해쉬함수를 이용한 전자서명 알고리즘분석 및 구현

유희경*, 성 경**

요약

본 논문에서 제안하는 A_SHA1해쉬 함수는 메시지의 무결성을 검증하기 위해 사용하였으며, 기존의 SHA1해쉬 함수와 달리 메시지를 압축하기 전에 역방향 저장과 순환 시프트연산, XOR 연산 과정을 추가하여 해쉬값을 계산함으로써 역변환 공격에 좀더 보안을 강화시켰다. A_RSA 서명 알고리즘은 메시지 인증과 신원확인, 부인방지를 위해 사용하였으며, 기존 RSA 서명 알고리즘의 최종 서명문에 순환 시프트 연산과 XOR 연산을 추가하여 보안을 향상시켰다. RSA 암호 알고리즘은 기밀성을 부여 하기 위해 사용하였으며, 전자서명 검증과정에서 필요한 원 메시지를 보호하여 전송하기 위해 암호화하여 구현하였다.

Analysis and implementation of Digital Signature Algorithm using Hash function

Hee-Kyung Yoo*, Kyung Sung**

ABSTRACT

The A_SHAI hash function I propose in this paper is used to verify the integrity of messages. This hash function, unlike the existing SHAI hash function, enables us to tighten further security against possible reverse attacks by calculating a hash value to which we adds a backward storage, a circular shift operation and ax XOR operation prior to the message digital sign. The A_RSA signature algorithm is used for message certification, authentication, and non-repudiation, which is also intended to tighten further security by adding a circular shift operation and an XOR operation to the final digital signature of the established RSA signature algorithm. RSA encryption algorithm is used to secure confidentiality in encrypted form for a protective transmission of the orginal message necessary for digital signature and verification.

Key Words : Hash function, Digital signature, Algorithm, A_SHA1, RSA

* 강원대학교 컴퓨터공학과(✉hkyoo@kangwon.ac.kr)

** 목원대학교 컴퓨터교육과

· 제1저자(First Author) : 유희경 · 교신저자(Correspondent Author) : 성 경

· 접수일(2011년 4월 8일), 수정일(1차 : 2011년 5월 9일), 게재확정일(2011년 5월 12일)

I. 서 론

네트워크를 통하여 물품구입을 하기 위해서는 사용자의 카드번호 및 신상 정보를 제시해야 하며 이것은 곧 안전하지 못한 통신망에 개인의 중요정보가 노출되는 위험이 있다. 또 쇼핑몰을 개설하고 있는 상점이 정당한 거래를 수행하고 있는 상점인지 또는 상점을 대상으로 물품을 구입하고자 접촉해 오는 사용자가 정당한 사용자인지에 대한 확인이 쉽지 않다[10]. 이러한 위험을 해결할 수 있는 방법이 암호 알고리즘과 전자서명 알고리즘이다. 먼저 위의 전자상거래 행위에서 사용자의 카드 번호 및 신상정보와 같은 민감한 정보는 암호 알고리즘을 통하여 암호화 하여 보호 할 수 있고, 쇼핑몰과 사용자 사이의 인증문제는 전자서명 알고리즘을 통하여 해결할 수 있다.

본 연구에서는 신용카드 번호와 같은 중요한 정보를 보호하기 위하여 RSA 암호 알고리즘을 사용하여 구현하였고, 인증문제는 A_RSA전자 서명 알고리즘을 이용하여 구현하였다. 또한, 송.수신된 메시지의 무결성 체크를 위해서 A_SHA1해쉬 함수를 사용하였다.

II. 관련연구

2.1 해쉬 함수(Hash Function)

SHA1은 SHA의 기술과 보안을 향상시킨 것으로 160비트의 메시지 다이제스트를 생성하여 128비트를 생성하는 MD5보다 약간 느리지만, MD5가 갖고 있는 전역 충돌과 역변환 공격에 대해 안전하다. 또한 SHA1은 메시지 다이제스트 값에서 원본 메시지를 찾거나, 서로 다른 두 개의 메시지로부터 똑같은 메시지 다이제스트를 만들어 내는 것이 연산 상 불가능하기 때문에 안전하다[1, 2].

해쉬 함수를 이용하여 메시지 무결성을 검증하는 방법을 살펴보면 그림 1과 같다.

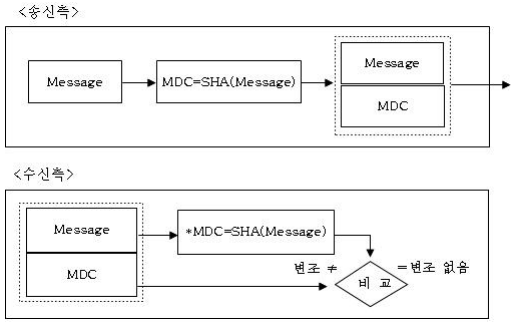

그림 1. 해쉬함수에 의한 메시지 무결성 검증 개략도

Fig. 1 Hash function, message integrity verification by the schematic

전자 서명 과정에 해쉬함수를 사용하는 또 다른 이유는 서명문의 길이를 줄이기 위함이다. 서명 알고리즘인 DSS(Digital Signature Standard)의 경우 160비트의 입력 메시지로부터 320비트의 서명문을 생성한다. 메시지의 길이가 길어질 경우, 이 메시지를 160비트 블록 단위로 나누어 서명을 한다면, 서명 후의 길이는 블록 당 원문의 두 배가 되어 전송시간이나, 메모리 낭비를 초래한다. 따라서 임의 길이의 입력 메시지를 고정된 길이(128비트, 160비트)의 출력 값으로 압축시킨 후, 이 압축된 해쉬 값에 대해 서명을 한다면 원문의 길이에 상관없이 고정된 길이의 서명문을 얻을 수 있게 된다[10].

2.2 전자서명(Digital Signature)

비밀키는 사용자 자신만이 알고 있는 키를 말하며 송신자는 이 키를 이용하여 문서에 서명을 수행하게 된다. 공개키는 비밀키에 대응되는 키로서 문서를 수신할 상대방이 공개키를 이용하여 서명된 문서를 검증하게 된다[4, 12]. 공개키를 이용한 전자서명 알고리

즘은 소인수 분해를 이용한 방법으로서 RSA, Rabin알고리즘이 있고, 이산대수를 이용한 방법으로 DSA, ELGamal, Schnor, Nyberg-Rueppel 서명 기법들이 있고, 타원곡선을 이용한 이산대수 방법의 대표인 ECDSA(Elliptic Curve DSA)서명 알고리즘 등이 있다. 전자 서명 알고리즘은 안전성, 신뢰성 보장을 위해 기본적인 요구사항들을 만족해야만 한다[7, 11]. 공개키 방식을 이용한 전자 서명문의 생성과 검증 절차를 살펴보면 그림 2와 같다.

