論文 98-23-4-25

Sparse 소수를 사용한 효과적인 지수연산

正會員 고 재 영*, 박 봉 주*, 김 인 중*

A Fast Exponentiation with Sparse Prime

Jae Young Koh*, Bong Joo Park*, In Jung Kim* *Regular Members*

요 약

정보통신망에서 사용하는 공개키 암호시스템은 대부분 지수 연산을 사용한다. 하지만, 암호시스템은 안전성을 고려한 큰 수의 지수 연산을 사용하기 때문에 많은 계산 량과 준비시간을 요구한다. 이러한 문제점을 해결하기 위하여 모듈러 감소 연산에서 Montgomery, Yang, Kawamura등이 사전계산 방법, 중간계산, 그리고 테이블을 사용하는 방법을 제안하였으며, 지수 연산에서 Coster, Brickel, Lee등이 addition chain, window, 그리고, 고정된 수를 사용하는 경우 사전 계산을 하는 방법을 제안하였다.

본 논문에서는 sparse 소수를 사용한 모듈러 감소 연산 방법을 제안하고 지수연산시 계산 량을 줄이는 방법을 제안한다. 이는 이산대수 방식의 암호시스템에서 매우 효과적으로 적용할 수 있다.

ABSTRACT

Most public cryptosystem widely used in communication network are based on the exponentiation-arithmetic. But, cryptosystem has to use bigger and bigger key parameter to attain an adequate level of security. This situation increases both computation and time delay. Montgomery, Yang and Kawamura presented a method by using the pre-computation, intermediately computing and table look-up on modular reduction. Coster, Brickel and Lee persented also a method by using the pre-computation on exponentiation.

This paper propose to reduce computation of exponentiation with spare prime. This method is to enhance computation efficiency in cryptosystem used discrete logarithms.

Ⅰ. 서 론

현재 사용되는 대부분의 공개 키 암호시스템은 소인수 분해와 이산대수의 어려움에 기인한다. 대표적인 예로서 Diffile-Hellman의 키 분배 과정[1], P가 소수일 때 Z_p^* 상에서 ElGamal 프로토콜[2], P, Q가 소수일 때 모듈러 수 $n=PQ$를 사용하는 RSA 시스템[3]등이 있다. 이러한 암호시스템의 안전성을 보장하기 위하여 암호 알고리즘의 개선보다는 좀더 큰 수를 사용하는

*국방과학연구소
論文番號:97364-1010
接受日字:1997年 10月 10日

데 중점을 두고 있다. 하지만, 암호시스템에서 사용하는 큰 수의 지수 연산은 많은 계산 량을 요구한다. 일정 크기 이하의 수는 안전성에 치명적인 문제를 야기시키므로 가급적 사용을 자제한다.

계산 량을 줄이는 방법은 여러 가지 관점에서 연구되었다. 첫 번째 방법은 곱셈과 모듈러 연산 과정에서의 계산 량을 감소시키는 방법이다. Yang[5]과 Montgomery[6]는 반복적인 나눗셈을 하지 않고 빠른 모듈러 감소를 하는 방법을 제안하였다. Chivers[19]와 Kawamura[7]는 베이스단위 이내의 몫에 대한 테이블을 미리 만들고 덧셈을 사용하여 단계적으로 모듈러 감소를 하는 방법을 제안하였다. 전자의 방법은 곱셈 연산을 사용하므로 계산 량이 많고, 후자는 테이블을 사용하므로 메모리를 많이 차지하게 된다.

두 번째 방법은 지수 승 연산과정에서 계산 량을 감소시키는 방법이다. 지수 승을 계산하기 위한 가장 효과적인 방법은 addition chain[9]을 사용하는 것이다. 일반적으로 512비트 모듈러 지수연산하기 위하여 크기가 605인 addition chain을 사용할 수 있다고 알려져 있다.

세 번째 방법은 RSA 시스템의 경우에 베이스가 고정이라는 점에 고려하여 사전 계산으로 각 지수 승 위치에 대하여 사전 계산한 값을 이용하는 것이다. Brickell[10]과 P.J.Lee[11]이 제안한 이러한 방법은 베이스가 고정되지 않은 시스템에서는 사용할 수가 없는 문제점이 있다.

마지막 방법은 큰 수를 사용하면서도 계산 량을 줄이는 방법이다. Vanstone과 Zuccherato[12]의 논문에서 이러한 주제에 대하여 처음으로 언급하였다.

본 논문에서는 안전성을 해치지 않으면서 가중치가 매우 크거나 작은 sparse 소수[13]를 이용하는 방법을 고려한다. Sparse 소수의 사용은 RSA시스템과 같이 소인수 분해를 이용한 암호시스템에서는 제한된 범위로 인하여 불리하지만, ElGamal이나 Diffile-Hellnam과 같이 이산대수 방식을 이용한 암호시스템에서는 쉽게 적용 가능하다. 다만, Sparse 소수는 어느 정도 제한 범위를 가지고 있으므로 소인수 분해의 공격에 쉽게 당할 수 있다. RSA 시스템에서 sparse 모듈러가 두 개의 소수 p, q로 분리되면 자연적으로 자신의 비밀키가 노출된다. 하지만 이산대수 방식은 지수 연산의 복잡성과 관계된다. $y=x^d \bmod P$에서 비밀 키 d의 안전성은 $d=\log_x y \pmod P$의 계산에 따른 어려움에 기인한다. 따라서 이산대수를 사용하는 암호시스템에서 sparse 소수를 사용하여 계산 량을 줄이는 과정이 안전성에 어떠한 영향을 주지 않는다. 오히려 같은 계산 량에 대하여 sparse 소수를 사용하게 되면 P의 크기를 상대적으로 증가시킬 수 있게 되므로 더욱 더 안전한 암호시스템을 구축할 수 있다.

