論 文

공개키 암호체계를 위한 Modular 곱셈개선과 통신회로구현에 관한 연구

正會員 韓 善 景* 正會員 李 先 馥* 正會員 劉 泳 甲*

Implementation of Modular Multiplication and Communication Adaptor for Public Key Cryptosystem

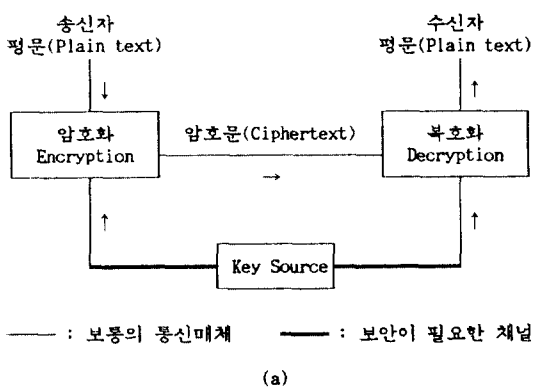
Seon Kyoung HAN* Sun Bok LEE* Young Gap YOU* *Regular Members*

要 約 공개키암호화에 대한 지수계산 방법의 개선과 serial 통신선에 실용적으로 적용하는 방법을 제시한다. RSA 형의 암호화 및 복호화 회로에 사용하기 위한 고속 modular 곱셈 알고리즘을 개선하였다. 기존의 고속 modular 곱셈 알고리즘에서 비교 과정에 사용되는 control bit값 선정을 개선하여 부분곱과 modular값의 비교과정에서 오류가 발생되지 않도록 하였다. 이 개선된 알고리즘은 C 언어를 사용하여 작성한 simulation program에 의한 simulation을 통하여 그 정상 동작을 확인하였다. 또한 computer간의 serial 통신선에서 입력되는 serial data를 sampling하여 이것을 RSA 방식으로 암호화하여 송신하게 되고 수신측에서는 이의 역순으로 처리하며, 이 sampling 및 암호화에 Z80 microprocessor를 중심으로 암호회로를 실제 제작하였다.

ABSTRACT An improved modular multiplication algorithm for RSA type public key cryptosystem and its application to a serial communication circuit are presented. Correction on a published fast modular multiplication algorithm is proposed and verified thru simulation. Cryptosystem for RS-232C communication protocol is designed and prototyped for low speed data exchange between computers. The system adopts the correct algoroithm and operates successfully using a small size key.

I. 암호화체계

정보통신 체제의 발달과 함께 정보가치의 유지를 위한 수단으로서 암호의 역할이 중요시 되고 있다[3,4,5,7,8]. 암호시스템은 메시지를 변역하는 두가지 형태의 방법을 가지고 있다. 그림 1.1에 보여진 바와 같이 암호 system은 평문(plain



(a)

*忠北大學校 情報通信工學科
Dept. Computer and Communication Eng.
Chungbuk National University
論文番號 : 91-61(接受1991. 3. 22)

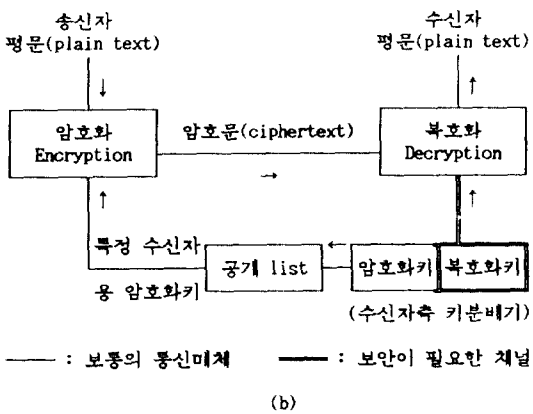


그림 1.1. (a) 관용 암호시스템 ; (b) 공개키 암호시스템.
Fig. 1.1. (a) Conventional cryptosystem ; (b) Public key cryptosystem.

text)이 송신자에 의하여 암호문(cipher text)으로 바뀌며, 이렇게 암호화된 내용은 통신 채널로 수신자에게 보내어진다. 통신 채널을 통하여 수신된 암호문은 수신자(decryptor)에 의하여 다시 평문으로 복호화된다. 송신자(encryptor)나 수신자(decryptor)는 사람이나 컴퓨터, 혹은 특별히 고안된 장치에 의해 수행되는 일종의 procedure이며, 이때 이들은 암호문 생성을 위하여 key distribution center로부터 key를 취하여 사용한다[2].

기존 암호화 방식의 표준인 DES(Data Encryption Standard)는 데이타를 보호하기 위한 수학적인 알고리즘이다. 미국 상무성 표준국(NBS : National Bureau of Standard)이 1973년 암호화 알고리즘을 공모하여, IBM사가 1975년 발표한 DES를 채택하였다. 그후 FIPS (Federal Information Processing Standard) PUB. 46 표준안(1977. 7. 15)으로 확정되었다[4,7]. ANSI (American National Standard Institute)는 표준으로 지정하였고, ABA(American Bankers Association)에서는 권고사항으로 결정하기에 이르렀다.

이 DES 방식의 중요한 문제는 어떻게 송신자나 수신자에게 key를 누설되지 않게 제공하는 방법을 강구하는 것이다. 암호문을 해독하거나 어떤 메시지를 암호화 하는데 사용되는 key는 우리가 사용할 수 있는 key들만을 모아 놓은 key distribution center에서 무작위로 선택되어 얻어 진다. 그러나 종래의 방법에는 암호화나 해독하는데 있어서 필요한 key가 같으므로 이에 대한 비밀유지가 가장 커다란 문제점으로 나타났었다. 이 DES 알고리즘을 시스템 레벨에서 보면 key를 만들고, 보관하고, 전송하는 등의 조작이 많다. 따라서 key가 누설될 위험이 높으므로, DES를 포함한 관용 암호시스템(conventional cryptosystem)에서는 key의 관리가 가장 중요하며, 최근의 암호 알고리즘이 key관리에 중점을 두는 것도 이때문이다.