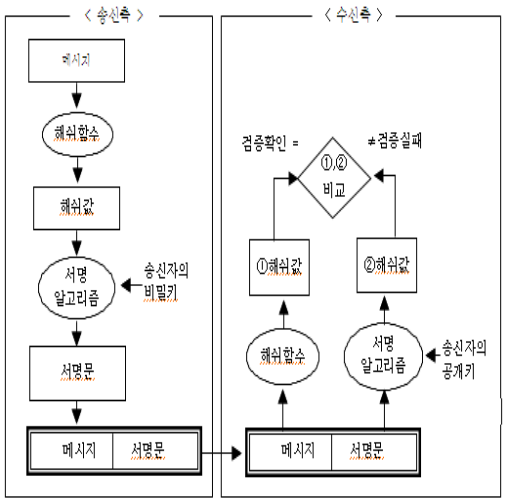

그림 2. 전자서명 생성 및 검증 과정
Fig. 2 Digital signature creation and verification process

송신된 원래의 메시지가 전송 도중에 제 3자에 의해 노출되어도 상관이 없는 것이라면 그대로 평문으로 보내고, 도청되거나 비밀로 간직해야할 것이라면 암호화 하여 전송되어야 한다. 그렇기 때문에 암호화. 복호화 알고리즘이 필요하다.

2.3 공개키 알고리즘(Public Key Algorithem)

암호 알고리즘은 크게 비밀키 알고리즘과 공개키 알고리즘으로 구분할 수 있다. 비밀키 알고리즘은 암호화 키와 복호화 키가 같이 사용 되는 것으로써, 송신자가 수신자에게 평문을 암호화하여 메시지를 보낼 때 사용한 키와 수신자가 암호문을 평문으로 복호화 하는데 사용되는 키가 동일하다는 것을 의미한다.

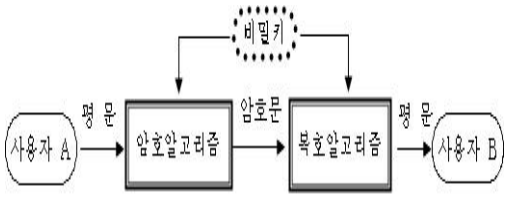

그림 3. 비밀키 암호 방식
Fig. 3 Secret-key cryptography

공개키 알고리즘은 송신자와 수신자가 암호화할 때 사용하는 키와 복호화할 때 사용하는 키가 서로 다르다. 즉, 송신자는 수신자의 공개키(public key)로 평문을 암호화해서 메시지를 보내면, 수신자는 자신의 비밀키(private key)로 암호문을 복호화 한다[6, 14].

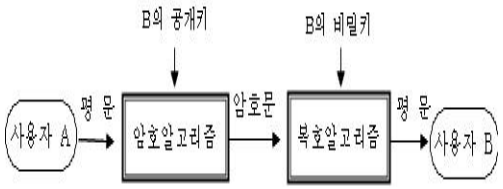

그림 4. 공개키 암호 방식
Fig. 4 Public key cryptography

공개키 시스템은 비밀키 시스템에서 필요로 하는 안전한 키 분배가 필요 없다[9].

인증기관은 공개키가 어떤 개인의 소유임을 인증하고 그것을 등록하며 공개키에 대한 증명서(Certification)를 발행하고, 갱신하고, 폐기하는 등 모든 공개키에 관한 관리를 중립적인 입장에서 관리, 감독하는 기능을 한다[8, 10]. 공개키 시스템에서의 보안 통신을 하는 과정을 살펴보면 [그림 2-5]와 같다.

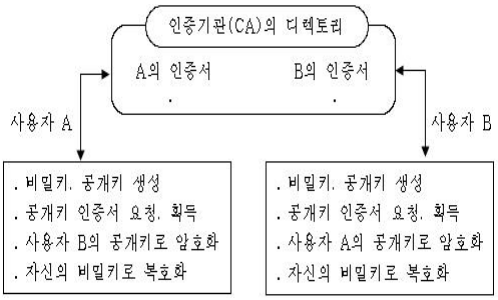

그림 5. 공개키 시스템의 보안통신 과정
Fig. 5 Process of public key systems secure communication

III. A_SHA1해쉬 함수와 A_RSA전자서명 알고리즘

3.1 알고리즘 전체 구성

1) 공개키와 인증서

통신을 시작할 때 서로의 프로토콜 버전과신분을 확인하고, 이후 공유 비밀키, 암호 알고리즘, 서명 알고리즘, 해쉬함수 등의 서로 공유할 비밀 정보를 생성한다[5]. 이러한 역할을 키 교환(Key Exchange) 프로토롤이 수행한다. 이 프로토콜은 합의된 정보를 보안 통신 시작전에 생성하고 유지, 소멸 등의 키 관리 작업을 자동으로 수행하기 위해 대부분 공개키 암호 기법을 사용한다. 공개키 암호 기법의 사용자는 먼저 자신의 공개키와 비밀키 쌍을 생성하고, 공개키에 대한 인증서를 인증기관(CA: Certificate Authority)을 통해 요청하고 획득한다.

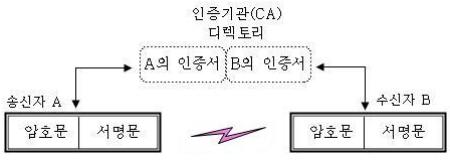

그림 6. 공개키와 인증서
Fig. 6 Public key and certificate

2) 알고리즘 구성과 기능

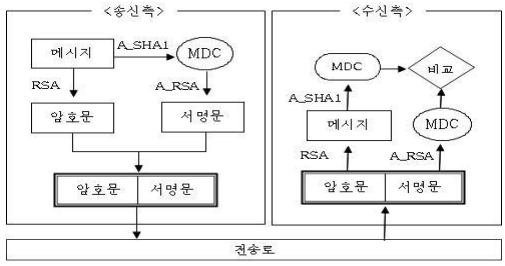

그림 7. 알고리즘 전체 흐름도
Fig. 7 Algorithm, the entire flow

송신자는 수신자의 암호화 키인 공개키를 이용하여 메시지를 암호화하여 암호문과 서명문을 함께 전송한다. 수신자는 먼저 수신된 암호문을 RSA 알고리즘의 복호화 키인 비밀키를 이용하여 원래의 메시지를 얻은 후, 이 메시지에 A_SHA1해쉬 함수를 적용하여 메시지 다이제스트 값을 생성한다. 또 수신된 서명문에 송신자의 서명 검증키인 A_RSA 공개키를 이용하여 원래의 메시지 다이제스트 코드값을 복호화 한다. 두 메시지 다이제스트 코드값을 서로 비교하여 일치하면 메시지의 변조가 없다는 것을 확인하는 메시지 인증을 할 수 있고, 서명문을 생성하고 전송한 진정한 사용자라는 신원 인증을 확인 할 수 있다. 또한 암호문과 서명문을 송신하였다는 발신 사실을 부인할 수 없게 된다.