Ⅱ. 예비 사항

본 논문에서는 이후 사용하는 용어, 기호, 부울 함수의 여러 가지 암호학적 특성을 정의한다.

정의 1) 정수 P가 베이스(Base: 기저, 진법) b로 하는, 크기가 N 자리수일 때 다음과 같이 표현한다.

$$P=\sum_{i=0}^{N-1} p_i b^i, \quad \text{for} \quad 0\le p_i<b, \quad 0\le i<N-1$$

또한, 표기의 편이성을 위해 $p[i:j]=p_i b^{i-j}+\cdots+p_j \ (i>j,\ p[i:j]=p_i)$로 표현한다.

정의 2) 소수 P가 베이스 2로 하는, 크기가 N 자리수일 때 $(p_0, p_1, \cdots, p_{N-1})$에 대하여 다음과 같이 정의한다.

1. 가중치(Weight) w_p는 P를 2진수로 표현할 때 1의 개수이다.
2. 1's Sparse P는 w_p가 N에 가까운 소수로서, 0 보다 1이 많은 소수이다.
3. 0's Sparse P는 w_p가 1에 가까운 소수로서, 1 보다 0이 많은 소수이다.
4. 길이(Length) l_p는 P의 비트 수이다.

정의 3) 세 자연수 a, b, P에 대하여 다음과 같이 정의한다.

1. a와 b의 뺄셈 $a-b$가 P의 배수일 때 'a와 b는 모듈러 P에 관하여 합동 (a is congruent to b modulo P)' 라고 하고 $a\equiv b \pmod P$로 나타낸다.
2. '$b \bmod P$'라 함은 모듈러 P에 관하여 b와 합동이면서, P 보다 작고 0 이상인 자연수를 의미한다. 즉, $a=b \bmod P$는 $a\equiv b \pmod P$이고, $0\le a\le P$ 임을 의미한다.

3. 자연수 a에 대하여 $\lceil a \rceil$는 a 보다 작지 않은 최대 자연수이다.

정의 4) 두 자연수 g, b^N에 대하여 다음과 같이 정의한다.

1. $g \gg N$은 g를 $l_{b-1} \times N$번 오른쪽으로 이동시킨 값이다.
2. $g \ll N$은 g를 $l_{b-1} \times N$번 왼쪽으로 이동시킨 값이다.
3. $g \% b^N$은 g를 b^N에 대하여 모듈러 값이다.

Ⅲ. Sparse 소수

모듈러 감소를 위하여 많은 알고리즘이 있다. 예를 들어, 고전적인 방법[14], Barrett의 알고리즘[15], Yang의 알고리즘[5], Montgomery[6]의 알고리즘 등이 있다. 단지 감소 과정만을 고려한다면, 알려진 알고리즘 중에서 가장 빠른 것은 Montgomery[6]의 감소방법이다. 이 장에서는 특별한 형태의 모듈러를 ($0 < P' < b^{N'}$, $N' < N$) 이용하여 모듈러 감소 과정을 단순화시킨다.

소수 P는 $P = b^N - P'$를 만족하면, 감소 지수 형태(diminished radix form)를 갖는다고 말한다. (간단히, DR 모듈러라고 표현하자.) 이런 형태는 모듈러 감소 과정을 매우 간단하게 만들어 준다[14]. 예를 들어, x mod P (x는 $2N$-digit의 정수)를 모듈러 감소하는 경우를 고려해 보자. 합동식 $b^N = P' \bmod P$로부터 다음의 간단한 알고리듬이 모듈러 감소 연산을 수행함을 알 수 있다.

```
for(i = 2N-1 ; i >= N ; i = i-1){
    x[i-1 : i-N] = x[i-1 : i-N] + x_i P'
    if(carry) x[i-1 : i-N] = x[i-1 : i-N] + P'
    }
```

알고리듬을 수행 후 $x[N-1:0] \geq P$이면, $x[N-1:0]$에서 P 만큼 뺀 값이 최종 결과이다. 그러나, 모듈러 연산시 중간 모듈러 감소단계에서는 위의 감산과정을 할 필요가 없다. 이 알고리듬의 성능은 조건 문에 매우 밀접한 관계를 갖는다. 상대차수 δ가 $\delta = N - N'$라고 하면, 조건 문을 수행할 확률은 평균적으로 $b^{-(\delta-1)}$ 보다 작게 된다. 즉, 캐리(carry)는 x가 거의 모두 1인 경우에 발생하게 된다. 결론적으로 위의 알고리즘은 조건 문 내의 수행을 상당부분 생략할 수 있다. Bosselaers[8]이 제시한 세 가지 방법과 DR모듈러에 대한 계산 량을 표 1과 같이 비교 분석하였다.

표 1. $2N$ 크기를 갖는 x에 대하여 P의 모듈러 감소(기존의 방법)

Table 1. Modular reduction method of x mod P of $2N$ size (classical method)

알고리즘	Classical	Barrett	Montgomery	DR
곱셈	$N(N+2.5)$	$N(N+4)$	$N(N+1)$	$N(N-\delta)$
나눗셈	N	0	0	0
사전계산	*Normalization*	$b^{2N} \div P$	$-p_0^{-1} \bmod b$	$b^N - P$
작업 메모리	$2N$	$4N$	$4N$	$2N$

고속 감소 법을 위하여 DR 모듈러를 사용하는 것은 Mohan과 Adiga[16]가 처음에 제안하였는데 $P' < b^{N/2}$를 갖는 $P = b^N - P'$ 형태의 RSA 모듈러를 사용한다. 하지만, Meister[17]은 P'의 크기가 크지 않은 선택으로 인한 안전성에 문제가 있다는 것을 제시하였다. 하지만 이러한 안전성은 소인수분해 방법을 사용하는 RSA 시스템 등에 한정된 사항으로 이산대수 방식의 시스템에서의 안전성에 대해서는 언급되지 않았다.

