理學碩士 學位論文

RSA 공개키 암호시스템의 구현

A Realization of RSA Public-key Encryption

指導教授　裵　在　國

2001年 2月

韓國海洋大學校 大學院

應 用 數 學 科

金　錦　喆

本 論文을 金錦喆의 理學碩士 學位論文으로 認准함

主審 : 理學博士　琴 尙 昊

委員 : 工學博士　金 宰 煥

委員 : 理學博士　裵 在 國

2001年 2月

韓國海洋大學校 大學院

應 用 數 學 科

金 錦 喆

목 차

ABSTRACT

약 4000년 전에 처음 사용되기 시작한 암호는 제2차 세계대전을 거치면서 비약적인 발전을 거듭하게 되었는데 그 중에서 1978년 미국의 MIT 대학에서 R. Rivest, A, Shamir, L. Adleman에 의해 개발된 RSA 공개키 암호 시스템은 그 안전성을 수학적인 근거 즉, 큰 정수의 소인수분해가 어렵다는 사실에 근거를 둔 것으로써 지금까지 가장 널리 사용되고 있는 암호 시스템이다. 본 논문에서는 이를 C++로 구현하고자 했는데, 보내고자 하는 메시지 M은 대칭키 암호 시스템 중에서도 호환 암호를 사용하여 M'으로 암호화하였으며, 이를 암호화하는데 필요한 키 m은 반드시 비밀화 되어야 하므로 RSA 공개키 암호 시스템을 사용하여 c로 암호화 시켜서 결국 (M, c)를 전송하고 복호시에는 이와 반대과정을 수행함으로써 가능하게 하였다. 여기서 호환 암호에서 사용되는 키는 $\{1, 2, ,3, . . . , 128\} \rightarrow \{1, 2, 3, . . . , 128\}$인 permutation을 사용하였고, RSA 암호 시스템에서의 두 개의 소수 변수는 보다 높은 안전성을 위하여 1024비트의 무작위 소수를 사용하였다.

1. Introduction

현대 암호학은 개인의 프라이버시를 보호하고 국제적으로 범죄를 예방하거나 국가적 정보를 보호해 주는 데 결정적인 역할을 한다. 암호학은 원래 군사와 외교적인 목적에서 주로 알려져 왔으나, 최근 컴퓨터 통신망과 관련된 범죄가 빈번하게 발생함에 따라 더욱 일반의 주목을 받게 되었다. 정보화 시대에서의 정보는 기존의 재화나 용역이 갖는 일반적인 가치 이상의 가치를 갖는다. 더욱이 요즘과 같은 무한경쟁시대에서 중요한 정보의 유출이나 잘못된 정보의 획득, 그리고 보유하고 있던 정보의 망실과 같은 일은 당사자에게 매우 치명적일 수 있다. “정보의 안전한 관리”는 아무리 강조해도 지나침이 없다. 요즘시대에 정보의 안전한 관리는 바로 암호를 통해서 이루어진다고 해도 과언이 아니다.

암호의 사용은 약 4000년 전 이집트인에 의해 처음 사용되어졌다고 알려져 있는데 제 2차 세계대전을 계기로 급속도로 발전하기 시작하였다[3]. 그러나, 예전에는 암호학이 그렇게 발달하지 않아서 보통때는 보이지 않다가 열을 가하면 보이는 문자를 쓴다던지 하는 방식으로 암호학에 대한 요구를 해결했다[4]. 요즘과 같이 인터넷이 발달하고, 전자 상거래의 싹이 쑥쑥 자라고 있는 시점에서는, 자신의 데이터를 다른 사람이 알지 못하게 하는 암호화에 대한 요구가 부쩍 증가하게 되었다.

최근 수학적인 근거를 이용하여 사용되고 있는 암호 시스템은 대칭키(symmetric-key) 암호 시스템과 공개키(public-key) 암호 시스템이 있다. 본 논문에서는 대칭키 암호 시스템을 이용하여 메시지 M을 암호화 시키고 이를 복호화 하는 데 필요한 비밀키 m은 공개키 암호 시스템을 사용하여 암호화 시키는 방법을 사용하였다.

(1) 호환 암호(Transposition cipher)

호환 암호는 대칭키 암호 시스템의 일종이다. K를 암호화 및 복호화 하는 모든 키들의 집합 즉 키공간이라 하고 $E \subseteq K$를 암호화 하는 데 필요한 키들의 집합이라 하며, $D \subseteq K$를 복호화 키들의 집합이라고 하자.

정의 1.1 (대칭키) 키공간 K에서 암호화와 복호화 변환(transformation) $\{E_e : e \in K\}$과 $\{D_d : d \in K\}$ 를 생각하자. 각각의 키 순서쌍 (e, d)에서 d만을 알고도 e를 쉽게 계산할 수 있을 때 이를 대칭키라고 한다.

예 1.2 $A = \{A, B, C, ..., X, Y, Z\}$라 하고 M과 C를 A에서 길이가 5인 문자들의 집합이라 하자. 키 e는 A에서의 permutation으로 석택되어진다. 암호화 하기 위해서 메시지를 5개의 문자로 된 그룹으로 나누고 permutaion e를 각각에 적용시킨다. 이를 복호화 하기 위하여 inverse permutation $d = e^{-1}$를 계산하면 된다. 예를 들어,

$$e = \begin{pmatrix} A & B & C & D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z \\ D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z & A & B & C \end{pmatrix}$$

라 하면 메시지

$$m = THISC\ IPHER\ ISCER\ TAINL\ YNOTS\ ECURE$$

은

$$c = E_e(m) = WKLVF\ LSKHU\ LVFHU\ WDLQO\ BQRWV\ HFXUH$$

로 암호화 된다.