표 1은 본 논문에서 사용하는 각 알고리즘의 종류와 기능을 간단하게 보여주고 있다.

표 1. 알고리즘의 기능 비교
Table 1. Compare the features of the algorithm

알고리즘 종류	역 할
A_SHA1	데이터의 무결성 확인
A_RSA	메시지 인증, 신원확인, 부인방지
RSA	메시지 암호화, 복호화

3.2 A_SHA1 해쉬 알고리즘

SHA1해쉬 함수는 임의 길이의 입력 메시지를 고정된 길이의 코드 값으로 압축하여 출력하는 함수로써, 메시지의 무결성 검증이나 메시지 인증에 사용된다. SHA1는 입력 메시지를 512 비트 블록으로 처리하고, 출력은 160 비트로 생성한다. 512 비트 단위 블록을 처리하는 다이제스트 함수는 모두 80단계로 구성되며 해쉬값을 계산하는 연쇄변수 H_0, H_1, H_2, H_3, H_4는 5개이다. SHA1는 임의의 길이의 메시지가 입력되면, 이 메시지를 512 비트의 배수로 만든 후, 512 비트 블록으로 나눈다. 각각의 512 비트의 메시지 블록은 32 비트 워드 $W_0, W_1, \ldots, W_{15}$로 나뉘어 지고, 80단계를 거쳐 메시지를 다이제스트한 후의 연쇄변수 H_0, H_1, H_2, H_3, H_4를 비트열로 연접한 것이 해쉬값이 된다[6].

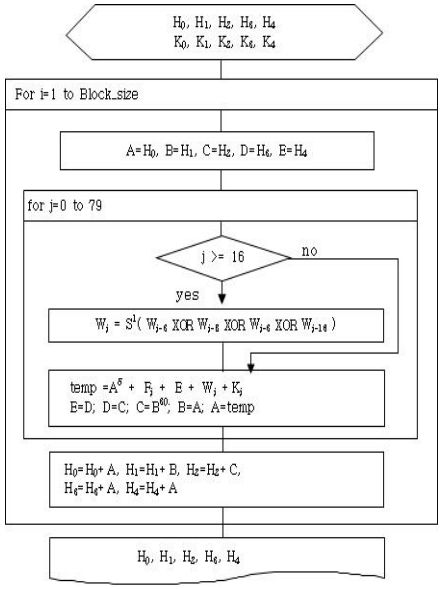

그림 8. SHA1 알고리즘 흐름도
Fig. 8 SHA1 Algorithm Flowchart

메시지 다이제트코드 값이 생성되는 자세한 과정은 다음과 같다.

(1) 초기값

A_SHA1의 변수 초기값에 사용되는 값은 16진수 표기로 다음과 같다.

H_0 = 67452301, H_1 = EFCDAB89, H_2 = 98BADCFE,
H_3 = 10325476, H_4 = C3D2E1F0

(2) 상수선언

각 연산 단계에 사용되는 상수를 16진수로 표기하면 다음과 같다.

K_i = 5A827999 $(0 \le i \le 19)$
K_i = 6ED9EBA1 $(20 \le i \le 39)$
K_i = 8F1BBCDC $(40 \le i \le 59)$
K_i = CA62C1D6 $(60 \le i \le 79)$

(3) 함수 선언

논리함수는 세 개의 워드 B, C, D를 이용하여 32비트 워드를 출력하며, 세 개의 논리함수는 다음과 같다.

F_i(B, C, D) = (B AND D) OR ((NOT B) AND D) $(0 \le i \le 19)$
F_i(B, C, D) = B XOR C XOR D $(20 \le i \le 39, 60 \le i \le 79)$
F_i(B, C, D) = (B AND C) OR (B AND D) OR (C AND D) $(40 \le i \le 59)$

(4) 메시지 다이제스트 연산

512 비트 단위로 나누어진 메시지 블록을 $M_1, M_2, \ldots M_n$이 처리되는데, M_i 처리과정은 다음과 같다.

① 각각의 Mi를 16개의 32비트 워드 $W_0, W_1, \ldots, W_{15}$로 나눈다.

② i = 16 to 79 일 때,

$W_i=S^1(W_{i-3}$ XOR W_{i-8} XOR W_{i-14} XOR $W_{i-16})$

③ $A=H_0$, $B=H_1$, $C=H_2$, $D=H_3$, $E=H_4$

④ i = 0 to 79 일 때,
$temp = S^5(A) + F_i(B, C, D) + E + W_i + K_i$
$E = D, D = C, C = S^{30}(B), B = A, A = temp$

⑤ $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$

Mn을 처리한 후 메시지 다이제스트코드값은 연쇄변수 H_0, H_1, H_2, H_3, H_4를 비트열로 연접한 것이 된다[11, 13, 15].

본 논문에서 제안하는 A_SHA1는 기존의 SHA1과는 달리 메시지 다이제스트 연산 과정의 32 비트 워드 $W_0, W_1, \ldots, W_{15}$에서 각 W_i의 16진수 값을 역방향으로 구성하여 저장하였다. 그리고 역방향으로 저장된 W_i메시지를 세 비트 씩 왼쪽으로 순환 시프트를 한다. 이 값과 시프트하기 전의 값을 XOR연산하여 W_i값을 제 구성 하였다. 즉, 위의 SHA1알고리즘 ①의 과정을 변형하고 추가시킨 것으로써 자세한 과정은 아래와 같다.

축약하려는 메시지는"To kim chul su! i love kangwon national university samcheok campus. goodby^-^"이다.

① 위의 메시지를 아스키 코드값으로 변환하여 2진수의 비트 스트링으로 나열하면 아래와 같다.

<1블럭 512비트>
w[0]=01010100 01101111 00100000 01101011
.....
w[15]=00100000 01100011 01100001 01101101

<2블럭 512비트>
.....
w[2]=01100100 01100010 01111001 01011110
w[3]=00101101 10101110*10000000 00000000
.....
w[14]=00000000 00000000 00000000 00000000
w[15]=00000000 00000000 00000010 01110000

위의 2블럭 w[3]에서 *부분에 "1"이 입력되는 것은 메시지의 끝을 표시하기 위함이다. 메시지의 끝 구분자 "1"이후에는 0이 입력되고 마지막 2블럭의 w[14]와 w[15]는 원 메시지의 비트 길이 624(78byte*8)가 입력된다.

② 비트 스트링으로 저장된 값을 16진수로 구성하여 W변수에 저장하면 아래와 같다.

< 512비트 1블럭 >
w[0]=546f206b w[1]=696d2063
w[2]=68756c20 w[3]=73752120
w[4]=2069206c w[5]=6f766520
....
w[12]=69747920 w[13]=73616d63
w[14]=68656f6b w[15]=2063616d

< 512비트 2블럭 >
w[0]=7075732e w[1]=20676f6f
w[2]=6462795e w[3]=2d5e8000
w[4]=00000000 w[5]=00000000
....
w[12]=00000000 w[13]=00000000
w[14]=00000000 w[15]=00000270

③ 위의 W변수의 값을 역방향으로 구성하여 다시 저장 하면 아래와 같다.