3.1 소수가 $b^N \pm 1$ 형태인 경우

DR에서 제시한 방법의 특별한 경우로서 $P' = 1$이면 보다 효과적인 계산이 가능하며, $P' = -1$ 일 때도 같은 효과를 갖는다. 즉, P의 값이 $b^N \pm 1$의 형태이면, $P' = b^N - P = b^N - (b^N \pm 1) = \mp 1$이다. P'가 P 보다 훨씬 작은 1의 차수를 가지므로, 상대차수 $\delta = N - 1$이므로, 조건문의 수행을 할 확률은 $b^{-(N-2)}$ 보다 작게 된다.

```
for(i = 2N-1 ; i >= N ; i = i-1){
    x[i-1 : i-N] = x[i-1 : i-N] ∓ x_i
    if(carry or borrow) x[i-1 : i-N] = x[i-1 : i-N] ∓ P
}
```

위의 수식에서 for 문이 생략 가능하다. x_i는 상위 비트 열을 표현하므로 다음과 같이 표기가 가능하다.

$$\begin{aligned} &x[N:0] = \mp x[2N-1:N] + x[N-1:0] \\ &\text{if}(x \geq P)\ x[N-1:0] = x[N:0] - P \\ &\text{if}(x < 0)\ x[N-1:0] = x[N:0] + P \end{aligned} \quad (1)$$

$b^N \pm 1$의 형태를 갖는 모듈러 수를 가지고 모듈러 감소를 수행하게 되면 곱셈, 나눗셈, 사전계산이 모두 필요 없게 되면서 단 한 번의 덧셈과 한 번의 조건 문으로 모듈러 감소를 수행하게 된다.

식 (1)을 이용하여 곱셈(또는 제곱)과정에서 직접 모듈러 감소 연산을 수행 할 수 있다. 기존의 방법은 베이스 b에 대한 N의 크기를 갖는 X와 Y를 곱한 결과 값 Z는 $2N$의 크기를 가지므로 Z에 대하여 다시 모듈러 감소를 해야하지만, $b^N \pm 1$의 특성을 이용하게 되면 곱셈과정에서 직접 모듈러 감소를 수행 할 수 있는 특징을 갖는다.

X와 Y를 다음과 같이 놓자. 계산의 편의상 N을 짝수로 설정한다.

$$X = x_1 b^{N/2} + x_0,\ 0 \le x_i < b^{N/2},\ i = 0, 1$$
$$Y = y_1 b^{N/2} + y_0,\ 0 \le y_i < b^{N/2},\ i = 0, 1$$

$Z = X \times Y \bmod P$는 식 (1)의 특성에 따라 다음과 같이 표현한다.

$$\begin{aligned} &X \times Y \bmod P \\ &= (x_1 b^{N/2} + x_0) \times (y_1 b^{N/2} + y_0) \bmod P \\ &= x_1 y_1 b^N + (x_0 y_1 + x_1 y_0) b^{N/2} + x_0 y_0 \bmod P \end{aligned}$$

$$R_1 = ((x_0 y_1 + x_1 y_0) \,\%\, b^{N/2}) b^{N/2} + x_0 y_0$$
$$R_2 = x_1 y_1 + ((x_0 y_1 + x_1 y_0) \gg N/2)$$

$$Z_1 = R_1 \,\%\, b^N$$
$$Z_2 = R_2 + (R_1 / b^N)$$

$$Z = Z_1 \mp Z_2$$
$$\text{if}(Z \ge P)\ Z = Z - P$$
$$\text{if}(Z < 0)\ Z = Z + P$$

그림 1은 소수가 $b^N \pm 1$ 형태인 값을 사용할 때 곱셈 결과 식의 위치를 보여준다. 실제 사용할 수 있는 $b^N \pm 1$의 성질을 갖는 소수는 몇 개 존재하지 않는다. 그러나, $XY \bmod P$ 연산시 기존의 곱셈 방법에 대하여 작업 메모리의 크기 $2N$을 N으로 줄일 수 있다. 또한, 모듈러 감소 연산을 따로 할 필요가 없어진다.

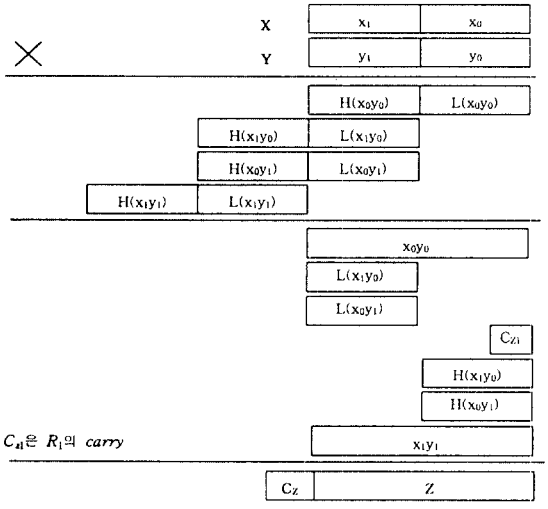


그림 1. 소수가 $b^N \pm 1$ 형태일 때 $XY \bmod P$의 계산방법
Fig. 1 Arithmetic of $XY \bmod P$ with prime $b^N \pm 1$

3.2 소수가 $b^N \pm \beta$ 형태인 경우

3.1절의 방법에서도 언급하였듯이 $b^N \pm 1$의 형태를 갖는 소수는 상당히 제한적이라고 할 수 있다. 이산대수 암호시스템에서 사용하는 소수의 유연성을 확보하기 위하여 여유 치(margin)를 갖는 것이 좋다. 이 논문에서는 소수에 범위를 좀 더 확장하여 사용할 수 있도록 $b^N \pm 1$의 주변에 상수 β 만큼의 여유 치를 갖는 소수를 사용한다. n이 $\lceil l_\beta / l_{b-1} \rceil$라고 할 때, $b^N \pm \beta$의 형태를 이용하여 $2N$의 크기를 갖는 수를 모듈러 감소하는 방법을 분석한다.