대칭키 시스템의 중요한 문제 중의 하나는 양쪽에서 동의하고 키를 안전하게 바꿀 수 있는 방법을 찾는 것이다. 모든 사람이 이러한 암호화

기술을 알고 있다면 복호화 키 d는 반드시 비밀화 되어야 한다. 또한 대칭키 암호에서 d가 비밀화 되어야 한다는 것은 곧 e도 비밀화 되어야 한다는 것을 뜻한다[1].

대칭키 암호 기술에서 가장 잘 알려진 것은 블록 암호(block cipher)이다[7]. 또한 블록 암호는 크게 치환 암호(substitution cipher)와 호환 암호(transposition cipher)의 두 가지로 나뉘어진다.

정의 1.3 (호환 암호) 블럭의 길이가 t인 대칭키 암호 기술을 생각하자. K를 집합 $\{1, 2, 3, ..., t\}$에서의 모든 permutation들의 집합이라 하자. 각각의 $e \in K$에 대하여

$$m = (m_1 m_2 ... m_t) \in M$$

일 때 암호화 함수

$$E_e(m) = (m_{e(1)} m_{e(2)} ... m_{e(t)})$$

를 정의한다. 이러한 모든 permutation들의 집합을 호환 암호라고 한다. e에 대응하는 복호화 키 d는 e의 inverse permutation $d = e^{-1}$이다.

$$c = (c_1 c_2 ... c_t)$$

를 복호화 하기 위해서

$$D_d(c) = (c_{d(1)} c_{d(2)} ... c_{d(t)})$$

를 계산한다.

고정된 주기 t를 가지는 호환 암호에서 암호화를 위해서 평문(plaintext)을 t개의 문자들의 블럭으로 그룹을 나누고 각각의 블럭들을 하나의 permutation e에 적용시켜야 한다. 더 정확히 말해서 평문 블럭

$$m = m_1 m_2 ... m_t$$

에 대응하는 암호문(ciphertext)은

$$c = E_e(m) = m_{e(1)} m_{e(2)} \ldots m_{e(t)}$$

이다. 여기서 암호화 키는 e**이고 키공간** K**에는 주어진** t**에 대하여** $t!$ **개의 원소를 가진다. 복호화는** $d = e^{-1}$**를 구함으로써 할 수 있다**[10].

예 1.4 $t = 6$**이고** $e = (6\ 4\ 1\ 3\ 5)$**인 호환 암호를 생각하자. 메시지**

$$m = CAESAR$$

은 e**에 의해**

$$c = RSCEAA$$

로 암호화 된다. 복호화는 e**의** inverse permutation

$$d = e^{-1} = (3\ 6\ 4\ 2\ 5\ 1)$$

를 구함으로써 할 수 있다.

대칭키 암호 기법에서 가장 큰 문제점은 바로 메시지의 전달자와 수신자가 똑같은 비밀키를 다른 사람들이 모르게 공유해야 한다는 점이다. 특히 메시지의 전달자와 수신자가 물리적으로 서로 떨어져 있다면 그 두 사람은 매우 안전한 통신수단을 이용하여 비밀키를 전달하여야 할 것이다. 만약 공격자가 이 두 사람간의 통신경로상에서 비밀키를 얻어 낸다면 그 후로부터 메시지의 송신자와 수신자가 그 비밀키를 사용하여 메시지를 암호화 한다 하여도 그 공격자는 자신이 훔친 비밀키를 이용하여 모든 메시지를 복호화 하여 그 내용을 알 수가 있게 될 것이다. 또한 비밀키 암호 기법에 있어서 모든 키들은 그것을 사용하는 두 사람 이외에는 아무도 알 수 없도록 비밀리에 보관되어야 하므로 만약 수많은 사람들이 동시에 상호 메시지를 주고 받고자 하는 경우에는 이러한 비밀키의 관리가 매우 어려워진다. 이와 같이 암호화를 위한 키의 생성과 전달, 그리고 보관하는 문제 즉, 키의 관리 문제는 암호를 이용한 정보보안에 있어서 중요한 이슈가 되는 문제이다. 이러한 문제를 해결하

기 위해서 공개키 암호기법을 사용하게 되는데, 대칭키 기법에서의 암호화 키 e를 공개키 암호 시스템을 이용하여 암호화 시켜서 전송할 수 있다.

(2) 공개키 암호 시스템

K가 키공간일 때, $\{E_e : e \in K\}$를 암호화 변환(transformation)의 집합이라 하고 $\{D_d : d \in K\}$를 이에 대응하는 복호화 변환의 집합이라 하자. 결합된 암호화/복호화 변환의 순서쌍 (E_e, D_d)에서 각각의 순서쌍은 E_e를 알고도 주어진 무작위 암호문(random ciphertext) $c \in C$에서 $E_e(m) = c$를 만족하는 메시지 $m \in M$을 찾는 것이 어렵다는 성질을 갖는다고 하자. 이 성질은 암호화 키 e를 통해서 복호화 키 d를 찾는 것이 어렵다는 것을 뜻한다.

__정의 1.5 (one-way function)__ $f : X \to Y$에서 모든 $x \in X$에 대해서는 $f(x)$를 계산하기는 쉽지만 대부분의 원소 $y \in Im(f)$에 대해서 $f(x) = y$를 만족하는 $x \in X$를 구하기가 계산적으로 어려울 때 함수 f를 one-way function이라 한다.

__정의 1.6 (trapdoor one-way function)__ one-way function $f : X \to Y$인데 주어진 정보에 의해서 $y \in Im(f)$에 대하여 $f(x) = y$를 만족하는 $x \in X$를 구하기가 쉬울 때 함수 f를 trapdoor one-way function이라 한다.