<512비트 1블럭>

w[0]=b602f645 w[1]=3602d696
w[2]=02c65786 w[3]=02125737
w[4]=c6029602 w[5]=025667f6
....
w[12]=02974796 w[13]=36d61637
w[14]=b6f65686 w[15]=d6163602

<512비트 2블럭>

w[0]=e2375707 w[1]=f6f67602
w[2]=e5972646 w[3]=0008e5d2
w[4]=00000000 w[5]=00000000
....
w[12]=00000000 w[13]=00000000
w[14]=00000000 w[15]= 07200000

④ 역 방향으로 저장된 각 32비트 워드 W_i에 대해 왼쪽으로 세 번 순환 시프트를 한 후, 시프트 전의 워드 값과 XOR연산을 한다.

$W_0 = S^3(W_0)$ XOR W_0, $W_1 = S^3(W_1)$ XOR W_1, $W_2 = S^3(W_2)$ XOR W_2, $W_3 = S^3(W_3)$ XOR W_3, $W_4 = S^3(W_4)$ XOR W_4, $W_5 = S^1(W_5)$ XOR W_5,

$W_6 = S^3(W_6)$ XOR W_6, $W_7 = S^3(W_7)$ XOR W_7, $W_8 = S^3(W_8)$ XOR W_8, $W_9 = S^3(W_9)$ XOR W_9, $W_{10} = S^3(W_{10})$ XOR W_{10},

$W_{11} = S^3(W_{11})$ XOR W_{11},

$W_{12} = S^3(W_{12})$ XOR W_{12},

$W_{13} = S^3(W_{13})$ XOR W_{13},

$W_{14} = S^3(W_{14})$ XOR W_{14},

$W_{15} = S^3(W_{15})$ XOR W_{15}

이후의 과정은 SHA1과 동일하다.

⑤ i = 16 to 79 일 때,

$W_i = S^1(W_{i-3}$ XOR W_{i-8} XOR W_{i-14} XOR $W_{i-16})$

⑥ $A=H_0$, $B=H_1$, $C=H_2$, $D=H_3$, $E=H_4$

⑦ i = 0 to 79 일 때,

$temp = S^5(A) + F_i(B, C, D) + E + W_i + K_i$

$E = D$, $D = C$, $C = S^{30}(B)$, $B = A$, $A = temp$

⑧ $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$

위의 메시지에 대하여 A_SHA1알고리즘으로 구현한 최종 해쉬값은 "847F0275 95F09BD8 86DE98BE A3C574BE 37586B16 "이다.

```
a_sha1(message)
{
    block_cnt=message / 64 +1;  // 블록수 계산
    w_word(message, w);

    for(c=0; c<=block_cnt; c++)
    {
       for(i=0 ; i<=15 ; i++)
       {
            inverse(w[i], w_inver);  // 32비트 워드값을 역방향으로 저장
            t_w[i]=w_inver[i] ^ left_rot(w_inver[i],3);
            // 왼쪽 3번 순환 쉬프트 후, 시프트 전의 w_inver[i]과 XOR연산
       }
       A=H0;   B=H1;   C=H2;   D=H3;   E=H4;

       for(i=0 ; i<=15 ; i++)
       {    K=select_k(i);    //상수 K선택
            F=sekect_f(i);  // 논리함수 F 선택
            temp = left_rot(A,5) + F + E + t_w[i] + K;
            E = D; D = C; C = left_rot(B,30);  B = A;  A = temp;
       }
       for(i=16 ; i<=79 ; i++)
       {    K=select_k(i);
            F=sekect_f(i);
            temp_w=left_rot(t_w[i-3] ^t_w[i-8] ^ t_w[i-14] ^
                        t_w[i-16],1);
            temp=left_rot(A,5) + F + E + temp_w + K;
            E=D;  D=C;  C=left_rot(B,30);  B=A;  A=temp;
       }
       H0 = H0 + A;  H1 = H1 + B;  H2 = H2 + C;
       H3 = H3 + D;  H4 = H4 + E;

    }
    return(H0, H1, H2, H3, H4);
}
```

그림 9. A_SHA1 알고리즘
Fig. 9 A_SHA1 Algorithm

3.3 A_RSA 서명 알고리즘

RSA가 사용하는 소인수 분해의 어려움이란 서명자가 백 자리 크기 이상의 두 개의 소수 p, q를 선택하여 n=p. q를 계산한다. 이때 서명자는 n의 소인수를 알 수 있으나 그 외의 사람은 n값을 공개하여도 n값으로부터 p와 q를 찾는 것이 어렵다는 것이다. 특히, 소수 p, q의 크기가 클수록 소인수 분해의 어려움이 커지므로 암호화시키는 비중은 커지게 되며 보안의 강도는 비례적으로 증가하게 된다. RSA 서명 방식의 서명 생성 절차와 검증 과정은 그림 10과 같다.

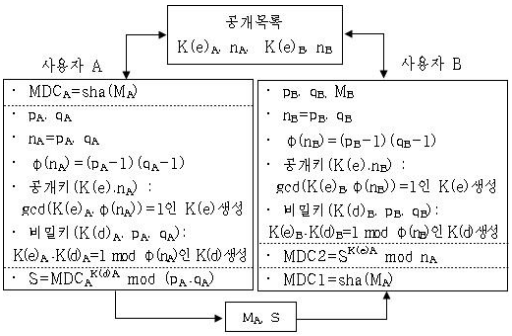

그림 10. RSA 전자서명 생성/검증 과정
Fig. 10 RSA digital signature generation / verification

본 논문에서 제안한 A_RSA 서명 알고리즘은 기존의 RSA 서명 알고리즘의 최종 서명문에 비트 연산인 순환 shift연산과 XOR연산을 추가 하여 보안을 조금 더 강화 시켰다. A_RSA 서명 알고리즘의 자세한 계산 과정을 살펴보면 아래와 같다

1) 서명문 생성과정

서명하려는 원 메시지는 "To kim chul su! i love kangwon national university samcheok campus. goodby^-^" 이며, A_SHA1해쉬 함수에 의해서 생성된 MDC 값"847F0275 95F09BD8 86DE98BE A3C574BE 37586B16"을 입력 값으로 한다.

① 사용자 A의 공개키는 K(e)A=35, nA=851이고, 비밀키 K(d)A=611, pA=23, qA=37 이다.

```
key_generation( )
{
      p=prime_create(); // 소수 p, q 생성
      q=prime_create();

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

      while(1)  // 공개키 생성
      {
            temp=rand();
            temp_gcd=cal_gcd(t, temp);
            if(temp_gcd==1 && temp > 1 && temp < t)
            {       public=temp;
                    break; }
      }
      for(i=2; i<= t; i++)  // 비밀키 생성
      {       private=( public * i ) % t ;
            if (private==1)
            {       private=i;
                    break; }
      }
      return (public, private)
}
```

그림 11. A_RSA 키 생성 알고리즘
Fig 11 A_RSA key generation algorithm

② 16진수로 구성된 MDC값으로 서명문을 생성하기 위하여 16진수 두 자리를 10진수로 변환한다.