P의 값이 $b^N \pm \beta$이면, $P' = b^N - P = \mp\beta$이다.

$$x[N+n:0] = x[2N-1:N] + x[N-1:0] \quad (2)$$
$$x[N-1:0] = x[N+n:0] \bmod P \quad (3)$$

곱셈과정에서 소수 $b^N \pm \beta$의 형태를 사용하여 모듈러 감소 연산을 수행하면 식 (2)에서 한 번의 곱셈과 식 (3)에서 n 크기의 모듈러 감소 연산을 요구한다. 따라서, n이 N 보다 상대적으로 작으면 매우 효과적임을 알 수 있다. 3.1절의 Z_2에 β를 곱해주면 된다.

$$Z_2 = (R_2 + (R_1 / b^N)) \times \beta$$
$$Z = Z_1 \mp Z_2$$
$$Z = Z \bmod P \text{ (여기서, } n \textit{ size} \text{ 모듈러 감소)}$$

그림 2는 소수가 $b^N \pm \beta$ 형태인 값을 사용할 때 곱셈 결과 식의 위치를 보여준다. 비록 n 번의 곱셈과 n 크기의 모듈러 감소 연산을 사용해야 하지만 소수의 선정에 유연성이 갖게 되므로 매우 효과적이라 할 수 있다. 계산 효율성을 갖기 위한 효과적인 상수 β의 크기는 b 보다 작은 값을 사용하는 것이다. 이렇게 하면 곱셈과 모듈러 감소의 양을 줄일 수가 있다. XY mod P 연산시 기존의 곱셈 방법에 대하여 사용 메모리의 크기 $2N$을 $N+n$으로 줄일 수 있다.

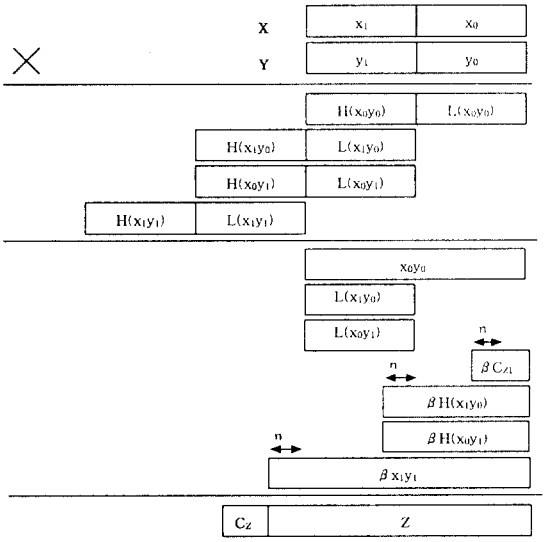


그림 2. 소수가 $b^N \pm \beta$ 형태일 때 XY mod P의 계산방법
Fig. 2 Arithmetic of XY mod P with prime $b^N \pm \beta$

3.3 소수에 특정한 상수 α를 곱했을 때 $b^{N+n} \pm 1$ 형태인 경우

3.1절과 3.2절의 방법은 소수의 영역이 b^N의 주변으로 제한된다는 것이다. 따라서, 수시로 소수를 변경하는 시스템에서는 조건에 맞는 소수를 구하는 일이 쉬운 일이 아니다. 이러한 문제점을 해결하기 위하여 소수에 상수인 특정한 수 α를 곱했을 때 $b^{N+n} \pm 1$의 형태를 만족하는 소수를 사용한다. 여기서, P가 $b^{N-1} \le P < b^N$ 이므로, αP는 $b^{N+n} \pm 1$의 값을 갖기 위해서, α는 $b^n \le \alpha < b^{n+1}$를 만족하는 값을 가져야 한다. 이제 $b^{N+n} \pm 1$을 이용하여 모듈러 감소방법을 제시한다.

αP의 값이 $b^{N+n} \pm 1$이면, $\alpha P' = b^{N+n} - \alpha P = \mp 1$이다.

$$x[N+n:0] = \mp x[2N-1:N+n] + x[N+n-1:0] \quad (4)$$

$$x[N-n:0] = x[N+n:0] \bmod P \quad (5)$$

곱셈과정에서 $b^{N+n} \pm 1$의 형태를 갖는 모듈러 수를 가지고 모듈러 감소를 수행하게 되면 3.2절과 같이 중간에 곱셈은 필요 없지만, 식 (5)과 같이 n 크기의 모듈러 감소 연산이 요구된다.

$$R_1 = (x_1 y_1 \,\%\, b^n) b^N + ((x_0 y_1 + x_1 y_0) \,\%\, b^{N/2+n}) b^{N/2} + x_0 y_0$$

$$R_2 = (x_1 y_1 \gg n) + ((x_0 y_1 + x_1 y_0) \gg (N/2+n))$$

$$Z_1 = R_1 \,\%\, b^{N+n}$$

$$Z_2 = (R_2 + (R_1 / b^{N+n}))$$

곱셈과정에서 그림 3은 소수에 특정한 상수 α를 곱했을 때 $b^{N+n} \pm 1$ 형태인 값을 사용할 때 곱셈 결과 식의 위치를 보여준다. 이러한 조건을 만족하는 소수는 상당수 존재한다. 효과적인 상수 α의 크기는 n이 작은 값을 갖도록 하는 것이다. 이렇게 하면 적은 수의 나눗셈만으로 충분히 가능하다. 역시, 곱셈 연산시 기존의 곱셈 방법에 대하여 사용 메모리의 크기 $2N$을 $N+n$으로 줄일 수 있다.