예 1.7 $X = \{1, 2, 3, ..., n-1\}$에서 $f: X \to Y$를 $f(x) = x^3 \pmod n$ 이라 정의하자. 모든 $x \in X$ 에 대하여 $f(x)$를 구하기는 쉽지만 반대의 과정은 그렇지 못하므로 f는 one-way function이다. 그러나 여기서 두 개의 소수 $p = 48611$, $q = 53993$을 선택하고 $n = pq = 2624653723$으로 정의 했을 때 p와 q의 값을 안다면 Chinese Remainder Theorem을 이용해서 반대 과정을 쉽게 수행할 수 있다.

이러한 trapdoor one-way function은 공개키 암호 시스템의 기본이 된다[9]. Bob과 Alice가 서로 메시지를 주고 받기를 원한다고 가정하자. Bob은 키 순서쌍 (e, d)를 선택해서 암호화 키 e(이를 공개키라 부른다)를 어떤 채널을 통해서 Alice에게 보내고 복호화 키 d(이를 비밀키라 부른다)는 비밀화 시킨다. Alice는 Bob의 공개키 e를 받고 $c = E_e(m)$을 얻기 위해 e를 적용시켜 메시지 m을 Bob에게 보낸다. Bob은 암호문 c를 복호화 하기 위해서 역변환 D_d를 적용시킴으로써 메시지 m을 얻을 수 있다.

정의 1.8 (공개키 암호 시스템) 암호화와 복호화의 변환 $\{E_e : e \in K\}$, $\{D_d : d \in K\}$에서 각각이 결합된 암호화/복호화 순서쌍 (e, d)에 대하여 e는 공개화 시키고 d는 비밀화 시킬 때 이를 공개키 암호 시스템이라 한다. 안전성을 위해서 e로부터 d를 구하는 것은 계산적으로 어려워야 한다[5].

공개키 암호 시스템에서는 소수(prime number) 변수를 효과적으로 생성하는 것이 중요하다. 이러한 경우에 소수는 충분히 큰 사이즈가 되어야 하고 특정 소수 p가 선택되어 졌을 때 공격자가 확률적으로 알기

어려운 무작위(random) 소수가 생성되어져야 한다. 소수를 생성하는 가장 일반적인 방법은 적당한 크기의 무작위 수 n을 뽑아서 n이 소수인지를 검정하는 것인데 이는 n이 $\sqrt{n}$보다 작은 소수에 의해서 나누어지는지를 조사 함으로써 가능하다[2]. 그러나 이러한 방법도 n이 상당히 큰 경우에는 많은 시간이 소요되기 때문에 좋은 방법이라고는 할 수 없다. 확률적 소수 검정(primality test)에서 현재 가장 많이 쓰이고 있는 방법은 Miller-Rabin 테스트이다. 이 Miller-Rabin 테스트는 다음의 사실에 근거를 두고 있다.

n이 홀수인 소수라 하고 r이 홀수일 때

$$n-1=2^s r$$

이라 하자. 또한 a를

$$\gcd(a,n)=1$$

을 만족하는 정수라 하면 $0 \le j \le s-1$인 j에 대하여

$$a^r \equiv 1 \pmod{n} \text{ 또는 } a^{2^j r} \equiv -1 \pmod{n}$$

이다[8].

알고리즘 1.9 (Miller-Rabin의 확률적 소수 검정)

MILLER-RABIN (n, t)

INPUT: an odd integer $n \geq 3$ and security parameter $t \geq 1$

OUTPUT: answer "prime" or "composite" to the question :

"Is n prime?"

1. Write $n-1 = 2^s r$ such that r is odd
2. For i from 1 to t do the following:
   - 2.1. Choose a random integer a, $2 \leq a \leq n-2$
   - 2.2. Compute $y = a^r \pmod{n}$
   - 2.3. If $y \neq 1$ and $y \neq n-1$ then do the following:
     - $y \leftarrow 1$
     - While $j \leq s-1$ and $y \neq n-1$ do the following:
       - Compute $y \leftarrow y^2 \pmod{n}$
       - If $y = 1$ then return ("composite")
       - $j \leftarrow j+1$
     - If $y \neq n-1$ then return ("composite")
3. Return ("prime")

Miller-Rabin **테스트를 임의의 합성수** n**에 대하여** t**번 시행했을 때** n**이 소수라고 판단할 확률은** $\frac{1}{4^t}$**보다 작다는 것이 알려져 있다**[6].

Miller-Rabin **테스트와 같은 확률적인 검정과정을 거친 정수** n**을 소수라고 단정지을 수는 없다. 이러한 수를 후보 소수**(probable prime)**라고 하는데 이를 다시 확실한 소수임을 검증하기 위해 다음의 사실을 이용한다**.

$n \geq 3$인 정수라 하면, n이 소수일 필요충분조건은

(1) $a^{n-1} \equiv 1 \pmod{n}$과

(2) $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ for each prime divisor q of $n-1$

을 만족하는 정수 a가 존재한다.

위의 결과는 Z_n^*의 order가 $n-1$인 사실로부터 나온 것이다. 이러한 검증을 True primality test라 하고 이 검정을 통과한 정수 n을 provable prime이라 한다.

2. RSA 암호 시스템

RSA 암호 시스템은 1978년 R. Rivest, A. Shamir, L. Adleman에 의해 개발되었는데 현재 가장 늘리 쓰이고 있는 공개키 암호 시스템이다. 이 암호 시스템의 안전성은 큰 정수의 소인수분해가 어렵다는 사실에 기반을 두고 있다. 그리고 키를 생성하는 문제에 대해서는 무작위(random)소수 p와 q를 이용해서 공개키와 비밀키를 생성하고 이것으로 암호화 시킬 수 있어야 한다. 이 때 비밀키는 공개키를 통해서 구해져야 하는데 이는 다음의 정리에 근거를 두고 있다.