$84_{(16)} \rightarrow 132_{(10)}$, $7F_{(16)} \rightarrow 127_{(10)}$,
$02_{(16)} \rightarrow 2_{(10)}$, $75_{(16)} \rightarrow 117_{(10)}$,
$95_{(16)} \rightarrow 149_{(10)}$, $F0_{(16)} \rightarrow 240_{(10)}$,
$9B_{(16)} \rightarrow 155_{(10)}$, $D8_{(16)} \rightarrow 216_{(10)}$,
$86_{(16)} \rightarrow 134_{(10)}$, $DE_{(16)} \rightarrow 222_{(10)}$,
$98_{(16)} \rightarrow 152_{(10)}$, $BE_{(16)} \rightarrow 190_{(10)}$,
$A3_{(16)} \rightarrow 163_{(10)}$, $C5_{(16)} \rightarrow 197_{(10)}$,
$74_{(16)} \rightarrow 116_{(10)}$, $BE_{(16)} \rightarrow 190_{(10)}$,
$37_{(16)} \rightarrow 55_{(10)}$, $58_{16)} \rightarrow 88_{(10)}$,
$6B_{(16)} \rightarrow 107_{(10)}$, $16_{(16)} \rightarrow 22_{(10)}$

③ 10진수로 변환된 MDC값과 사용자 A의 비밀키 K(d)A=611, nA=851(pA * qA)를 이용하여 서명 생성식을 적용하면 아래와 같다.

$132^{611} \bmod 851=770$, $127^{611} \bmod 851=377$,
$2^{611} \bmod 851=685$, $117^{611} \bmod 851=179$,
$149^{611} \bmod 851=704$, $240^{611} \bmod 851=109$,

$155^{611} \bmod 851=793$, $216^{611} \bmod 851=302$,
$134^{611} \bmod 851=251$, $222^{611} \bmod 851=148$,
$152^{611} \bmod 851=250$, $190^{611} \bmod 851=311$,
$163^{611} \bmod 851=708$, $197^{611} \bmod 851=811$,
$116^{611} \bmod 851=829$, $190^{611} \bmod 851=311$,
$55^{611} \bmod 851=72$, $88^{611} \bmod 851=711$,
$107^{611} \bmod 851=194$ $22^{611} \bmod 851=735$

④ 서명 생성식에 의해 계산된 결과 값과 사용자 A의 비밀키 $n_A=851(p_A \cdot q_A)$를 이용하여 순환 shift연산과 XOR연산을 수행하는 과정은 [그림 3-7]과 같다.

$770_{(10)}$	0000001100000010	$377_{(10)}$	0000000101111001
left rotation shift(n^0)	0000001101010011	left rotation shift(n^1)	0000011010100110
XOR 연산	0000000001010001 $81_{(10)}$	XOR 연산	0000011111011111 $2015_{(10)}$
$685_{(10)}$	0000001010101101	$179_{(10)}$	0000000010110011
left rotation shift(n^2)	0000110101001100	left rotation shift(n^3)	0001101010011000
XOR 연산	0000111111100001 $4065_{(10)}$	XOR 연산	0001101000101011 $6699_{(10)}$
$704_{(10)}$	0000001011000000	$109_{(10)}$	0000000001101101
left rotation shift(n^4)	0011010100110000	left rotation shift(n^5)	0110101001100000
XOR 연산	0011011111110000 $14320_{(10)}$	XOR 연산	0110101000001101 $27149_{(10)}$
$793_{(10)}$	0000001100011001	$302_{(10)}$	0000000100101110
left rotation shift(n^6)	1101010011000000	left rotation shift(n^7)	1010100110000001
XOR 연산	1101011111011001 $55257_{(10)}$	XOR 연산	1010100010101111 $43183_{(10)}$
$251_{(10)}$	0000000011111011	$148_{(10)}$	0000000010010100
left rotation shift(n^8)	0101001100000011	left rotation shift(n^9)	1010011000000110
XOR 연산	0101001111111000 $21496_{(10)}$	XOR 연산	1010011010010010 $42642_{(10)}$
$250_{(10)}$	0000000011111010	$311_{(10)}$	0000000100110111
left rotation shift(n^{10})	0100110000001101	left rotation shift(n^{11})	1001100000011010
XOR 연산	0100110011111111 $19703_{(10)}$	XOR 연산	1001100100101101 $39213_{(10)}$
$708_{(10)}$	0000001011000100	$811_{(10)}$	0000001100101011
left rotation shift(n^{12})	0011000000110101	left rotation shift(n^{13})	0110000001101010
XOR 연산	0011001011110001 $13041_{(10)}$	XOR 연산	0110001101000001 $25409_{(10)}$
$829_{(10)}$	0000001100111101	$311_{(10)}$	0000000100110111
left rotation shift(n^{14})	1100000011010100	left rotation shift(n^{15})	1000000110101001
XOR 연산	0000111111100001 $50153_{(10)}$	XOR 연산	0001101000101011 $32926_{(10)}$
$72_{(10)}$	0000000001001000	$711_{(10)}$	0000001011000111
left rotation shift(n^{16})	0000001101010011	left rotation shift(n^{17})	0000011010100110
XOR 연산	0000001100011011 $795_{(10)}$	XOR 연산	0000010001100001 $1121_{(10)}$
$194_{(10)}$	0000000011000010	$735_{(10)}$	0000001011011111
left rotation shift(n^{18})	0000110101001100	left rotation shift(n^{19})	0001101010011000
XOR 연산	0000110110001110 $3470_{(10)}$	XOR 연산	0001100001000111 $6215_{(10)}$

그림 12. A_RSA 서명문 생성 과정
Fig. 12 A_RSA signature generation process

위의 4번째 과정 후의 결과 값들을 연접(concatenation)한 값이 최종 A_RSA서명문 "81201540656699143202714955257431832149642642197033921313041254095015332926795112134706215" 이 된다.

```
a_rsa_sign(mdc, private, p, q, len)
{
        n=p * q;
        cal_mdctodec(mdc, mdc_dec);
        for(i=0; i< len; i++ )
        {       temp_sign=mdc_dec^private % n;
                sign=temp_sign ^ left_rot(n, i);
        }
        return (sign);
}
```

그림 13. A_RSA 서명 생성 알고리즘
Fig. 13 A_RSA signature generation algorithm

2) 서명 검증 과정

① 사용자 A의 공개키인 nA=851값을 이용하여 서명문에 순환 shift연산과 XOR연산을 수행하는 과정은 그림 14와 같다.