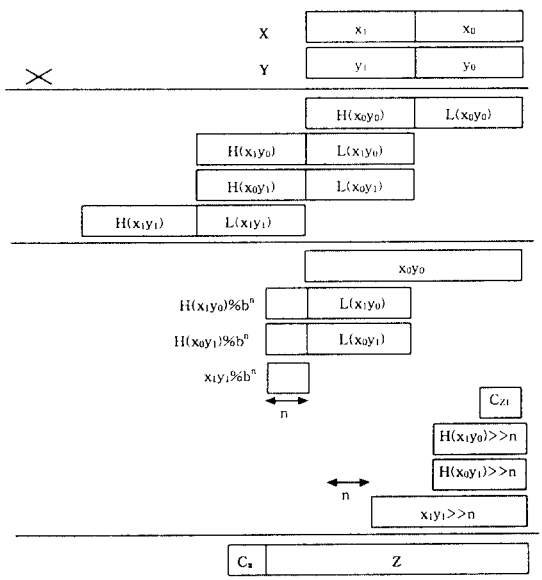


그림 3. 소수에 특정한 상수 α를 곱하여 $b^{N+n} \pm 1$ 형태일 때 XY mod P의 계산방법
Fig. 3 Arithmetic of XY mod P when $b^{N+n} \pm 1$ is $a \times$ prime

3.4 소수에 특정한 상수 α를 곱했을 때 $b^{N+n}\pm\beta$ 형태인 경우

본 논문에서 제시하는 소수는 3.2절과 3.3절의 방법을 결합한 형태로 $b^N\pm1$의 형태를 갖는 것처럼 보이지 않는다. 따라서, RSA 시스템에서도 제한적으로 사용할 수 있다. 이러한 성질을 갖는 수들은 매우 많이 분포하며 b^N 상의 주변에 있는 값이 아닌 수를 사용할 수 있으므로 안전성에서도 강하다고 할 수 있다. 본 논문에서는 소수에 상수인 특정한 수 α를 곱했을 때 $b^{N+n}\pm\beta$의 형태를 만족하는 소수를 사용한다. 여기서, P가 $b^{N-1}\le P<b^N$이므로, αP가 $b^{N+n}\pm\beta$의 값을 갖기 위해서는 α는 $b^n\le\alpha<b^{n+1}$의 값을 가져야 한다. m이 $\lceil l_\beta/l_{b-1}\rceil$라고 할 때, $b^{N+n}\pm\beta$를 이용하여 $2N$의 크기를 갖는 수를 모듈러 감소를 하는 방법을 분석해 본다.

αP의 값이 $b^{N+n}\pm\beta$이면, $\alpha P'=b^{N+n}-\alpha P=\mp\beta$이다.

$$x[N+n:0]=x[2N-1:N+n]+x[N+n-1:0] \quad (6)$$
$$x[N-1:0]=x[N+n:0] \bmod P \quad (7)$$

$b^{N+n}\pm\beta$의 형태를 갖는 모듈러 수를 가지고 모듈러 감소를 수행하게 되면 식 (6)에서 한 번의 곱셈이 요구되며, 역시 식 (7)에서 n 크기에 대한 모듈러 감소 연산이 요구된다. 만일 β의 크기 m이 $2n$ 보다 큰 값을 갖게 되면 $m-n$의 모듈러 감소 연산을 수행한다.

$$x[N+m-n:0]=x[2N-1:N+n]+x[N+n-1:0] \quad (6')$$
$$x[N-1:0]=x[N+m-n:0] \bmod P \quad (7')$$

곱셈과정에서 $b^{N+n}\pm\beta$의 형태를 갖는 모듈러 수를 가지고 모듈러 감소를 수행하게 되면 3.2절과 같이 중간에 곱셈은 필요 없지만, 식 (5)과 같이 n 크기의 모듈러 감소 연산이 요구된다.

$$Z_2=(R_2+(R_1/b^{N+n}))\times\beta$$

그림 4는 소수에 특정한 상수 α를 곱했을 때 $b^{N+n}\pm\beta$ 형태인 값을 사용할 때 곱셈 결과 식의 위치를 보여준다. 비록 β와 곱셈을 하게되면 $m-n$ 크기의 모듈러 감소 연산을 해야 하지만 소수가 널리 분포할 수 있으므로 안전성에 매우 효과적이다. 효과적인 상수 α, β의 크기는 작은 값을 사용하는 것이다. 이렇게 하면 곱셈과 모듈러 감소 연산의 양을 줄일 수가 있다. 만일 α, β의 크기가 커지면 커질수록 계산 량은 그만큼 커지므로 실제 제안한 방법의 효과가 준다. 곱셈 연산시 기존의 곱셈 방법에 대하여 사용 메모리의 크기 $2N$을 $N+n$으로 줄일 수 있다. 단, m의 크기가 $2n$의 크기보다 작거나 같은 경우에 해당되며, m이 $2n$ 보다 크면 $m-n$ 만큼 사용 메모리가 필요하다.