<u>정리 2.1</u> p와 q를 서로 다른 홀수의 소수라 하고 $n = pq$라 하자.

$$\gcd(e, (p-1)(q-1)) = 1$$

을 만족하는 양의 정수 e와 정수 c에 대해서

$$x^e \equiv c \pmod{n}$$

을 만족하는 x가 유일하게 존재한다.

<u>증명</u> 함수 $f : \{0, 1, 2, ..., n-1\} \to \{0, 1, 2, ..., n-1\}$를

$$f(x) \equiv x^e \pmod{n}$$

으로 정의한다. 그러면 함수 f가 전단사(bijection)임을 보이면 충분한데

$$|\{0, 1, 2, ..., n-1\}| < \infty$$

이므로 단사(injection)인 것만 보이면 충분하다.

$$x, y \in \{0, 1, ,2, ..., n-1\} \text{ and } x^e \equiv y^e \pmod{n}$$

이라 가정하자.

(1) p**가** x**를 나누면** $x = px_1$, $y = py_1$ for some integer x_1, y_1, **따라서**

$$p^e x_1^e \equiv p^e y_1^e \pmod{pq}.$$

또한 $n = pq$**이고** n**은** $p^e(x_1^e - y_1^e)$**을 나누므로** q**는** $x_1^e - y_1^e$**을 나눈다. 따라서**

$$x_1^e \equiv y_1^e \pmod{q}$$

가 성립한다. 만약 q**가** x_1**을 나눈다면** q**는** y_1**도 나눈다. 따라서**

$$x \equiv 0 \pmod{n}, \quad y \equiv 0 \pmod{n}$$

이 성립한다. 따라서

$$x = y = 0$$

이다. 만약 q**가** x_1**을 나누지 못하면** q**는** y_1**나누지 못한다.** field Z_q**에서의 모든 경우를 생각하면**

$$\left(\frac{x_1}{y_1}\right) = 1 \text{ and } e^{q-1} = 1$$

이 된다. 따라서

$$ea + (q-1)b = 1$$

을 만족하는 정수 a, b**가 존재한다.**

$$\frac{x_1}{y_1} = \left(\frac{x_1}{y_1}\right)^{ea+(q-1)b} = \left(\left(\frac{x_1}{y_1}\right)^e\right)^a \left(\left(\frac{x_1}{y_1}\right)^{q-1}\right)^b = 1^a \cdot 1^b = 1$$

이므로

$$x_1 \equiv y_1 \pmod{q}$$

가 성립한다. 따라서 q**가** $(x_1 - y_1)$**을 나누고 이에 의해** pq**는**

$p(x_1 - y_1)$을 나눈다.

$$p(x_1 - y_1) = x - 1$$

이므로

$$x = y$$

가 성립한다.

(2) 만약 p가 x를 나누지 못한다면 p는 y도 나누지 못한다.

$$x^e \equiv y^e \pmod{p}$$

이고 field Z_p에서

$$\left(\frac{x}{y}\right)^e = 1, \quad (e, p-1) = 1$$

이된다. 따라서

$$ea_1 + (p-1)b_1 = 1$$

을 만족하는 정수 $a_{1,}\ b_1$이 존재한다.

$$\frac{x}{y} = \left(\frac{x}{y}\right)^{ea_1 + (p-1)b_1} = \left(\left(\frac{x}{y}\right)^e\right)^{a_1}\left(\left(\frac{x}{y}\right)^{p-1}\right)^{b_1} = 1^{a_1} \cdot 1^{b_1} = 1$$

이므로

$$x \equiv y \pmod{p}$$

가 된다. 같은 방법으로,

$$x \equiv y \pmod{q}$$

도 성립한다.따라서 p는 $x - y$를 나누고 q도 $x - y$를 나눈다. 한편 어떤 정수 s에 대하여

$$x - y = ps$$

로 표현할 수가 있고 그러면 q는 ps를 나누고 또한 q는 s도 나눈다. 그러므로 어떤 정수 r에 대해서

$$s = qr$$

로 표현할 수 있다. 따라서

$$x - y = sr = pqr = nr$$

이 성립하고 이에 의해 n은 $x - y$를 나눈다. 따라서

$$x \equiv y \pmod{n}$$

이 성립하게 된다.

위의 정리를 통해서 구해진 x가 RSA 공개키 암호 시스템에서의 비밀키로 사용되어진다. RSA 공개키 암호 시스템의 알고리즘은 아래와 같이 키를 생성하는 알고리즘과 생성된 키를 이용해서 메시지 m을 암호화 하고 복호화 하는 알고리즘으로 구성되어져 있다.

<u>**알고리즘 2.2 (RSA의 키생성)**</u>

SUMMARY: each entity creates an RSA piblic key and a corresponding private key

Each entity A should do the following:

1. Generate two large random (and distinct) primes p and q, each roughly the same size.
2. Compute $n = pq$ and $\varnothing = (p-1)(q-1)$
3. Select a random integer e, $1 < e < \varnothing$ such that
$$\gcd(e, \varnothing) = 1$$
4. Compute the unique integer d, $1 < d < \varnothing$, such that
$$ed \equiv 1 \pmod{\varnothing}$$
5. A's public key is (n, e); A's private key is d.