$81_{(10)}$	0000000001010001	$2015_{(10)}$	0000011111011111
left rotation shift(n^0)	0000001101010011	left rotation shift(n^1)	0000011010100110
XOR 연산	0000001100000010 $770_{(10)}$	XOR 연산	0000000101111001 $377_{(10)}$
$4065_{(10)}$	0000111111100001	$6699_{(10)}$	0001101000101011
left rotation shift(n^2)	0000110101001100	left rotation shift(n^3)	0001101010011000
XOR 연산	0000001010101101 $685_{(10)}$	XOR 연산	0000000010110011 $179_{(10)}$
$14320_{(10)}$	0011011111110000	$27149_{(10)}$	0110101000001101
left rotation shift(n^4)	0011010100110000	left rotation shift(n^5)	0110101001100000
XOR 연산	0000001011000000 $704_{(10)}$	XOR 연산	0000000001101101 $109_{(10)}$
$55257_{(10)}$	1101011111011001	$43183_{(10)}$	1010100010101111
left rotation shift(n^6)	1101010011000000	left rotation shift(n^7)	1010100110000001
XOR 연산	0000001100011001 $793_{(10)}$	XOR 연산	0000000100101110 $302_{(10)}$
$21496_{(10)}$	0101001111111000	$42642_{(10)}$	1010011010010010
left rotation shift(n^8)	0101001100000011	left rotation shift(n^9)	1010011000000110
XOR 연산	0000000011111011 $251_{(10)}$	XOR 연산	0000000010010100 $148_{(10)}$
$19703_{(10)}$	0100110011111111	$39213_{(10)}$	1001100100101101
left rotation shift(n^{10})	0100110000001101	left rotation shift(n^{11})	1001100000011010
XOR 연산	0000000011111010 $250_{(10)}$	XOR 연산	0000000100110111 $311_{(10)}$

13041(10)	0011001011110001	25409(10)	0110001101000001
left rotation shift(n^{12})	0011000000110101	left rotation shift(n^{13})	0110000001101010
XOR 연산	0000001011000100 708(10)	XOR 연산	0000001100101011 811(10)
50153(10)	0000111111100001	32926(10)	0001101000101011
left rotation shift(n^{14})	1100000011010100	left rotation shift(n^{15})	1000000110101001
XOR 연산	0000001100111101 829(10)	XOR 연산	0000000100110111 311(10)
795(10)	0000001100011011	1121(10)	0000010001100001
left rotation shift(n^{16})	0000001101010011	left rotation shift(n^{17})	0000011010100110
XOR 연산	0000000001001000 72(10)	XOR 연산	0000001011000111 711(10)
3470(10)	0000110110001110	6215(10)	0001100001000111
left rotation shift(n^{18})	0000110101001100	left rotation shift(n^{19})	0001101010011000
XOR 연산	0000000011000010 194(10)	XOR 연산	0000001011011111 735(10)

그림 14. A_RSA 서명 검증 과정

Fig. 14 A_RSA signature verification process

② 위의 XOR 연산 결과 값과 사용자 A의 공개키인 K(e)A=35, nA=851값을 이용하여 서명 검증 계산식을 적용하면 아래와 같다.

$770^{35} \bmod 851=132$, $377^{35} \bmod 851=127$,
$685^{35} \bmod 851=2$, $179^{35} \bmod 851=117$,
$704^{35} \bmod 851=149$, $109^{35} \bmod 851=240$,
$793^{35} \bmod 851=155$, $302^{35} \bmod 851=216$,
$251^{35} \bmod 851=134$, $148^{35} \bmod 851=222$,
$250^{35} \bmod 851=152$, $311^{35} \bmod 851=190$,
$708^{35} \bmod 851=163$, $811^{35} \bmod 851=197$,
$829^{35} \bmod 851=116$, $311^{35} \bmod 851=190$,
$72^{35} \bmod 851=55$, $711^{35} \bmod 851=88$,
$194^{35} \bmod 851=107$, $735^{35} \bmod 851=22$

③ 위의 10진수로 구성된 결과 값을 16진수로 변환하면 아래와 같이 MDC값으로 복원되는 것을 확인할 수 있다.

$132_{(10)} \rightarrow 84_{(16)}$, $127_{(10)} \rightarrow 7F_{(16)}$,
$2_{(10)} \rightarrow 02_{(16)}$, $117_{(10)} \rightarrow 75_{(16)}$,
$149_{(10)} \rightarrow 95_{(16)}$, $240_{(10)} \rightarrow F0_{(16)}$,
$155_{(10)} \rightarrow 9B_{(16)}$, $216_{(10)} \rightarrow D8_{(16)}$,
$134_{(10)} \rightarrow 86_{(16)}$, $222_{(10)} \rightarrow DE_{(16)}$,
$152_{(10)} \rightarrow 98_{(16)}$, $190_{(10)} \rightarrow BE_{(16)}$,
$163_{(10)} \rightarrow A3_{(16)}$, $197_{(10)} \rightarrow C5_{(16)}$,
$116_{(10)} \rightarrow 74_{(16)}$, $190_{(10)} \rightarrow BE_{(16)}$,
$55_{(10)} \rightarrow 37_{(16)}$, $88_{(10)} \rightarrow 58_{(16)}$,
$107_{(10)} \rightarrow 6B_{(16)}$, $22_{(10)} \rightarrow 16_{(16)}$

```
a_rsa_verify(sign, public, n, len)
{
        for(i=0; i< len; i++ )
        {       temp_mdc=sign ^ left_rot(n, i);
                mdc=temp_mdc^public % n;
        }
        cal_mdctohexa(mdc, mdc_hexa);
        return (mdc_hexa);
}
```

그림 15. A_RSA 서명 검증 알고리즘

Fig. 15 A_RSA signature verification algorithm

3.4 RSA 암·복호화 알고리즘

RSA 암호 알고리즘의 절차와 암호화, 복호화 과정은 그림 16과 같다.

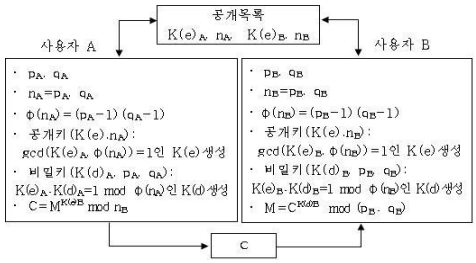

그림 16. RSA 암.복호화 과정

Fig. 16 RSA encryption, decryption Process

1) 암호화 과정

암호화 하려는 메시지는 "To kim chul su! i love kangwon national university samcheok campus. goodby^-^" 이다.

① 사용자 B의 공개키는 K(e)B=35, nB=3,869이고, 비밀키는 K(d)B=107, pB=73, qB=53 이다.

② 위의 메시지를 ASCII코드값의 10진수로 변환하면 그림 17과 같다.