그러나 공개키 암호시스템(public key cryptosystem) 은 이러한 key 분배의 문제점을 극복한 새로운 방식의 암호체계를 말하는 것이다. 공개키 암호시스템의 중요한 개념은 암호화 key와 복호화 key의 두가지 종류가 있으며 그 성질은 다음과 같다. 첫째, 서로 다른 복호화 key와 암호화 key가 존재한다. 둘째, 암호화 key와 복호화 key로 이루어지는 순서쌍은 컴퓨터에서 사용 가능하다. 세째, 암호화 key를 안다고 해서 복호화 key를 알 수 있는 것은 아니다. 그림 1.2에 보여진 바와 같이 한쌍의 key를 만들어서 그 중 암호화 key만을 A에게 보내주면 된다. 이때 key에 대한 비밀유지는 필요없으며, 그 key는 메시지를 암호화할 수만 있을뿐 해독에 사용할 수는 없다.

공개키 암호체계에서는 암호화 key에 대한 비밀유지가 필요없기 때문에 그것을 컴퓨터 네트워크 화일에 보관할 수 있다. 이렇게 하면 수신자에게 메시지를 보내고자 하는 사람이면 누구나 공개된 암호화 key를 이용하여 메시지를 암호화하여 보낼 수 있다. 수신자는 복호화 key를 다른 사람에게 알릴 필요가 없으므로 그 메시지의 비밀은 유지될 수 있으며 다만 복호화 key를 가진 사람만이 그 메시지를 해독할 수 있는 것이다.

공개키 암호시스템(public key cryptosystem)을 적용시킬 수 있는 실질적인 시스템의 구현 방법중 대표적인 방식인 RSA(Rivest-Shamir-

-Adleman) 암호 시스템에서는 key가 보통 150자리 이상이 되면 이 암호화 체계는 깨지지 않는다고 볼 수 있다. 암호화 key는 두개의 소수(prime number)의 합성으로 구성되는데, 이 소수는 암호조립자가 임의로 선택한다. 또한 복호화 key도 같은 두개의 소수와 공개된 공식을 사용하여 구할 수 있다. 따라서 누구든지 그 두개의 소수만 알면 복호화 key를 계산할 수 있으므로 그 숫자에 대해서만은 비밀이 유지되어야 한다. 그러나 key를 분석하여 그 두개의 소수를 얻어내는 방법은 무척 어려운 일이다.

이 연구에서 다루고 있는 것은 RSA 방식의 공개키 암호시스템의 구현 문제이다. 암호화 및 복호화 과정에서 핵심적인 역할을 하는 modular 곱셈의 정확성을 높이는 일이 수행되었다. 기존의 modular 곱셈 알고리즘에 대한 해석과 그 알고리즘이 갖고 있던 오류를 정정하는 작업이 제시되었다. 또한 RSA 방식의 구현으로서 RS-232C 통신회로의 개조를 통하여 실제적인 암호통신 실험을 완료하였다.

이 논문의 제 2장에서는 암호화와 복호화에 사용되는 modular 곱셈과 개선책이 제시되었다. 제 3장에서는 암호화 회로의 구성과 동작원리를 기술하였고, 제 4장에서는 결론을 맺었다.

Ⅱ. Modular 곱셈

공개키 암호화에서는 두 개의 소수를 이용하여 공개키와 비밀키를 합성하게 된다. 실제 2개의 소수를 선택할 때 몇가지 고려할 사항이 있다. 이것은 공개키와 modulo값 N으로 부터 비밀키를 분석하지 못하도록 하기 위한 방법으로서 두소수의 상관관계를 나타낸다. 첫째, 두 소수 p, q는 몇 digit 정도의 차이가 나도록 한다. 둘째, (p-1)과 (q-1)은 큰 소인수를 포함하도록 한다. 세째, (p-1)과 (q-1)의 최대공약수는 반드시 작아야 한다. 이상 3가지 조건들이 비밀키의 분석을 어렵게 한다. 일반대중에게 공개하는 공개키(w)와 비밀키(s)는 다음 식을 이용하여 선택한다. 여기서 p,q는 소수이다.

$$(s)\cdot(w) \bmod ((p-1)(q-1))=1$$
$$N=pq$$

암호화 및 복호화 과정의 식은 다음과 같다. 여기에서 X는 평문(plain text), Y는 암호문(cipher text)를 나타낸다.

$$Y=X^{w} \bmod N$$
$$X=Y^{s} \bmod N$$

문제가 되는 것은 암호화 작업에 소요되는 계산 시간이다. 키의 길이가 길어지게 되면, 이 키를 지수로 쓰는 지수계산이 많은 계산작업을 요구하게 되어서, 고속전송망에서의 실시간 처리가 어렵게 되는 것이다. Modular 곱셈은 지수계산을 고속으로 수행하기 위한 효과적인 방법으로 제시되었다[1,2]. 이것은 지수계산의 매 단계마다 modular 연산을 수행함으로써, 중간곱의 크기를 줄이도록 하였으며 따라서, 계산속도를 빠르게 하는 방법이다.

2.1 Carry Save Addition과 Modular 곱셈

RSA 암호방식 구현에 있어서 풀어야 할 핵심적인 요소는 암호화 속도를 높이는 것과 그 정확성을 유지하는 것이다. 이 연구에서는 기존의 암호화 알고리즘을 개선하여 정확한 결과가 얻어지도록 하였다. 먼저 속도 개선을 위한 기존의 이론을 소개하기로 한다. 암호화 과정에서도 특히 긴 시간이 요구되는 지수연산을 신속하게 수행하기 위하여 두 가지의 특별한 방법이 사용되었다[1]. 첫째는 carry전파에 의한 지연을 방지하기 위한 carry save addition의 효과적인 도입이며, 둘째는 중간곱의 크기를 제한시키기 위한 modular 곱셈 방식을 쓰는 것이다. 이 연구에서는 속도 개선에 사용되는 이 modular 곱셈에 있어서 기존 이론이 가지고 있는 오류를 정정하는데 촛점을 맞추었다.