여기서 $n = pq$이 공개되어지는 키이므로 단계 1에서 생성되어지는 무작위 소수 p, q는 충분히 큰 사이즈가 되어야 한다. p, q가 큰 정수가 아니라면 n을 소인수분해하기가 쉬워지기 때문에 안전성은 깨지게 된다.

알고리즘 2.3 (RSA 공개키 암호시스템)

SUMMARY: B encrypts a message m for A, which A decrypts

1. Encryption: B should do the following:
   (a) Obtain A's authentic public key (n, e)
   (b) Represent the message as an integer m in the interval $[0, n-1]$
   (c) Compute $c = m^e \pmod n$
   (d) Send the ciphertext c to A
2. Decryption: To recover plaintext m from c,
   A should do the following:
   (a) Use the private key d to recover $m = c^d \pmod n$

복호화 작업의 증명 $ed \equiv 1 \pmod{\varnothing}$이므로 $ed = 1 + k\varnothing$인 정수 k가 존재한다. $\gcd(m, p) = 1$이면 Fermat의 정리에 의해

$$m^{p-1} \equiv 1 \pmod p$$

가 된다. 이 식의 양 변에 $k(q-1)$제곱하고, m을 곱하면 식은 다음과 같다.

$$m^{1+k(p-1)(q-1)} \equiv m \pmod p$$

한편 만약 $\gcd(m, p) = p$이면 modulo p의 0 양변이 congruent 하기 때문에 위의 합동식은 유효하다. 그러므로 모든 경우에 대해

$$m^{ed} \equiv m \pmod{p}$$

같은 방식으로

$$m^{wd} \equiv m \pmod{q}$$

결국, p와 q가 서로 다른 소수이므로

$$m^{ed} \equiv m \pmod{n}$$

이 된다. 따라서

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

을 얻을 수 있다.

예 2.4 (RSA encryption) Alice는 소수 $p = 2357$, $q = 2551$을 선택하고 $n = pq = 6012707$을 계산하며, $\varnothing = (p-1)(q-1) = 6007800$을 구한다. Alice는 또한 $e = 3674911$을 선택하고 $ed \equiv 1 \pmod{\varnothing}$를 만족하는 $d = 422191$ 을 계산한다. Alice의 공개키는 순서쌍 $(6012707, 3674911)$이고 비밀키는 $d = 422191$이다.

암호화: 메시지 $m = 5234673$ 을 암호화 하기 위해 Bob은

$$c = m^e \pmod{n} = 5234673^{3674911} \pmod{6012707} = 3650502$$

를 계산한다. 이때 c를 Alice에게 보낸다.

복호화: Alice는

$$c^d \pmod{n} = 3650502^{422191} \pmod{6012707} = 5234673$$

을 얻는다.

이제 RSA 공개키 암호 시스템의 안전성에 대하여 알아보자. 공격자의 목적은 공개된 정보 (n, e)를 이용해 암호문 c에 대응하는 평문 m을 찾는 것인데 이 문제를 RSA 문제(RSAP)라고 한다.

공격자가 RSA 문제를 해결할 수 있는 방법은 n을 소인수분해 해서

$\varnothing$와 d를 구하는 것이다. 일단 d가 구해지면 공격자는 암호문 c를 복호화 할 수 있다. 따라서 공개키 (n, e)로부터 비밀키 d를 구하는 문제는 n을 소인수분해 하는 문제와 계산적으로 같다는 것을 알 수 있다.

다른 방법으로 공격자가 $n = pq$와 $\varnothing$를 알고 있다면 다음의 두 방정식에 의해 쉽게 n을 소인수분해 할 수 있다. 미지수 p, q에 대해

$$n = pq, \quad \varnothing = (p - 1)(q - 1)$$

두 번째 식에 $q = \frac{n}{p}$을 대입함으로써 다음과 같은 미지수 p에 대한 이차방정식을 얻는다.

$$p^2 - (n - \varnothing + 1)p + n = 0$$

이 방정식의 근은 p와 q일 것이다. 따라서 공격자는 $\varnothing$를 알 수 있게 되고 그 경우 이 암호계는 깨지게 된다. 하지만 $\varnothing$를 안다는 것은 n을 소인수분해 하는 문제와 같으므로 이 작업 또한 쉽지가 않다.[11]

3. 구현 과정

지금까지 설명한 시스템을 기반으로 해서 실제로 암호 시스템을 구현하고자 한다. 여기서 사용된 프로그램 언어는 C++언어인데 시스템의 안전성을 보장하기 위해서는 C++에서 제약을 받는 사이즈가 큰 정수의 사용이 문제시 된다. 이 문제를 극복하기 위해서 큰 정수의 사용을 가능하게 해 주는 ntl이라는 헤드파일을 사용하였다. 구현 과정의 단계는 다음과 같다.

먼저 Alice가 Bob에게 보내고자 하는 메시지 M을 호환 암호를 이용해 암호화 한다. 이때

$$v : \{1, 2, 3, ..., k = 128\} \rightarrow \{1, 2, 3, ..., k = 128\}$$

인 permutation v를 생성하여야 하는데 이는 무작위(random)로 뽑은 것이다. 여기서 생성되어진 v의 숫자들을 그대로 붙여서 하나의 큰 정수로 만들어야 하는데 이를 위해서는 v에 오는 첫 번째 수는 반드시 100보다 큰 정수이어야 한다. 그래서 처음 오는 수 $v[0]$는 구간 $[100, k = 128]$에서 무작위로 선택하고 난 다음 나머지

$$v[1], v[2], v[3], ..., v[127]$$

을 선택한다. 이 때 숫자들을 무작위로 선택하기 때문에 중복되는 경우가 생길 수도 있는데 이를 방지하기 위하여 두 개의 배열 $v[i]$와 $v1[i]$를 선언한다.