메시지	T	o		k	i	m		c	h	u
ASCII코드	84	111	32	107	105	109	32	99	104	117
메시지	l		s	u	!		i		l	o
ASCII코드	108	32	115	117	33	32	105	32	108	111

메시지	v	e		k	a	n	g	w	o	n
ASCII코드	118	101	32	107	97	110	103	119	111	110
메시지		n	a	t	i	o	n	a	l	
ASCII코드	32	110	97	116	105	111	110	97	108	32
메시지	u	n	i	v	e	r	s	i	t	y
ASCII코드	117	110	105	118	101	114	115	105	116	115
메시지		s	a	m	c	h	e	o	k	
ASCII코드	32	115	97	109	99	104	101	111	107	32
메시지	c	a	m	p	u	s	.		g	o
ASCII코드	99	97	109	112	117	115	46	32	103	111
메시지	o	d	b	y	^	-	^			
ASCII코드	111	100	98	121	94	45	94			

그림 17. 메시지를 ASCII코드값으로 변환
Fig. 17 Converts the ASCII code value for the message

③ 10진수로 변환된 값과 사용자 B의 공개키 K(e)B=35, nB=3,869를 이용하여 암호문 생성식에 적용하면 아래와 같다.

$84^{35} \bmod 3869=564$, $111^{35} \bmod 3869=3529$,
$32^{35} \bmod 3869=1549$, $107^{35} \bmod 3869=3446$,
$105^{35} \bmod 3869=1695$, $109^{35} \bmod 3869=1093$,
$32^{35} \bmod 3869=1549$, $99^{35} \bmod 3869=1547$,
$104^{35} \bmod 3869=989$, $117^{35} \bmod 3869=1382$,
. .
. .
$100^{35} \bmod 3869=1579$, $98^{35} \bmod 3869=3761$,
$121^{35} \bmod 3869=546$, $94^{35} \bmod 3869=650$,
$45^{35} \bmod 3869=2542$, $94^{35} \bmod 3869=650$

위의 결과 값들을 연접한 값이 최종 암호문 "564352915493446169510931549154798913821873154915661382230515491549169515491873352932721400154934462892338176922173529233815492338289304916953529233828918731549138223381695327214002758156616953049546154915662891093154798914003529344615491547289109335621382156617061549176935293529157937615466502542650" 이 된다.

```
rsa_encryption(message, len, public, n)
{
    convert_asc(message, m_asc);
    for(i=0; i < len; i++)
        cipher=m_asc^public % n;
    return cipher;
}
```

그림 18. RSA 암호화 알고리즘
Fig. 18 RSA encryption algorithm

2) 복호화 과정

① 수신된 암호문에 사용자 B의 비밀키 K(d)B=107, pB=73, qB=53를 이용하여 복호화 생성식에 적용하면 아래와 같다.

$564^{107} \bmod 3869=84$, $3529^{107} \bmod 3869=111$,
$1549^{107} \bmod 3869=32$, $3446^{107} \bmod 3869=107$,
$1695^{107} \bmod 3869=105$, $1093^{107} \bmod 3869=109$,
$1549^{107} \bmod 3869=32$, $1547^{107} \bmod 3869=99$,
$989^{107} \bmod 3869=104$, $1382^{107} \bmod 3869=117$,
. .
. .
$1579^{107} \bmod 3869=100$, $3761^{107} \bmod 3869=98$,
$546^{107} \bmod 3869=121$, $650^{107} \bmod 3869=94$,
$2542^{107} \bmod 3869=45$, $650^{107} \bmod 3869=94$

② 위의 결과 값을 다시 ASCII 코드값으로 변환하면 원 메시지로 복원되는 것을 [그림 3-14]에서 확인할 수 있다.

ASCII코드	84	111	32	107	105	109	32	99	104	117
메시지	T	o		k	i	m		c	h	u
ASCII코드	108	32	115	117	33	32	105	32	108	111
메시지	l		s	u	!		i		l	o
ASCII코드	118	101	32	107	97	110	103	119	111	110
메시지	v	e		k	a	n	g	w	o	n
ASCII코드	32	110	97	116	105	111	110	97	108	32
메시지		n	a	t	i	o	n	a	l	
ASCII코드	117	110	105	118	101	114	115	105	116	115
메시지	u	n	i	v	e	r	s	i	t	y
ASCII코드	32	115	97	109	99	104	101	111	107	32
메시지		s	a	m	c	h	e	o	k	
ASCII코드	99	97	109	112	117	115	46	32	103	111
메시지	c	a	m	p	u	s	.		g	o
ASCII코드	111	100	98	121	94	45	94			
메시지	o	d	b	y	^	-	^			

그림 19. ASCII코드값을 메시지로 변환
Fig. 19 Converted into ASCII code value of the message

```
rsa_decryption(cipher, len, private, p, q)
{
    n=p * q;
    for(i=0; i < len; i++)
        message = cipher^private % n;
    convert_asc(message, m_asc);
    return m_asc;
}
```

그림 20. RSA 복호화 알고리즘
Fig. 20 RSA decryption algorithm

IV. 시뮬레이션 및 평가

4.1 SHA1과 A_SHA1 해쉬 함수

본 논문에서 제안한 A_SHA1는 기존의 SHA1과는 달리 메시지를 압축하기 전에 32 비트 워드 $W_0, W_1, \ldots, W_{15}$에서 각 Wi의 16진수 값을 역방향으로 구성하여 저장 하였다. 그리고 역방향으로 저장된 Wi메시지를 세 비트 씩 왼쪽으로 순환 시프트를 한다. 이 값과 시프트하기 전의 값을 XOR연산하여 메시지를 다이제스트 함으로써 역변환 공격에 대해 좀 더 보안을 강화시켰다. 그림 21은 A_SHA1해쉬 함수에 의해 메시지 다이제스트 코드값을 생성하는 시뮬레이션 화면이다.

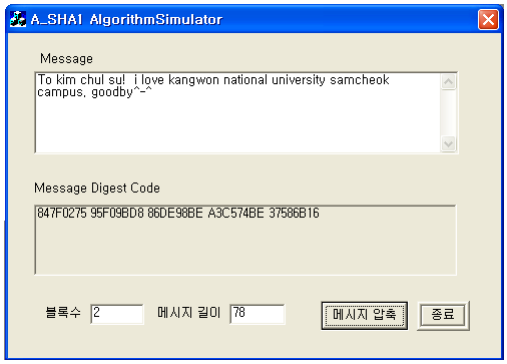

그림 21. A_SHA1 시뮬레이션 화면
Fig. 21 Simulations A_SHA1

4.2 RSA와 A_RSA 서명 알고리즘

본 논문에서 제안하는 A_RSA서명 알고리즘은 기존의 RSA서명 알고리즘의 최종 서명문에 비트 연산을 추가하였다.