Carry save addition은 각 자리의 숫자 표기에 있어서 redundant number system을 채택하고 있다. 이 숫자표기의 redundancy에 의하여 carry 전파가 불필요하게 된다. 대표적인 redundant

number format으로서 각 자리가 두 bit로 구성되어 한개는 sum bit 다른 한개는 carry bit로 사용하는 것이다. 이 carry save 개념이 Norris와 Simmons에 의하여 수정되었는데, 각 자리에 사용되는 이 2 bit를 하나는 t bit와 d bit로 재정의한 것이다[1]. 즉, si와 ci가 기존의 redundant number system에서 각각 i번째 자리의 sum bit와 carry bit라고 하자. 그러면 Norris와 Simmons의 경우 i번째 자리의 $t_i$bit와 $d_i$bit는 다음과 같이 정의된다[1]. 여기서 ⊕는 exclusive-OR 이다.

$$t_i = s_i \oplus c_i$$
$$d_i = s_i c_i$$

B register의 값을 t_i와 d_i로 구성된 register의 값에 더하여 i+1번째 자리의 sum bit와 carry bit를 다음과 같이 구한다. 여기서 b_i를 B register의 i번째 bit값이라고 하면 다음과 같다.

$$s_{i+1} = t_i \oplus d_i \oplus d_i$$
$$c_{i+1} = t_i d_i \cup t_i b_i \cup bidi$$

모든 i에 대하여 $d_{i+1} t_i = 0$이며 $d_0 = 0$이다.

이때 분명한 것은 i+1번째 자리의 계산을 위하여 i번째 자리의 값들만이 사용되었으며 i-1번째 이하의 값들은 전혀 고려하지 않아도 된다는 것이다. 즉 여러 bit에 걸치는 연속적인 carry전파가 억제되어 있다는 것이다. Norris와 Simmons가 기존의 si, ci에서 ti, di로 변형을 시도한 것은 곱셈에 수행해야 하는 덧셈에 걸리는 시간을 줄임으로써 곱셈의 속도를 높이고자 한 것이다. 이제 이 곱셈의 속도를 더욱 높이는 방법을 소개하고자 한다.

2.2 Modular 곱셈방법

RSA 암호화 과정에서 사용되는 modular 곱셈에는 두가지의 문제점이 있다. 평문을 숫자화하여 여기에 큰 지수를 써서 지수계산이 수행되고 나면 대단히 큰 자릿수의 숫자가 얻어지게 된다. 따라서 첫 문제점은 이 큰 숫자를 저장하는데 소요되는 기억용량이며, 둘째는 modular 연산에 소요되는 과도한 계산시간이다. 이 기억용량과 계산시간 부담을 줄이기 위하여 도입한 것이 지수계산 도중에 생성되는 중간곱에 대한 modular 연산이다. 중간 단계에서 숫자의 크기를 제속하여 줄여 감으로써 첫째, 요구되는 기억용량의 크기를 줄이고, 둘째, 비교적 작은 숫자에 대한 modular 연산이 큰 숫자에 대한 연산보다 빠르므로 결과적으로 계산 속도를 줄이는 것이다. 이제 이 modular 곱셈을 좀 더 자세히 서술하고자 한다.

먼저 A와 B가 각각 길이 n인 redundant number format의 register이며 a_i와 α_i쌍과 b_i와 β_i쌍이 각각 A와 B의 i번째 자리의 두 bit라고 하자. 그러면 A와 B에 저장된 두 수의 곱 D는 다음과 같이 계산된다.

$$D \leftarrow D + a_{n-1} \cdot 2^{n-1} B$$
$$D \leftarrow D + a_{n-2} \cdot 2^{n-2} B + \alpha_{n-1} \cdot 2^{n-1} B$$
$$\vdots$$
$$D \leftarrow D + a_1 \cdot 2B + \alpha_2 \cdot 2^2 B$$
$$D \leftarrow a_0 \cdot B + \alpha_1 \cdot 2B$$

여기서 이 redundant number의 곱셈이 기존의 carry save addition보다 유리한 점이 발견된다. 그것은 항상 $a_i \alpha_{i-1} = 0$이기 때문에, $a_i 2^i B$ 또는 $\alpha_{i-1} 2^{i-1} B$중의 하나가 0이 되어서 계산이 간단해진다는 것이다. 즉, 일반적인 carry save 덧셈의 경우 $a_i \alpha_{i-1} = 1$이 될 수 있지만, 이 redundant number system의 경우 $a_i \alpha_{i-1} = 0$이 되기 때문에, 각 단계에서 B의 shift와 한개의 addition만 하면 된다는 것이다.

이제 modular 연산을 수행하여, E≡D mod C가 얻어지도록 한다. 여기서 C는 modular값을 정하는 변수이다.

$$\text{If}(D \geq 2^{n-1} C) \text{ then } D \leftarrow D - 2^{n-1} C$$
$$\text{If}(D \geq 2^{n-2} C) \text{ then } D \leftarrow D - 2^{n-2} C$$

If(D≧2C) then D←D−2C
E←D

이 과정은 전형적인 나눗셈 algorithm이다. D를 dividend로 잡고, C의 2^{n-1}승한 값을 divisor로 잡아서 계속적으로 감산해 가는 방법이다. 여기서 D가 아주 크게 되면 곱셈의 경우에서처럼 긴 단계를 거쳐야 한다. 이 단계를 줄이는 것이 modular 곱셈의 속도개선에 큰 도움이 된다.

Modular 곱셈의 속도를 개선하고 중간곱의 저장에 필요한 기억용량을 줄이기 위하여, 위의 곱셈 algorithm과 나눗셈 algorthm을 결합하여 하나의 단일 algorithm으로 만든다[1]. 곱셈 algorithm 에서 덧셈 연산이 종료될 때마다 나눗셈 algorithm을 하기위한 뺄셈 연산을 수행하도록 하는 것이다. 이 나눗셈 연산 결합과정에서 문제가 되는 것은 dividend와 divisor를 비교하는 과정이다. 실제로 이 비교를 수행하기 위해서는 뺄셈 연산을 수행하여 부호의 변화를 관찰하여야 하는데, redundant number system에서는 이 과정이 상당히 복잡하고 적지않은 시간이 요구되기 때문에 속도 개선의 효과를 상실하게 될 우려가 있다.

