같다.

[표 1] 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	

#### 3.2 BigInteger 클래스

BigInteger 클래스는 JDK1.3에 포함되어 있는 클래스이며, 정수의 범위를 넘어가는 수들에 대한 모듈러 지수 연산, 최대공약수 연산, 승산역원 그리고 소수

판정 및 생성등을 지원한다. 이 클래스는 공개키 암호 알고리즘에 반드시 필요하며 자바 카드 API는 제공하지 않기 때문에 구현을 한후, 이를 이용하여 RSA를 구현하였다.

### 3.3 RSA 클래스 구현

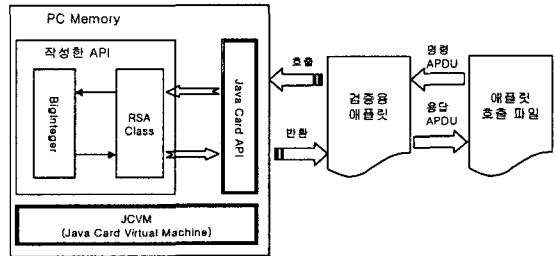
구현된 RSA에서 실제 암호화와 복호화 수식부분만을 표현하면 다음과 같다. 연산에 필요한 파라미터와 키를 처리하는 부분은 생략하였다.

```
// 암호화 메소드
private byte[] Encrypt(byte input[]){
    // 공개키에서 모듈러 nB 값을 읽음.
    short mod_len = pubKey.getModulus(mod_byte, 0);
    // 읽어온 mod_byte를 BigInteger형으로 변환
    BigInteger nB= new BigInteger(1, mod_byte);
    // 공개키에서 공개키 exponent e 값을 읽음.
    short exp_len = pubKey.getExponent(exp_byte, 0);
    // 읽어온 exp_byte를 BigInteger형으로 변환
    BigInteger e = new BigInteger(1, exp_byte);
    // 암호화할 메시지를 BigInteger형으로 변환
    BigInteger M = new BigInteger(1, input);
    // 암호화  $C = M^e \text{ mod } n$ 을 계산
    BigInteger C = M.modPow(e, nB);
    // 암호화된 C의 바이트 배열을 반환
    return C.toByteArray();
}

// 복호화 메소드
private byte[] Decrypt(byte input[]){
    // 개인키에서 모듈러 n 값을 읽음.
    short mod_len = priKey.getModulus(mod_byte, 0);
    // 읽어온 mod_byte를 BigInteger형으로 변환
    BigInteger n= new BigInteger(1, mod_byte);
    // 개인키에서 비밀키 exponent d 값을 읽음.
    short exp_len = priKey.getExponent(exp_byte, 0);
    // 읽어온 exp_byte를 BigInteger형으로 변환
    BigInteger d = new BigInteger(1, exp_byte);
    // 복호화할 메시지를 BigInteger형으로 변환
    BigInteger C = new BigInteger(1, input);
    // 복호화  $M = C^d \text{ mod } n$ 을 계산
    BigInteger M = C.modPow(d, n);
    // 복호화된 M의 바이트 배열을 반환
    return M.toByteArray();
}
```

### 4. 시뮬레이션

원칙적으로 개발한 RSA 알고리즘을 검증하기 위해서는 오프 카드 설치 프로그램과 온 카드 설치 프로그램을 이용하여 실제 자바 카드상에 설치한 후에 검증을 해야 하지만, Sun에서 제공하는 java card development kit을 이용하여 검증하였다. 이때 생성된 CAP파일의 정당성을 검증해주는 verifycap 명령을 이용하여 파일의 정당성을 검증받고, 실행은 작성한 API와 검증용 애플릿을 메모리에 함께 로드한다. 그 후 검증용 애플릿과 APDU 명령으로 서로 통신을 수행하는 애플릿 호출 파일을 작성해서 그 통신 내용을 확인하였다. 이 애플릿 호출 파일은 배치파일과 같이 동작한다. 검증하는 과정이 다음 [그림 5]에 나타난다.



[그림 5] 검증 애플릿 동작 과정

위의 [그림 5]에서 검증용 애플릿에서 암호화/복호화를 요청하면 자바카드 API가 이를 받아서 RSA 클래스를 호출한다. 이때, RSA 클래스가 BigInteger 클래스를 호출해서 연산을 수행하고, 결과를 반환하게 된다. 이 단계들은 모두 애플릿 호출 파일에서 APDU 명령어를 메모리 상에 로드되어 있는 애플릿에 전달되면서 실행된다.