A_RSA 서명 알고리즘에서 사용하는 비밀키와 공개키는 1,024비트 이상이어야만 안정성이 확보된다. 그러나 본 논문에서는 프로그램의 효율을 위하여 키의 사이즈를 16비트 이내로 제한하였다. 또한 A_SHA1함수에 의해 생성된 160비트의 메시지 다이제스트 코드값이 A_RSA 서명알고리즘에서의 'modulo n'을 취하는 n값과 비트 길이가 같도록 하여야 하나 프로그램의 효율을 위하여 n의 비트 길이를 16비트로 제한하였다. 따라서 16진수로 구성된 메시지 다이제스트 코드의 두 자리를 취하여 10진수로 변환한 후, 변환된 10진수를 16비트로 구성하여 'modulo n'을 계산하였다.

그림 22는 A_RSA 서명 알고리즘의 시뮬레이션 화면으로 입력 메시지는 앞 절의 A_SHA1해쉬 함수의 메시지 다이제스트 코드값인 "847F0275 95F09BD8 86DE98BE A3C574BE 37586B16"이며, 사용자 A의 비

밀키로 서명문을 생성하였고 사용자 A의 공개키로 서명문을 검증하였다.

그림 22. A_RSA 시뮬레이션 화면
Fig. 22 Simulations A_RSA

4.3 RSA 암·복호화 알고리즘

RSA 알고리즘 역시 암호화와 복호화에 사용하는 비밀키와 공개키의 키 사이즈가 1,024비트 이상이어야 하나, 본 논문에서는 프로그램의 효율을 위하여 키의 사이즈를 16비트 이내로 제한하였다.

그림 23은 RSA암호 알고리즘의 시뮬레이션 화면으로 입력 메시지는 "To kim chul su! i love kangwon national university samcheok campus. goodby^-^" 이며, 사용자 B의 공개키를 사용하여 암호문을 생성하고, 사용자 B의 비밀키로 복호화 하였다.

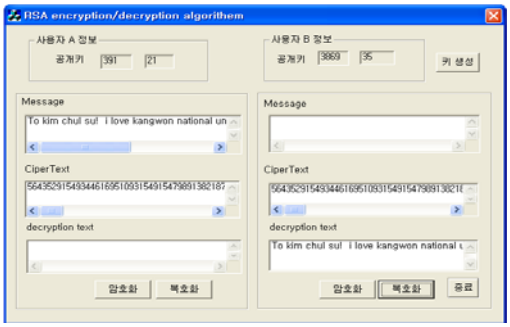

그림 23. RSA 암·복호화 시뮬레이션 화면
Fig. 23 RSA encryption, decryption simulation screen

V. 결 론

본 논문에서 분석하고 구현하는 알고리즘은 각각 A_SHA1, A_RSA, RSA이다. 이중 RSA 알고리즘은 전자서명 검증 과정에서 메시지 인증과 무결성 검증을 위해 필요한 원 메시지를 암호화하고 복호화하기 위해 프로그램을 통하여 구현하였다. 그리고 A_SHA1 해쉬 함수와 A_RSA알고리즘은 기존의 알고리즘에 계산식을 추가하여 보안을 강화 시켰다. A_SHA1해쉬 함수는 메시지를 압축하기 전에 32 비트 워드 $W_0, W_1, \ldots, W_{15}$에서 각 Wi의 16진수 값을 역방향으로 구성하여 저장 하였다. 그리고 역방향으로 저장된 Wi메시지를 세 비트 씩 왼쪽으로 순환 시프트를 한다. 이 값과 시프트하기 전의 값을 XOR연산하여 메시지를 다이제스트 함으로써 역변환 공격에 대해 좀 더 보안을 강화시켰다.

A_RSA 서명 알고리즘은 기존 RSA 서명 알고리즘의 최종 서명문에 순환 시프트 연산과 XOR연산을 추가 하여 보안을 더 강화 시켰다.

따라서 본 논문에서 사용한 A_SHA1과 A_RSA알고리즘은 기존의 알고리즘에 각각 역방향 저장과 순환 시프트연산, XOR 연산을 추가하여 보안이 향상된 것이다.

참고문헌

[1] Federal Information Processing Standards Publication(FIPS PUB), "*Secure Hash Standard*", Vol 57, No.21,Jan., 31, 1992, pp.3747-3749.

[2] M. J. B. Robshaw, "*MD2, MD4, MD5, SHA and Other Hash Function*", Technical Report TR-101, Version 3.0, RSA Laboratories, Jul. 1994.

[3] PKCS #1 v2.1 "*RSA Cryptography Standard*" (Draft.2, January 5, 2001), pp.6-20, p24-38.

[4] R. Rivest, A. Sharmir, L. Adleman, "*A Method for Obtaining*

Digital Signature and Pubic Key Cryptosystem", Communication of the ACM, Vol.21, No2, Feb, 1987, pp.120-126.

[5] jeong, Eun Hee "*A Design and Estimation of Elliptic Curve Security Socket Layer Protocol*", 2002, pp.8-10.

[6] in-seock, Cho "*Improved_Secure Electronic Transaction Protocol based on Elliptic Curve Cryptosystem*", 2003, pp.9-10, pp.35-37.

[7] 이만영외 5인 공저 "*인터넷 보안기술*", 생능출판사, 2002, pp.120-122, pp.126-127.

[8] 이만영외 5인 공저 "*차세대 네트워크 보안기술*", 생능출판사, 2002, pp.96-100.

[9] 함명묵, 이철수 공저 "*정보보호 개론*", 정익사, 2006, pp.204-207, pp.166-167.

[10] 고훈, 신용태 공저 "*인터넷과 정보보안*", 정익사, 2004, pp.12-13, pp.45-47, pp.51-52, pp.119-120, pp.179-181.

[11] 이병관, "*전자상거래 보안*", 남두도서, 2002, pp.231-232, pp.243-253.

[12] 한국정보통신기술협회, TTAS.KO-12.0001/R1, "*인증서 기반 전자 서명 알고리즘*", 2000, pp.1-6.

[13] 한국정보통신기술협회, TTAS.KO-12.0011/R2, "*해쉬 알고리즘 표준(HAS-160)*", 2005, pp.1-12.

[14] http://www.securitytechnet.com/

[15] 원동호, "*현대암호학*", 성균관대학교 전기전자 및 컴퓨터 공학부 정보통신보호 연구실, 2002, pp.185-189

저자소개

유희경(Hee-Kyung Yoo)

1997년 동국대학교 대학원 이학박사

1992년-현재 강원대학교 컴퓨터공학과 교수

※ 관심분야 : 데이터마이닝, 컴퓨터시뮬레이션, 컴퓨터보안

성 경(Kyung Sung)

2003년 한남대학교 컴퓨터공학과 (공학박사)

1994 ~ 2004년 동해대학교 컴퓨터공학과 교수

2004년~현재 목원대학교 컴퓨터교육과 교수

※ 관심분야 : 정보보호 및 정보관리, 컴퓨터네트워크, 신경회로망, 컴퓨터교육