$0 \le i \le k - 1$ 인 모든 i에 대하여

$$v1[i] = -1$$

이라 정의하고

$$v[i] = a \text{일 때 } v1[a] = 0$$

으로 놓는다. 따라서 한 번 선택되어진 정수 a에 대해서는 $v1[a] = 0$ 이므로 $i \neq j$인 j에 대해서 $v[j] = 0$이면

$$v[j] = j + 1 \pmod{128}$$

로 놓으면 완전한 permutation v를 생성할 수 있다.

다음은 m을 계산하는 방법인데 만약

$$v[i] = a, \quad v[i+1] = b$$

라면

$$m = (a * 1000) + b$$

를 계산 함으로써 m을 구할 수 있다.

<u>**예 3.1**</u> $k = 10$이라 한다면 random permutation $v = (10\ 3\ 8\ 7\ 4\ 1\ 9\ 6\ 2\ 5)$이 되고 $m = 1003080704010906020 5$이 된다.

다음은 Alice가 Bob에게 전달하고자 하는 메시지 M을 대칭키를 사용하여 암호화 하는 과정이다. 우선 전하고자 하는 메시지 M에 대해서 k개의 문자 단위로 읽어 들인 다음 각각의 문자에 대하여

$$v2[0], v2[1], v2[2], \ldots, v2[k-1]$$

을 정의하고

$$M'[i] = v2[v[i]-1]$$

로 치환 함으로써 암호화 된 메시지 M'을 구할 수 있다. 여기서 문제가 되는 것은 k 개의 문자 단위로 읽어 들이고 남는 나머지 문자들의 처리문제이다. 만약 남아 있는 문자의 개수가 k보다 현저히 작다면 본 시스템의 안전성이 무너질 수 있으므로 메시지 M이 끝나고 마지막 문

자에는 특수문자를 사용하고 그 다음부터 k 번째 까지는 그 이전에 사용한 문자를 그대로 둠으로써 메시지 M의 끝임을 분명하게 알리는 방법을 사용하였다. 따라서 메시지를 복호화 하는 사람은 특수문자가 나온 다음의 문자는 무시함으로써 원래의 문장을 복구할 수 있는 것이다.

<u>예 3.2</u> M = korea maritime university !

은

$v = (10\ 3\ 8\ 7\ 4\ 1\ 9\ 6\ 2\ 5)$

에 의해

M' = iramekr oaeminetvui eii^Qtrv !sy

로 암호화 된다. 여기서 ^Q는 특수문자이다.

다음 단계는 RSA 공개키 암호 시스템에서의 공개키 (n, e)와 비밀키 d를 생성하는 과정이다. 최근 RSA 시스템에 사용되는 변수는 적어도 512비트를 사용하고 있으나 본 논문에서는 확실한 안전성을 위하여 1024비트의 소수 p와 q를 사용하였다. 여기서 p와 q는 무작위 소수이고

$$n = pq, \quad a = (p-1)(q-1)$$

이며 e는

$$\gcd(e, a) = 1$$

을 만족하는 구간 $[1, a]$에서의 무작위 수이다. 그리고 d는 구간 $[1, a]$에서 $ed \equiv 1 \pmod{a}$를 만족하는 유일한 정수이다.

<u>예 3.3</u> p, q의 비트수를 40이라 하고

$$p = 1033457003507, \quad q = 578562833609$$

이 선택되어졌다고 하면 공개키는 순서쌍

$$(n, e) = (5979198123620761704667 63, 909643223635314406911 85)$$

이고 비밀키는 $d = 354507373090120790659489$이다.

호환 암호를 이용하여 생성한 메시지 m을 공개키 (n, e)를 사용해서 암호문 c로 암호화 하는 과정이다. 이 과정은

$$c = m^e \pmod{n}$$

을 계산 함으로써 수행된다.

<u>예 3.4</u> $n = 597919812362076170466763$, $e = 909643223635314406911185$, 그리고

$m = 100308070401090602 05$

이라하면

$c = m^e \pmod{n} = 208131164331094228137 71$

을 구할 수 있다.

Alice가 위와 같은 과정을 통해 구한 (M', c)를 Bob에게 보내게 된다. 그러면 Bob은 자신의 비밀키 d를 이용하여 c에 대응하는 m을 구하고 다시 m을 이용해서 M'에 대응하는 M을 구하게 된다. 먼저 m을 구하는 과정은 비밀키 d와 공개키 중의 하나인 n을 사용해서 $m = c^d \pmod{n}$을 구할 수 있다.

<u>예 3.5</u> 앞의 예에서

$c = 20813116433109422813771$, $d = 354507373090120790659489$ 그리고 $n = 597919812362076170466763$이므로

$m = c^d \pmod{n} = 1003080704010906020$5을 구하게 된다.

이렇게 구한 m을 두 자리씩 분리하여 permutation
$v = (10\ 3\ 8\ 7\ 4\ 1\ 9\ 6\ 2\ 5)$을 구하고 이에 의해 v의 inverse permutation v^{-1}를 계산한다. 따라서 v^{-1}에 의해 암호화된 문장 M으로부터 복호화된 문장 M'을 구할 수 있다.

예 3.6 $v = (10\ 3\ 8\ 7\ 4\ 1\ 9\ 6\ 2\ 5)$
일 때

$$v^{-1} = (6\ 9\ 2\ 5\ 10\ 8\ 4\ 3\ 7\ 1)$$

을 계산하고 이에 의해

M' = iramekr oaeminetvui eii^Qtrv !sy

은

M = korea maritime university !^Qive

로 복구된다. 복구화된 문장에서 특수문자 ^Q 이하의 문자는 버림으로써 완전한 문장을 얻어 낼 수 있다.