2.3 Algorithm 개선

나누기 과정에서 dividend와 divisor의 비교과정을 단순하게 하는 것은 다음과 같은 현상을 기초로 하여 이루어진다. 앞서 설명한 절차에 대하여 곱셈연산을 얼마간 수행하고 나면, 중간곱의 결과에서 상위 몇 bit가 변하는 확률이 지극히 작아지게 된다. 따라서 dividend와 divisor의 상위 몇 bit만 비교하면 대부분의 경우 divisor 가 dividend보다 큰지 작은지 판단할 수 있다. 이 현상을 이용하여 비교 연산을 상위 4bit에 국한시킨 것이 Brickell이 사용한 방법이다[1]. 그러나 상위 4bit가 같은 경우에는 다음 몇 bit를 더 검사하여 대소판정을 실행하여야만 하는데, Brickll의 algorithm은 실제의 크기와는 상관없이 dividend가 작은 것으로 단정해 버리는 오류를 범하고 있는 것이다.

Brickell의 오류를 정정함에 있어서 이 논문에서 제시하는 algorithm은 상위 4bit가 같은 경우 다음 4bit씩을 순차적으로 비교하여 대소판정의 정밀성을 기하였다. 이 오류 발생을 다음 simulation 에 의하여 보이고자 한다. 먼저 Modular 11111과 multiplier 8888, multiplicand 7777을 선택해보자. Brickell의 잘못된 알고리즘이 그림 2.1a에, 그리고 이것을 정정한 algorithm을 사용하여 정확한 결과를 산출한 예를 그림 2.1b에

```
multiplier    =>> 8888
binary value  0000000000010001010111000
multiplicand  =>> 7777
binary value  0000000000001111001100001
modular       =>> 11111
binary value  0000000000010101101100111
              j value is 1
B*      <—    0000000000001111001100001
K_11    <—    0101001001100100000000000
sum2    <—    1001
t2      <—    0
K_10    <—    1010100100110010000000000
sum1    <—    1110
t1      <—    0
K*      <—    0000000000000000000000000
D       <—    1000001111010110111101110
A       <—    0000000000100000000000000
              j value is 2
              .
              .
D       <—    0101101000010001111011100
A       <—    0000000001000000000000000
              .
              .
              j value is 3
sum2    <—    1010
t2      <—    0
sum1    <—    1111
t1      <—    0
              .
              .
Result value  1011010010010000000000000
```

(a)

```
multiplier    ==>> 8888
binary value  000000000000100001010111000
multiplicand  ==>> 7777
binary value  000000000000011111001100001
modular       ==>> 11111
binary value  000000000000101011011001111
                 j value is 1
B*      <--   000000000000011111001100001
K_11    <--   010100100110010000000000000
sum2    <--   1001
t2      <--   0
K_10    <--   101010010011001000000000000
sum1    <--   1110
t1      <--   0
K*      <--   000000000000000000000000000
D       <--   100000111101011011101110
A       <--   000000000001000000000000000
              j value is 2
              .
              .
D       <--   010110100001000111101110 0
A       <--   000000000010000000000000000
              .
              .
              j value is 3
sum2    <--   1010
t2      <--   0
sum1    <--   0000
t1      <--   1
              .
              .
Result value  000000110111101000000000000
```

(b)

그림 2.1 (a) Brickell의 algorithm ; (b) 오류를 정정한 algorithm.

Fig. 2.1 (a) Brickell's algorithm ; (b) Corrected algorithm.

보였다. 수작업에 의한 결과는 445, 이진수로 000001101111101와 같다. 그림에서 j값은 연산 실행횟수를 표시한다. Brickell의 algorithm에 의한 3회째 sum1의 계산 과정을 살펴보면 2회 실행에서의 partial product D의 상위 4bit와 K_10의 상위 4bit의 크기가 같다.(서로 1의 보수 관계에 있다.) 이때 partial product D가 modulo 값보다 작은것으로 간주하게 되다. 그 결과로 control bit t_1의 값이 0가 되어, 다음 연산에서 뺄셈이 실행되지 않으며, 잘못된 결과값 1011 0100100100를 산출하게 된다. 이것은 십진수로 11556이며 옳은 값 445와 다르다. 그러나 오류를 정정한 algorithm의 경우에 있어서는 D와 K_10의 아위 4bit씩을 더 검사하여 D가 K_10보다 크다는 것을 밝혀 control bit t_1의 값을 1로 정하여 뺄셈이 수행되도록 하였다. 따라서 그 결과는 000001101111101이며 이것은 수작업에 의한 값 445와 일치한다.

```
Program FAST MULTIPLICATION;
  Do i=0 to 10                          /* Delayed carry register
     D ← D + A[i] × B;                     D의 초기화            */
  End.
  t1 ← 0, t2 ← 0;                       /* Control bit값의 초기화 */
  Do j = 1 to n                         /* Multiplication & modulo
     B* ← an-1B + αn2B,                    연산 */
     K* ← t2 2^11 K + t1 2^10 K,
     D ← 2(D + B* + K*),
     A ← 2A,
     j = j + 1;
     While (D ≥ 2^11 K) do              /* Partial dividend와
          D의 상위부터 4bit씩 + 2^11 K;    modular의 크기 비교   */
     If 2^11 K에 대하여 overflow        /* 비교 결과에 따른
        then t2 ← 1,                       control bit t2 설정   */
        else t2 ← 0;
     While (D ≥ 2^10 K) do
          D의 상위부터 4bit씩 + 2^10 K;
     If 2^10 K에 대하여
          overflow이고 t2 = 0           /* Control bit t1 설정   */
        then t1 ← 1,
        else t1 ← 0;
  End.
Program End.
```

그림 2.2. 고속 modular 곱셈의 알고리즘.

Fig. 2.2. Pseudo code for a fast modular multiplication algorithm.