아래에 검증용 애플릿에서 암호화와 복호화를 호출하는 부분이 있다.

```
private void proc(APDU apdu){
    // 주어진 파라미터로 공개키와 개인키를 생성.
    Cipher test =Cipher.getInstance(Cipher.ALG_RSA_PKCS1, true);
    // 암호화를 위한 초기화
    test.init(pubKey , Cipher.MODE_ENCRYPT);
    // 암호화
    short number = test.doFinal(M, 0, M.length, C, 0);
    // 복호화를 위한 초기화
    test.init(priKey, Cipher.MODE_DECRYPT);
    // 복호화
    number = test.doFinal(C, 0, C.length, M, 0);
}
```

다음은 검증용 애플릿과 통신을 위한 APDU 명령어로 구성된 파일이다. 이 파일은 카드의 호스트 프로그램에서 요청해야 하는 것이지만, 시뮬레이션 환경에서는 호스트에서 요청되는 명령어들을 배치 파일의 형태로 구성하고, 애플릿과 통신하도록 하였다. 아래에서 암호문을 4번에 걸쳐서 요청을 하는데, 이는 APDU 명령어의 Data는 최대 32Byte를 넘을 수 없도록 되어 있기 때문에 32바이트씩 분할해서 반환을 요청하였다[4].

```

    ... (생략) ...
// 평문 메시지 입력
0x00 0x10 0x00 0x00 0x14 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x02;
// Call RSA 호출 - 암호화와 복호화 수행
0x00 0x70 0x00 0x00 0x00 0x02;
// Call 암호문 블록 1 요청
0x00 0x21 0x00 0x00 0x00 0x7F;
    ... (생략) ...
// Call 복호화 결과 요청
0x00 0x25 0x00 0x00 0x00 0x7F;
    ... (생략) ...
    
```

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: 14, 01, 01, 01, 01, 01,
01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, Le:
00, SW1: 90, SW2: 00
CLA: c0, INS: 70, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90,
SW2: 00
CLA: c0, INS: 21, P1: 00, P2: 00, Lc: 00, Le: 20, 1c, 27, f1,
53, 86, 09, 88, 68, 5f, de, 1c, 6e, 75, 9e, b4, 3b, 2c, cf, 7f,
b1, 9e, 5a, 89, cb, 4a, a7, 7f, fa, d2, b9, e5, 62, SW1: 90,
SW2: 00
    ... (생략) ...
CLA: c0, INS: 25, P1: 00, P2: 00, Lc: 00, Le: 14, 01, 01,
01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01,
01, 01, SW1: 90, SW2: 00
    ... (생략) ...
    
```

위의 실행결과에서 CLA는 명령어를 수행할 애플릿을, INS는 명령어의 종류를, Lc는 데이터의 길이를 의미하고, Le는 응답 데이터의 길이를 의미한다. 즉, 첫줄에서 Lc 14는 평문 20바이트가 입력된다는 것을 의미하며, 여섯 번째 줄의 Le 20은 암호화된 결과 중에서 32바이트만을 요청한 것이다. 이 블록은 총 4개이며, 마지막 부분에서 다시 복호화를 수행해서 원래의 평문 20바이트가 반환된 것을 볼 수 있다. 상태 코드 SW1이 모두 90으로 성공적으로 응답이 완료되었다는 것을 확인할 수 있다.

5. 결론

본 논문에서는 자바카드 API에서 지원하지 않는 BigInteger 클래스를 이용하여 공개키 암호 알고리즘인 RSA 알고리즘을 구현하였다. 일반적으로 자바 카드와 같은 IC 카드에서는 별도의 수치 연산기를 하드웨어적으로 구현하여 공개키 암호 알고리즘의 연산을 수행하지만, PC 환경에서 이를 이용하기란 불가능하다. 만약 실제 카드 상으로 이식을 하더라도, 연산 부분을 호출하는 부분이 별도의 메소드로 구현이 되어 있기 때문에, 연산 부분만을 대처하면 사용이 가능하다. 새로운 공개키 암호 알고리즘으로 대두되고 있는 타원곡선 암호 알고리즘의 구현은 향후 연구 과제로 남긴다.

[1] 김연선, 이창욱, "자바카드 애플릿 설계 및 검증에 관한 연구", 한국통신정보보호학회 종합학술발표회 논문집, Vol.10 No.1 pp.805, 2000.  
 [2] 문상재 외, "차세대 IC 카드를 사용한 정보보호 신기술 시스템 개발", pp.17, 정보통신부 1997.  
 [3] Chen, Zhiqun, "Java Card Technology for Smart Cards", pp.42-77. ADDISON-WESLEY Company, 2000  
 [4] "Java Card™ 2.1.2 Development Kit User's Guide", pp.66. Sun Microsystems, Inc. 2001.  
 [5] <http://java.sun.com/products/javacard/datasheet.html>  
 [6] "PKCS#1: RSA Encryption Standard", pp.10-11, RSA laboratories, 1993