지금까지의 예에서는 무작위 소수 p와 q를 40비트의 수에서 여러 과정들을 소개 하였으나 실지로 높은 안전성을 유지하기 위해서는 보다 높은 비트수가 요구된다. 따라서 실제 프로그램에서는 p와 q를 1024비트를 가지는 무작위 소수로 생성하였고 permutation v 역시 문자들의 블록을 늘리기 위해서 $\{1, ,2 ,3, \ldots, 128\} \rightarrow \{1, 2, ,3, , \ldots, 128\}$인 무작위 permutation을 사용하였다.

4. Source

(1) 키생성

```
#include <NTL/ZZ.h>
#include <time.h>

#define BIT 1024

int main()
{
        ZZ p, q, n, a, a1, e, e1, d, x1, x2, y, y1, y2, c, f, t;

        /* 비밀키를 구하기 위한 값의 초기화 */
        x2 = 1;
        x1 = 0;
        y2 = 0;
        y1 = 1;

        t = time(NULL);
        SetSeed(t);
        p = RandomPrime_ZZ(BIT, 100);
        q = RandomPrime_ZZ(BIT, 100);

        n = p * q;
        a = (p-1) * (q-1);

        e = RandomBnd(a);

        for (; ;)
        {
                if (GCD(e, a) != 1)
```

```
            e++;
        else
            break;
}

/* 비밀키 d를 구한다. */
e1 = e;
a1 = a;
while (a1 > 0)
{
        c = e1 / a1;
        f = e1 - c * a1;
        d = x2 - c * x1;
        y = y2 - c * y1;

        e1 = a1;
        a1 = f;
        x2 = x1;
        x1 = d;
        y2 = y1;
        y1 = y;
}

d = x2;

if (d < 0)
        d = d + a;

if ((e*d)%a!=1)
{
        cout << "Error!" << "\n";
        exit(1);
}
```

```
        else
        {
                cout << "Public:" << "(" << n << "," << e << ")" << "\n";
                cout << "Private:" << d << "\n";
        }
}
```

(2) 암호화

```
# include <NTL/ZZ.h>
# include <time.h>

# define SIZE 128

ZZ v[SIZE], m;
long v2[SIZE], i, j;

void permutation();
void symencrypt();

main()
{
        ZZ n, e, c;

        permutation();

        /*  permutation의 정수화    */
        m = v[0];
        for(i = 0; i < SIZE-1; i++)
                m = (m * 1000) + v[i+1];

        cout << "m = " << m << "\n";
```

```
        cout << "public key :" << "\n";
        cout << "n = ";
        cin >> n;

        cout << "e = ";
        cin >> e;

        c = PowerMod(m, e, n);
        cout << "c = " << c << "\n";

        symencrypt();
}

/* random permutation의 생성 */
void permutation()
{
        ZZ t;
        long v1[SIZE], a, b, tmp;

        for (i = 0; i < SIZE; i++)
                v1[i] = -1;

        t = time(NULL);
        SetSeed(t);
        v2[0] = RandomBnd(SIZE+1);

        while (v2[0] < 100)
                v2[0] = RandomBnd(SIZE+1);

        a = v2[0];
        v1[a] = 0;

        for (i = 1; i < SIZE; i++)
```

```
{
        tmp = RandomBnd(SIZE+1);

        if (v1[tmp] == -1)
        {
                v2[i] = tmp;
                v1[tmp] = 0;
        }
        else
        {
                for(j=1; j < SIZE; j++)
                {
                        tmp = AddMod(tmp, 1, SIZE+1);
                        if( v1[tmp] == -1)
                        {
                                v2[i] = tmp;
                                v1[tmp] = 0;
                                break;
                        }
                }
        }
}

for(i=0; i < SIZE; i++)
{
        if(v2[i]==0)
        {
                v2[i]=SIZE;
                break;
        }
 }

for (i = 0; i < SIZE; i++)
```

```
            v[i] = to_ZZ(v2[i]);
}

/* permutation을 이용한 메시지의 암호화 */
void symencrypt()
{

        char v3[SIZE], string[SIZE];
        long pos, loop, end, a, b;

        FILE *fp, *fp1;

        if ((fp = fopen("direct.txt", "r")) == NULL)
        {
                cout << "Error opening file." << "\n";
                exit(1);
        }

        fseek(fp, 0, 2);
        end = ftell(fp);
        pos = ((end - 1)%SIZE);
        loop = ((end -1) / SIZE);
        rewind(fp);
        b = end - (loop * SIZE);

        if ((fp1 = fopen("letter.txt", "aw")) == NULL)
        {
                cout << "Error opening file." << "\n";
                exit(1);
        }

        while(1)
        {
```

```
                a = fread(v3, sizeof(char), SIZE, fp);
                if((a-1) == pos)
                        v3[b-1]=17; /* 끝부분은 특수문자로 처리 */

                if ((a != SIZE) && ( (a-1) != pos) )
                {
                        cout << "Reading was done. " << "\n";
                        fclose(fp);
                        fclose(fp1);
                        exit(1);
                }

                for (i = 0; i < SIZE; i++)
                        string[i] = v3[v2[i]-1];

                 if (fwrite(string, sizeof(char), SIZE, fp1) != SIZE)
                {
                        cout << "Error writing to file." << "\n";
                        exit(1);
                }
        }
}
```