정확성이 보정되도록 고쳐진 modular 곱셈 algorithm에 대한 pseudo code program이 그림 2.2에 보여졌다. 이 algorithm이 모든 입력 data에 대하여 정확한 암호를 발생시키는 것은 위의 계산과정을 이용하여 쉽게 증명할 수 있다. 여기서 modular 곱셈은 먼저 상위 11 bit의 multiplier에 대하여 add와 shift를 사용한 곱셈을 수행한 후 C의 배수 뺄셈을 한다. 이때 register D에 n+11의 delayed carry를 저장한다. 여기서 K는 modular C의 2의 보수이다. 그리고 t_1과 t_2는 control bit로 사용되며, 다음과 같은 방법으로 결정된다. D의 상위 4bit가 2^{11}C의 상위 4bit보다 클 때 t_2=1이 되고, t_2=0이고 D의 상위 4bit가 2^{10}C의 상위 3bit보다 클때 t_1=1로 set된다.

2.4 Software Description

기존의 modular 곱셈 알고리즘에서 발견되었던 오류를 정정하고 software simulation을 통하여, 그 정확성을 확인하였다. 이 simulation은 main 함수(function)를 포함하여 전체 8개의 함수로 구성되어 있다. 우선 main 함수에서는 다른 7개의 함수를 호출하여 algorithm의 구현을 총괄하게 된다. 모든 register와 그 register의 상태를 화면상에 표시하도록 하여 algorithm을 확인 할 수 있도록 하였으며, 각 단계마다 변화하는 상태를 볼 수 있다. 나머지 7개의 함수는 input(), dtob(), multiplication(), add(), shift(), twos(), output()이다.

우선 이 simulation program의 input() 함수는 연산에 필요한 데이타, 즉 multiplicand, multiplier, modular를 keyboard를 통하여 받아 들인다. 십진값으로 입력된 데이타를 이진수로 변환하여 화면상에 표시하여 입력 데이타를 확인할 수 있도록 하였다. 함수 dtob()는 십진수로 입력된 데이타를 이진값으로 변환시키는 기능을 가진다. 이 함수는 input() 함수에서 호출한다. 또한 multiplication() 함수는 그림 2.2의 pseudo code 부분에서 제일 처음 줄에 대한 기능을 가진다. Multiplicand의 상위 11 bit와 multiplier의 곱셈을 행하는 부분이다. 그 결과는 n+11의 길이를 갖는 register에 저장되며, 마지막에 결과를 포함하는 delayed carry register D의 초기화를 하게 된다. Add와 shift 연산을 통하여 구현된다.

이 dd()는 두개의 이진수의 합을 구하는 함수로써 모든 덧셈 연산에 사용되며, 앞서 설명한 함수 multiplication()에서 곱셈 연산을 수행하기 위하여 호출된다. Partial dividend와 modular의 크기를 비교하기 위하여 modular C의 2의 보수 K를 shift시킨 것과 D의 4bit씩 덧셈을 수행하는 부분에서도 호출된다. 이때 overflow의 여부를 판정하여 후에 control bit값을 결정하는 역할을 하게 된다. twos() 함수는 modular C의 2의 보수 즉, K를 산출하는 부분이다. shift() 함수는 partial product와 modular의 크기 비교를 위하여 덧셈에 사용되는 10 혹은 11자리 shift된 C의 2의 보수를 산출하게 된다. 마지막으로 output()함수는 modular 연산이 완전히 실행된 후 최종 결과값을 화면상에 표시하게 된다.

program 전반에 사용되는 register는 데이타 형지정문(typedef)으로 배열형을 지정하여 구현하였다. 세개의 연산수에 할당된 register의 bit 수는 program에서 작성된 것과는 관계없이 연산수의 크기에 따라 적당한 길이로 설계할 수 있다. 본 program에서는 14 bit로 각 연산수에 할당하였으며, 이 bit수는 쉽게 바꿀 수 있도록 설계하였고, 계산하고자 하는 값의 크기에 대한 제한은 없다고 할 수 있다.

입력 데이타는 multiplicand, multiplier, modular 세개의 십진수값이다. Keyboard를 통하여 입력하고 이진수로 변환하여 입력값을 확인할 수 있다. 각 연산수(operand)는 14 bit를 할당하였으며 필요에 따라 수정할 수 있다. 출력 형식은 임의의 key를 눌러 다음의 연산을 한 step씩 표시하도록 하였다. 전후 연산수를 비교하여 algorithm을 확인하면서 쉽게 algorithm을 이해할 수 있다. 각 register는 n+11 bit, 이 program상에서는 25bit를 화면상에 표시하고, 최종적인 결과값은 가장 왼쪽에서부터 n bit를 취하게 된다. n+11 bit는 multiplier과 multiplicand의 선행곱에 의한 delayed carry register에 기인한다.

기존의 algorithm에서 control bit값을 설정하기 위하여 partial product와 modular의 상위 4 bit를 비교하여 크기가 같은 경우, 이후의 bit를 계속 더 비교하여 오류 발생을 제거시키는 algorithm에 대하여 작성하였다. 기존의 modular 곱셈은 partial product의 크기가 modular와 같은 경우(실행 결과의 화면 display상에서는 서로 1의 보수 관계), 실제와는 무관하게 partial product가 작은것으로 간주하여 연산하므로 뺄셈(결과적으로 modulo 연산)이 덜 실행되어 잘못된 결과를 산출하게 된다. 따라서 기존의 algorithm은 오류가 있으며, 이 오류는 512 bit 연산에서 적지 않게 발생하며 사실상 사용이 불가능하다. 두 수의 전체 bit를 비교하면 이 오류를 방지할 수 있다. 전체 비교를 보다 효율적으로 하기 위하여 상위 bit부터 서로 값이 다를 때까지 4 bit씩 잘라서 비교하도록 하였다.

Ⅲ. 암호화회로의 구현

Computer간의 통신에서 많이 사용되는 RS-232C 비동기통신 adapter에 앞장에서 소개한 공개키 암호화 알고리즘을 Z-80 micro processor를 이용하여 구현하는 방법을 소개한다.