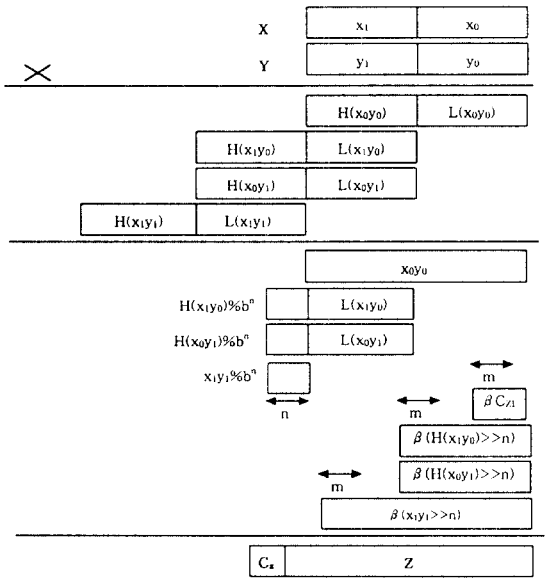


그림 4. 소수에 특정한 상수 α를 곱하여 $b^{N+n}\pm\beta$ 형태일 때 XY mod P의 계산방법

Fig. 4 Arithmetic of XY mod P when $b^{N+n}\pm\beta$ is $a\times$prime

3.5 소수에 특정한 상수 α를 곱했을 때 $b^{N+n}\pm\beta_1b^{r_1}\pm\cdots\pm\beta_kb^{r_k}$ 형태인 경우

본 논문에서는 sparse 소수의 일반적인 형태인 경우에 대하여 알아본다. 3.4절에서 설명한 것처럼 계산량을 줄이기 위하여 β의 크기가 작으면 작을수록 효과가 크다. 즉, β가 크면 클수록 모듈러 감소의 효과가 상대적으로 줄어드는 단점이 발생한다. β의 크기가 크지만, β의 형태가 $\beta_1b^{r_1}\pm\cdots\pm\beta_kb^{r_k}$ $(N>r_1>\cdots>r_k)$이면 모듈러 감소연산을 줄일 수 있다. 다만, b^{r_1}의 크기가 작을수록 효과가 커지지만, k가 많을수록 효과가 감소한다. 이러한 값들은 난수처럼 보이므로 RSA

시스템에서도 사용 가능하다. 이러한 성질을 갖는 수들은 Z_p 상에 많이 분포하며 b^N 상의 주변에서 멀리 떨어진 값을 사용할 수 있으므로 안전성에서도 강하다고 할 수 있다. 소수에 상수인 특정한 수 α를 곱했을 때 $b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k}$의 형태를 만족하는 소수를 사용한다. 여기서, P가 $b^{N-1}\le P<b^N$이므로, αP가 $b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k}$의 값을 갖기 위해서는 α는 $b^n\le\alpha<b^{n+1}$의 값을 가져야 한다. n_k가 $\lceil l_{\beta_k}/l_{b-1}\rceil$라고 할 때, $b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k}$을 이용하여 $2N$의 크기를 갖는 수를 모듈러 감소를 하는 방법을 분석해 본다.

αP의 값이 $b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k}$이면, $\alpha P'=b^{N+n}-\alpha P=b^{N+n}-(b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k})=\mp\beta_1 b^{r_1}\mp\cdots\mp\beta_k b^{r_k}$이다.

이 방법은 두 가지의 경우가 있는 데 $2n\ge(r_1+n_1)$일 때와 $2n<(r_1+n_1)$ 일 때로 구분한다.

첫 번째는 $2n\ge(r_1+n_1)$ 일 때의 표현 식이다.

$$\begin{aligned}x[N+n:0]&=x[2N-1:N+n]\times(\mp\beta_1 b^{v1}\mp\cdots\mp\beta_k b^{vk})\\&\quad+x[N+n-1:0]\\x[N+n:0]&=\mp x[2N-1:N+n]\beta_1 b^{v1}\mp\cdots\mp\\&\quad x[2N-1:N+n]\beta_k b^{vk}+x[N+n-1:0]\\x[N+n:0]&=(\mp x[2N-1:N+n]\ll r_1)\beta_1\mp\cdots\mp\\&\quad(x[2N-1:N+n]\ll r_k)\beta_k+x[N+n-1:0]\end{aligned}\tag{8}$$

$$x[N-1:0]=x[N+n:0]\bmod P\tag{9}$$

다음은 $2n<(r_1+n_1)$일 때의 표현 식이다.

$$\begin{aligned}x[N+r_1+n_1-2n:0]&=x[2N-1:N+n]\\&\quad\times(\mp\beta_1 b^{v1}\mp\cdots\mp\beta_k b^{vk})\\&\quad+x[N+n-1:0]\}\\x[N+r_1+n_1-2n:0]&=\mp x[2N-1:N+n]\beta_1 b^{v1}\mp\cdots\mp\\&\quad x[2N-1:N+n]\beta_k b^{vk}\\&\quad+x[N+n-1:0]\}\\x[N+r_1+n_1-2n:0]&=(\mp x[2N-1:N+n]\ll r_1)\beta_1\mp\cdots\mp\\&\quad(x[2N-1:N+n]\ll r_k)\beta_k\\&\quad+x[N+n-1:0]\}\end{aligned}\tag{8'}$$

$$x[N-1:0]=x[N+r_1+n_1-2n:0]\bmod P\tag{9'}$$

곱셈연산에서 $b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k}$의 형태를 갖는 모듈러 수를 가지고 모듈러 감소를 수행하게 되면 식 (8) 또는 (8')에서는 곱셈이 필요 없어지면서, 비트 shift가 일어난다. 다만 β_k와의 곱셈이 필요하다. 또한, 식 (9) 또는 (9')에서 모듈러 감소 연산이 요구된다.