(3) 복호화

```
#include <NTL/ZZ.h>

# define SIZE 128

ZZ key[SIZE], inv[SIZE], m;
long inv1[SIZE], i, j;
```

```
ZZ x = to_ZZ("1000");

void rsadecrypt();
void inverse();
void symdecrypt();

main()
{
        /* 정수 m을 permutation으로 바꾼다. */
        rsadecrypt();

                for (i = SIZE-1; i >= 0; i--)
                {
                        key[i] = m - ((m/x) * x);
                        m = m/x;
                }

        inverse();
        symdecrypt();
}

/* c로부터 m을 RSA를 통해 구한다. */
void rsadecrypt()
{
        ZZ c, d, n;

        cout << "ciphertext: " << "\n";
        cout << "c = ";
        cin >> c;

        cout << "private key : " << "\n";
        cout << "d = ";
        cin >>d;
```

```
        cout << "public key: " << "\n";
        cout << "n = ";
        cin >> n;

        m = PowerMod (c, d, n);

        cout << "m = " << m << "\n";
}

/* permutation의 inverse permutation을 구한다. */
void inverse()
{
        for (i = 0; i < SIZE; i++)
        {
                for (j = 0; j < SIZE; j++)
                {
                        if (key[j] == i+1)
                        {
                                inv[i] = j+1;
                                break;
                        }
                }
        }

}

/* inverse permutation을 이용해 메시지를 원래의 문장으로 복구한다. */
void symdecrypt()
{
        char v1[SIZE], string[SIZE];
        long pos, loop, end, a, b;
```

```
FILE *fp, *fp1;

if ((fp = fopen("letter.txt", "r")) == NULL)
{
        cout << "Error opening file." << "\n";
        exit(1);
}

fseek(fp, 0, 2);
end = ftell(fp);
pos = ((end - 1)%SIZE);
loop = ((end -1) / SIZE);
rewind(fp);
b = end - (loop * SIZE);

if ((fp1 = fopen("message.txt", "aw")) == NULL)
{
        cout << "Error opening file." << "\n";
        exit(1);
}
for (i = 0; i < SIZE; i++)
        inv1[i] = to_long(inv[i]);

while(1)
{
        a = fread(v1, sizeof(char), SIZE, fp);
        if ((a != SIZE) && ( (a-1) != pos) )
        {
                cout << "Reading was done. " << "\n";
                fclose(fp);
                fclose(fp1);
                exit(1);
        }
```

```
                for (i = 0; i < SIZE; i++)
                        string[i] = v1[inv1[i]-1];

                if (fwrite(string, sizeof(char), SIZE, fp1) != SIZE)
                {
                        cout << "Error writing to file." << "\n";
                        exit(1);
                }
        }
}
```

REFFERENCES

[1] H. Beker and F. Piper, Cipher Systems: The Protection of Communications, John Wiley & Sons, New York, 1982.

[2] D. M. Bressoud, Factorization and Primality Testing, Springer-Verlag, New York, 1989.

[3] "Communication theory of secrecy systems", Bell System Technical Journal, 27 (1948), 379 - 423, 623 - 656.

[4] "Cryptography and computer privacy", Scientific American, 228 (May 1973), 15 - 23.

[5] W. Diffie and M. E. Hellman, "Multiuser cryptographic techniques", Proceedings of AFIPS National Computer Conference, 109 - 112, 1976.

[6] W. Feller, An Introduction to Probability Theory and its Applications, John Wiley & Sons, New York, 3rd edition, 1968.

[7] X. Lai and J. L. Massey, "A proposal for a new block encryption standard", Advances in Cryptography-EUROCRYPT '90 (LNCS 473), 389 - 404, 1991.

[8] G. L. Miller, "Riemann's hypothesis and tests for primality", Journal of Computer and Systems, 13 (1976), 300 - 317.

[9] "New directions in cryptography", IEEE Transections on Information Theory, 22 (1976), 644 - 654.

[10] D. kahn, The Codebreakers, Macmillan Publishing Company, New York, 1967.

[11] **서광석**, **김용태**, **임종인**, **김창한**, **수론과 암호학**, **경문사**, 1998.

ABSTRACT

Cryptography has originated from its initial and limited use by the Egyptians about 4000 years ago. It has explosively developed through both world wars. The most striking developement came in 1976 when Diffie and Hellman introduced the revolutionary concept of public-key cryptography although their method was inpractical. In 1978, the first practical public-key cryptosystem was discovered by Rivest, Shamir, and Adleman, now refered to as RSA. RSA scheme is the most widely used system and its security is based on a hard mathematical problem, the intractability of factoring large integers.

In this paper, we introduce the general symmetric and public key cryptosystems and mainly, we attempt an effective realization of RSA cryptosystem using C++ language.

감사의 글

먼저 오늘의 제가 있기까지 물심양면으로 도와주신 지도교수님이신 배재국 교수님께 진심으로 감사의 말씀을 전합니다. 또한 좀 더 폭넓은 지식을 쌓을 수 있도록 힘써주신 금상호 교수님, 김재환 교수님, 김익성 교수님께도 진심어린 감사의 말씀을 전하고 싶습니다. 그리고 저의 일을 자신의 일처럼 생각하며 서로 도와가며 지냈던 해연 선배님 , 동훈 선배님, 비록 전공은 다르지만 많은 관심을 갖고 도와준 용수와 조교인 지영이를 비롯한 여러 동기들에게도 감사합니다. 또한 어려운 형편에도 제가 하고싶은 공부를 계속 할 수 있게 해주신 부모님을 비롯한 여러 가족들에게도 감사 드립니다. 끝으로 부족함이 많은 저에게 언제나 자상하신 모습으로 지도해 주신 배재국 교수님께 다시 한번 머리 숙여 감사드리며 그동안 배워왔던 수학적이고 합리적인 사고방식으로 앞으로의 삶을 살아가겠습니다.

2000년 1월 6일