3.1 암호통신 System

정보 교환을 행하는 송신자 A와 수신자 B 각각이 computer에 연결된 RS 232C를 이용하는 경우에 사용되는 암호체계 구현을 시도하였다. 먼저 앞장에서 설명한 암호화 이론을 이용한 회로를 구현하여 RS-232C에 부착시키고, assembly language를 사용한 통신 제어 program을 computer에 입력시킨 후 실행시키는 것이다. 이렇게 하면 송신자 A에서 송신되는 정보 내용은 새로 부착된 암호화 회로에 의해 암호로 바뀌어 수신자 B의 computer에 전송된후 해석되는데, key가 없는 terminal에는 암호문만이 출력되어 해독이 불가능하므로 비밀이 유지된다.

송신자 A에서 나오는 serial output의 가공과정에서(여기서는 sampling이라고 부름) RS-232C의 출력 신호를 암호화하는 것이다. 이 방법은 data가 EIA규격의 +12V와 −12V로 승압되기 이전의 serial data을, 특별한 회로를 써서 가로채어, SIPO(Serial Input Parallel Out) register를 써서 parallel data로 변환하여 암호화 한후, 다시 serial data로 변화하여 전송한다.

각 비트는 baud rate에 의해서 timing이 결정되며 baudout pin은 baud rate ×16 pulse가 나오게 되어 각 bit를 sampling하는데 이용된다. Serial bit 중 start bit가 나오는 순간부터 8개의 pulse를 세어 bit의 중앙에서 data를 sampling하고, 다시 16개의 pulse를 센 후 계속해서 sampling을 한다. Sampling회로의 block diagram은 그림 3.2와 같다.

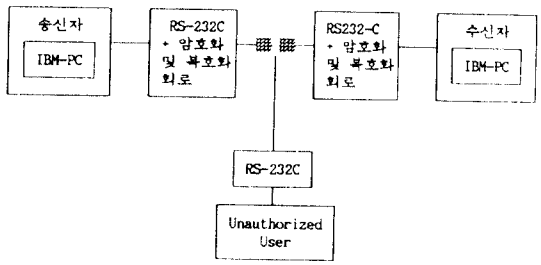


그림 3.1 암호통신 system 개략도.
Fig. 3.1 General cryptosystem configuration.

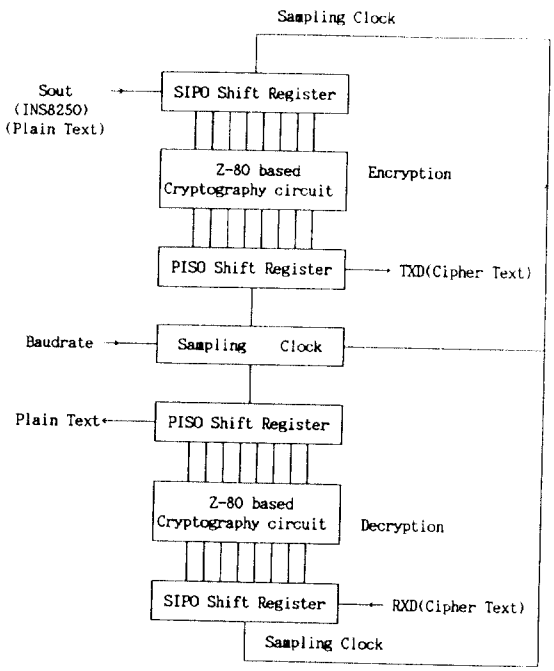


그림 3.2. 암호회로 block diagram.
Fig. 3.2. Block diagram of cryptography circuit.

3.2 암호 system의 구성

암호회로 설계 목표는 Z-80 microprocessor interface circuit을 이용하여, software based 공개키 암호화를 실현하는 것이다. 암호화를 하는데 있어서 key값의 자릿수는 1 byte로 하였으며 전송되는 data의 자릿수도 1 byte로 하였다. 이것은 암호체계의 보완성을 위해 수 백 bit 정도의 key를 사용하는 경우 특별한 집적회로 제작이 요구되지만[6], 이런 IC의 획득이나

제작이 어렵기 때문에, system의 정상동작을 확인하기 위하여 최소의 key 길이를 선택하여 제반 알고리즘의 정확성과 동작의 성능을 검토하였다. 이 연구에서 두 소수는 11과 23을 선택하였다.

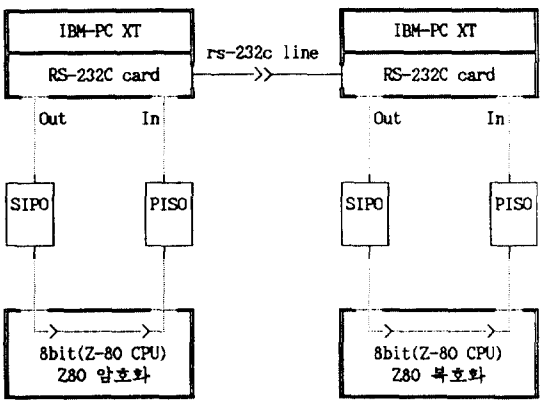


그림 3.3. Z-80을 기초로 하는 암호화 회로의 개략도.
Fig. 3.3. System configuration of a Z-80 based cryptosystem.

암호화 회로에 대한 전체 시스템의 block diagram 은 그림 3.3에 나타내었다. 하드웨어는 sampling circuit과 암호화 회로의 두 부분으로 나뉜다. Sampling circuit은 D flip-flop을 쓰는 delay based control circuit이다. 암호화 회로는 Z-80 microprocessor를 장착한 one-board micro computer를 사용하였다.

Sampling회로에서 사용된 port와 신호는 다음과 같다. RESET은 sampling circuit의 초기화를 통하여, system을 USART에서 출력되는 start bit감지 상태로 전이시킨다. END는 RESET와 같은 기능을 하게 되는데, RESET과 다른점은 control flow의 최종단으로부터 오는 신호이며, 이로써 system은 start bit의 재입력을 받을 수 있도록 한다. CLEAR COUNTER는 start bit의 감지를 하면 sampling counter를 0으로 clear 시킨다. LOOPO는 RESET or END signal에 의해서 활성화되며, USART의 출력핀으로부터 start bit 입력이 있을 때까지 계속해서 looping을 하게 된다.

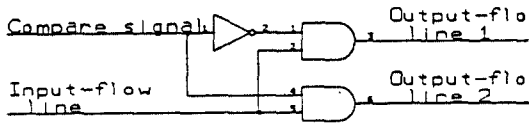


그림 3.4. 비교기.
Fig. 3.4. Compare unit.