$$R_3=(R_2+(R_1/b^{N+n}))$$
$$Z_2=\mp(R_3\ll r_1)\times\beta_1\mp\cdots\mp(R_3\ll r_k)\times\beta_k+R_3$$

그림 5는 소수에 특정한 상수 α를 곱했을 때 $b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k}$ 형태인 값을 사용할 때 곱셈 결과식의 위치를 보여준다. 비록 n 크기 또는 r_1+n_1-2n 크기의 모듈러 감소 연산을 사용해야 하지만 소수가 널리 분포할 수 있으므로 안전성에 매우 효과적이다. 효과적인 상수 α의 크기는 작은 값을 사용하는 것이 좋으며, r_1은 N에 비하여 작은 값이 유리하며 k도 작

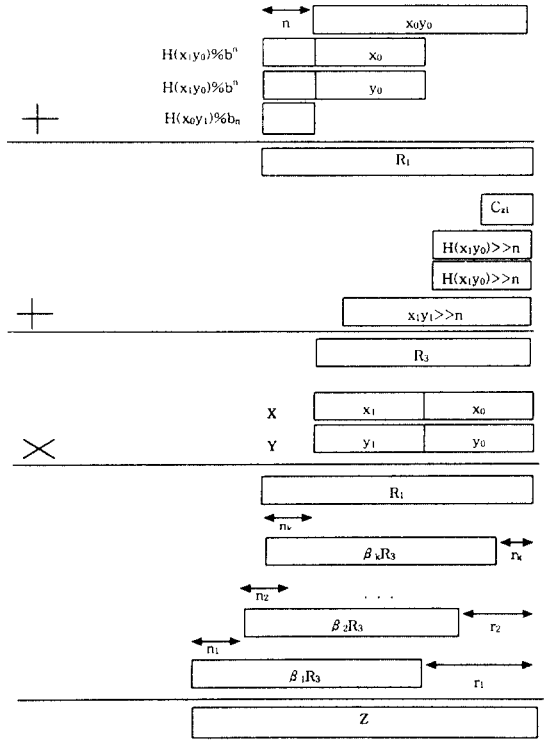


그림 5. 소수에 특정한 상수 α를 곱하여 $b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k}$ 형태일 때 XY mod P의 계산방법

Fig. 5 Arithmetic of XY mod P when $b^{N+n}\pm\beta_1 b^{r_1}\pm\cdots\pm\beta_k b^{r_k}$ is $a\times$prime

을수록 좋다. 이렇게 하면 모듈러 감소 연산 량을 줄일 수가 있다. 만일 r_1의 크기가 커지면 커질수록 계산 량은 그만큼 커지므로 실제 제안한 방법의 효과가 줄어든다. 역시, 곱셈 연산시 기존의 곱셈 방법에 대하여 사용 메모리의 크기 $2N$을 $N+n$ 또는 r_1+n_1-2n으로 줄일 수 있다.

Ⅳ. 응용 가능성 및 비교 분석

4.1 베이스 변환

암호시스템에서 사용하는 계수들은 시스템의 안전성을 보장하기 위하여 가능한 큰 정수들을 사용한다. 이러한 수들은 컴퓨터의 워드 길이보다 훨씬 더 큰 다정도 워드(multiple-precision words)이며, 베이스 b는 메모리의 효율과 속도를 향상시키기 위하여 컴퓨터와 구현할 프로그래밍 언어에 의해 결정된다.

일반적으로 베이스 b를 2의 멱승 값을 사용하지 않고 다른 베이스를 사용하여 sparse 소수의 성질이 노출되지 않도록 할 수 있다. 3.5에서 제안한 방법으로 다음과 같은 예를 들 수 있다. $P=7677_{(10)}$일 때, XY mod P를 계산해 보자. 16진수로 표현하면 $P=1DFD_{(16)}$이다. 2진수로 표현하면 $P=1110111111101_{(2)}$이고, $l_p=13$, $w_p=11$이다. 비록 가중치가 크지만 r이 매우 크므로 3.4절을 이용하는 경우 효과가 적다.

Z = XY mod P (P = 7677, $a=13$, $b^{r_1}=100$, $\beta_1=2$,
$b=10$, N = 4, n = 1, $\alpha P=99801$)
= 0x0EA1 × 0x253E mod 0x1DFD
= 3745 × 9534 mod 7677
= $37\times95\times10^4+(37\times34+45\times95)\times10^2+45\times34$ mod 7677
= $3515\times10^4+5533\times10^2+1530$ mod 7677
= $(50000+53300+1530)+(35100+500)\times2-(351+5)$ mod 7677
= $4830+35700\times2-357$ mod 7677
= $4830+71400-357$ mod 7677
= 75873 mod 7677
= $75873-9*7677$
= 6780
= 0x1A7C

위의 예제를 통하여 적절한 sparse 소수의 선택으로 지수연산시 효과적으로 계산 량을 줄일 수 있다.

4.2 비 교

이상으로 제안된 4가지의 sparse 소수를 사용하여 암호시스템을 설계하였을 때 발생하는 계산 량을 기존의 방법과 비교한 결과는 표 2와 같다. 이러한 많은 계산 량으로 인하여 복잡한 암호시스템 설계에서 실시간 처리라는 문제점을 야기시키게 된다. 하지만 이 논문에서 제안한 방식을 사용하게 되면 이와 같은 계산 량은 효과적으로 줄게된다.