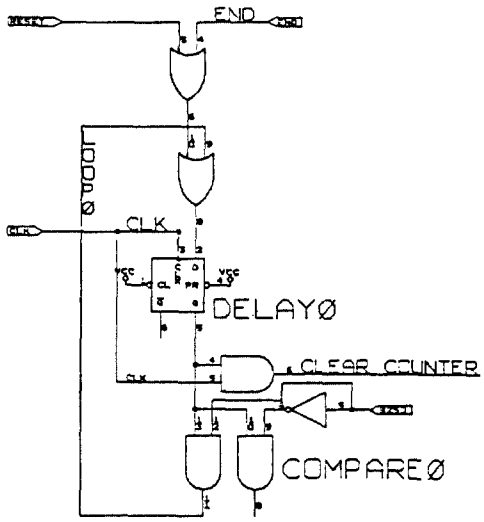


그림 3.5. Start bit 감지 회로.
Fig. 3.5. Start bit sensing circuit.

Sampling circuit은 counter와 shift register로 구성되었다. counter의 주기능은 sampling pulse와 8 bit data를 count하는 역할을 하게된다. Shift register는 그림 3.6에 보여진 SIPO (Serial Input Parallel Output)와, PISO(Parallel Input Serial Output) register로써 sampling된 data의 보관과 암호화 또는 복호화된 data를 일시적으로 보관하는 역할을 한다. 이 sampling circuit의 기능은 RS-232C의 USART chip에서 출력되는 serial data 출력을 baud rate에 동기시켜 8bit data을 sampling한다. 그때 sampling된 8bit data는 SIPO register에 저장된다. sampling 되어 저장된 data는 Z-80에 의해서 암호화 또는 복호화 되어 PISO에 저장된다. PISO에

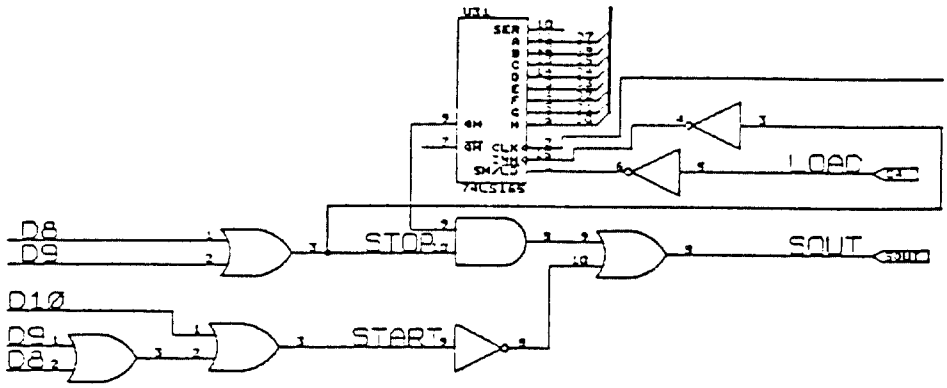


그림 3.6. Parallel Input Serial Out(PISO) 회로
Fig. 3.6. Parallel Input Serial Out(PISO) circuit.

저장된 data는 RS-232C 규격에 맞게 다시 serial로 변화되어 통신선의 baud rate에 맞도록 출력된다.

PISO의 START & STOP signal은 default가 low로써 SOUT 출력은 high를 유지하게 되는데, 이것은 RS-232C의 원리를 따라 동작한다. Data 출력이 없을때 SOUT 출력은 항상 high를 유지하게 되며, START signal이 high가 됨으로써 SOUT 출력이 low로 변한다. START & STOP bit 모두 high가 되면 data line상의 signal이 SOUT으로 출력이 되고 이때 8 bit data가 serial로 baud rate에 맞게 출력된다. Sampling circuit으로 부터 1 byte data를 받아 암호화 또는 복호화 연산을 수행한 후 다시 sampling circuit으로 되돌려 보낸다. 암호화 및 복호화에 Z-80이 사용되었다.

Z-80에 입력되는 암호화 program은 Z 80 assembly language로써 작성 되었다. 이 연구에서 두 소수는 11과 23를 선택하였으며 modular 값 N은 두 소수의 곱인 253을 선택하였다. 이 실행 program은 1-byte 단위의 암호화 및 복호화 연산을 하게 된다. 이 software에서는 다음 식과 같은 계산을 수행하게 된다.

$$Y^{147} \bmod 253 = (Y^1 \times Y^2 \times Y^{16} \times Y^{128}) \bmod 253$$

Cipher text인 Y를 128승까지 제곱승을 계속 진행한 후 그중에서 1, 2, 16, 128승된 data만을 추출해서 곱한뒤 mod 253을 취했다. 또한 매번 곱셈이 행해질 때 마다 mod 253을 함으로써 곱셈의 결과가 8 bit를 초과하지 않도록 하였다.

Z 80 암호화 회로에서 행해지는 암호화/복호화 작업은 data 전송시간의 지연을 일으킬 수 있다. 본 연구에서는 1byte의 key를 사용하여 1byte 단위의 data를 암호화하게 되는데, 실제 전송시간에 있어서의 초기 지연시간은 약 4.17ms정도였다. 이러한 지연시간 산출은 9600bps의 전송속도를 갖는 serial 전송라인에서 data sampling 시간에 의해 발생된다. 송신자측 전송시 첫 byte의 암호화 회로 통과시간과 수신자측에서의 첫 byte의 복호화회로 통과시간만이 지연시간에 영향을 주는데, 이는 암호화 및 복호화를 위한 작업으로써 serial data가 parallel data로의 변화과정에서 발생되는데 실제 암호화/복호화 연산시간은 영향을 주지 않는다. 첫 1byte가 암호화회로를 통과할 경우 그림 3.2나 그림 3.3에서 볼 수 있듯이 SIPO register를 통과한 뒤 Z-80 암호화회로에 의해 암호화가 수행되는 동안 두번째 1byte가 SIPO에 저장되는데 두번째 1byte의 저장이 완료되기 전에 암호화연산이 끝나게 되어 PISO에 보내져서 세번째 1byte가

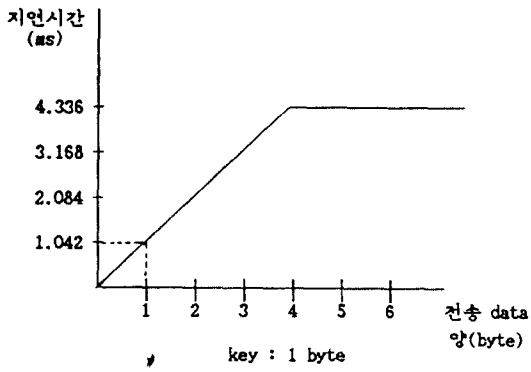


그림 3.7. 암호화 지연시간.
Fig. 3.7. Delay for cyphering opration.

SIPO에 저장되는 시점과 동시에 첫번째 byte가 PISO에 의해 출력된다. 즉, 9600bps에서의 2byte에 해당하는 전송지연시간을 갖게된다. 그림 3.7에서 보여진 것과 같이 수신자측도 마찬가지이다.

결국 초기 data의 sampling 시간만큼의 전송지연시간만이 존재함으로써, 암호화 key의 자리수가 증가하게 되면 그 key의 자리수와 같은 data가 암호화에 사용된다. 결국 data의 sampling 시간이 증가하며, 암호화 연산의 시간도 증가한다. 암호화 연산을 data의 sampling시간 동안에 수행하게 되면 전송지연의 누적현상을 없앨 수 있다. 여기서 사용한 Z-80 암호화회로는 4MHz의 clock을 사용하여 암호화시 0.11ms와 복호화시 0.98ms의 시간이 소요된다. 보다 높은 보안성을 위하여 연산전용 chip을 사용할 수 있다면, 수백자리의 key를 갖는 암호화도 전송지연시간의 부담이 없이 이루어질 수 있다.

Ⅳ. 결 론

공개키 암호화의 원리를 이용한 PC에서의 암호화 방법에 대한 연구 결과를 소개하였다. RSA방식의 공개키 암호화 회로설계에 있어서 지수함수의 계산 속도개선을 위하여는 modular 곱셈기를 활용하는 것이 일반적인 해결책이다. Brickell 등이 제안했던 modular 곱셈 알고리즘은 계산시간 단축에만 촛점을 맞추었기 때문에 계산결과의 정확성이 결여되어 있다. 이 Brickell의 알고리즘은 얼마간의 계산속도의 비용으로써 정확한 계산결과를 얻도록 개선하고, computer simulation을 통하여 그 정확성을 검증하였다.

Z-80 microprocessor를 이용하여 암호화를 구현하였다. 이 회로들은 RS232C serial port의 최종 단계에 연결되도록 설계함으로써 다른 기종의 computer에 이식할 수 있는 장점이 있다. 따라서 이 결과는 현재 저속 통신망의 비 실시간 응용이 가능한 것이며, 정보전송에서 정보가치 유지 수단으로 직접 사용될 수 있다.

여기서의 암호화는 1 byte를 기준으로 행해졌으나, 보다 빠른 통신을 위하여 곱셈전용 chip을 이용하여 암호화 속도를 증대시킴과 동시에 key의 byte수를 증가시킬 수 있다. Z-80 microprocessor 에서 실행된 assembly program을 곱셈용 chip으로 대치하고, 암호회로는 주문형 IC에 의해 간략화할 수 있다[6].

알 림

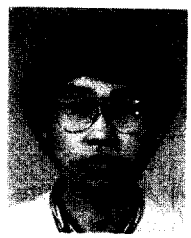
이 연구는 문교부 학술연구조성비에 의한 연구과제 ISRC90-E-DE-C002의 일부로 수행되었읍니다.

참 고 문 헌

1. E.F.Brickell, "A fast modular multiplication algorithm with application to two key crytography", *Proc. CRYPTO-82*, Santa Barbara, pp. 51-60, Aug. 1982.
2. H.C.A. van Tilborg, *An Introduction to Cryptology*, Kluwer Academic Publishers, Norwell, Massachusetts, 1988.
3. R.L.Rivest, A.Shamir and L.Adleman, "A method for obtaining digital signatures and public key cryptosystem", *ACM Comm*, vol. 21, no. 2, pp. 120-126, Feb. 1978.

4. U.S.Dept. of Commerce, National Bureau of Standards, *Data Encryption Standard*, Federal Information Processing Standard, Pub. 46, 1977.
5. W.Diffie and M.Hellman, "New direction in cryptography", *IEEE Trans. Info. Theory*, vol. IT-22, no. 6, pp. 644-659 Nov. 1976.
6. 유영갑등, Fault Tolerant Crytography용 IC개발, 서울대학교 반도체공동연구소 보고서, ISRC 90-E-DE-C002, 1990.
7. 임재홍, 변옥환, "정보통신 시스템에서의 시큐리티", 월간 전자과학, 제 30권 통권 344호, 134-156쪽, 1988년 1월호.
8. 호리시 아츠오, 컴퓨터 범죄와 암호화 대책, 한국전자통신연구소, 1990.

韓 善 景(Seon Kyoung HAN) 正會員
1987年 3月～1991年 2月 : 忠北大學校 情報通信工學科
1991年 3月～現在 : 忠北大學校 大學院 情報通信工學科 碩士課程



李 先 馥(Sun Bok LEE) 正會員
1987年 3月～1991年 2月 : 忠北大學校 情報通信工學科
1991年 3月～現在 : 忠北大學校 大學院 情報通信工學科 碩士課程

劉 泳 甲(Young Gap YOU) 正會員
1969年～1975年 : 西江大學校 電子工學科 (工學士)
1979年～1981年 : 美國 The University of Michigan, 電氣電算學科 (工學碩士)
1981年～1986年 : 美國 The University of Michigan, 電氣電算學科 (工學博士)
1975年～1979年 : 國防科學研究所 研究員
1986年～1988年 : 金星半導體(株) 責任研究員
1988年～現在 : 忠北大學校 情報通信工學科 敎授