표 2. $2N$ 크기를 갖는 x에 대하여 P의 모듈러 감소(제안한 방법)
Table 2. Modular reduction method of x mod P of $2N$ size (proposal method)

알고리즘	3.1절	3.2절	3.3절	3.4절	3.5절
곱셈	0	$N\times n$	0	$N\times m$	$N\times n_1$
모듈러감소 연산	0	n	n	n or $m-n$	n or r_1+n_1-n
사전 계산	*none*	*none*	*none*	*none*	*none*
작업메모리	N	$N+n$	$N+n$	$N+n$ or $N+m-n$	$N+n$ or $N+r_1+n_1-n$

표 3은 512비트에 대한 지수 승의 비교표이다. 제안한 방법은 addition chain[9]를 이용하여 사용하면 계산 량을 절반이하로 줄일 수 있다. 따라서, 지수가 가변인 비대화형 공개 키 암호시스템에서는 제안한 방법이 가장 효과적으로 사용할 수 있음을 알 수 있다.

표 3. 512비트 지수 승 연산 (평균 값)
Table 3. 512-bit exponential arithmetic(average)

알고리즘	Classic	Addition chain	Brickell	P.J.Lee	제안한 방법
제곱(A^2)	511	511	0	0	511
곱셈(AB)	256	97	130	90	256
모듈러(mod P)	767	608	130	90	0*
사전계산	none	none	necessary	necessary	none
테이블	none	none	necessary	necessary	none
단점	계산량이 많음	원도우 설계	밑수가 고정	밑수가 고정	sparse 소수

*)모듈러 감소 연산은 제곱과 곱셈 연산 내에 포함되어 있다.

이산대수 방식을 사용하는 ElGamal 알고리즘 등의 암호시스템에서 안전성은 소수의 분포나 노출 위험성보다는 수의 크기에 밀접한 관련을 갖는다.

본 논문에서 제안한 방법을 사용하면 동일한 시간 내에 기존의 키의 크기를 2배 이상으로 증가 할 수 있으므로 매우 효과적인 방법이 된다. 하지만, 소인수분해 방식을 사용하는 RSA 알고리즘등의 암호시스템에서는 안전성을 보장할 수 없다. 다만 3.5절에서 제안한 방법을 이용하면 제한된 시스템 상에서 모듈러 수로 사용이 가능하다.

4.3 시뮬레이션

A와 B의 세션키 K_{AB}를 생성하기 위하여 $K_{AB}=g^{S_A} \bmod P$를 구하기 위해서 평균적으로 768번의 곱셈과 모듈러 감소 연산이 요구된다.

이상과 같이 계산을 수행하기 위하여 암호시스템 내에 시스템 프로그래밍을 하여 알아본다. 구현환경은 TMS320C25-40MHz의 DSP를 사용하였으며 프로그램 메모리와 데이터 메모리는 모두 외부 메모리를 사용하였다. 제곱과 곱셈은 모두 동일한 함수를 사용하였으며, 모듈러 감소 연산은 Barrett의 방법을 사용하였다. Barrett의 모듈러 감소 연산에는 곱셈이 두 번 들어간다. 따라서, $AB \bmod P$를 계산하기 위해서는 3번의 곱셈연산이 요구된다.

512비트의 지수연산에 적용하며, 16비트와 16비트의 곱셈을 조합하여 사용한다. 따라서 512비트(32 word)와 512비트(32 word)의 곱셈은 1024회의 16비트와 16비트의 곱셈이 요구된다. 또한 992회의 덧셈이 요구되며 시뮬레이션 결과는 다음과 같다[8].

표 4. 512비트 크기를 갖는 $A \times B$ 곱셈연산 (평균값)
Table 4. 512-bit $A \times B \bmod P$(average)

가중치 w	소요시간 (Machine cycles)	소요시간 (40MHz)
$w_A=512,\ w_B=512$	28706	2.87msec
$w_A=0,\ w_B=0$	20057	2.00msec
$w_A=256,\ w_B=256$	21987	2.19msec(약 2.2msec)

위의 표 4를 통하여 512비트(32 word) 크기 A와 16비트(1 word) 크기의 B의 곱셈연산은 약 68μsec가 소요된다. 기존의 방법을 이용하여 세션키를 사용하여 만드는 데 걸리는 시간은 2.19msec에 총 곱셈횟수 768×3을 하면 5.04초가 걸린다. 이 수치는 곱셈연산에서만 고려된 사항이므로 다른 작업들-스마트카드로부터 데이터 입출력, 곱셈과 모듈러 감소간의 함수호출, 모듈러 감소 연산에서 발생하는 캐리에 대한 대책-은 고려하지 않은 값이다. 따라서, A가 B에 접속하기 위하여 세션키를 생성하는 시간은 최소한 5.04초 이상이다.

본 논문에서 제안한 방법을 사용하여 보자. Barrett의 모듈러 감소 연산에는 곱셈이 두 번 들어간다. 이 두 번의 곱셈을 사용하지 않게 되므로 그만큼 계산 량이 줄어든다. 따라서 평균적으로 3.1 절의 방법을 사용하게 되면, 세션키를 만들 때 1.68초가 걸린다. 3.2 논문에서 β의 값이 b 보다 작은 값인 경우에 요구되는 512비트와 16비트의 곱셈은 768개와 모듈러 감소에서 곱셈 768개가 요구된다. 이 시간은 각각 54msec가 걸린다.

따라서, A가 B와 통신하기 위한 세션키 생성에 걸리는 시간은 다음과 같다.

표 5. 세션키 생성 시간 (sec)
Table 5. Generation time of session key (average)

알고리즘	3.1절	3.2절	3.3절	3.4절	3.5절
세션키 생성	A	A +2B×n	A +B×n	A +B×m +B×n or A +B×(m-n)	A +B×n or A +B×(r_1+n_1-n)

A = 512비트와 512비트의 768회 평균 곱셈연산(1.68sec)
B = 512비트와 16비트의 768회 평균 곱셈연산(54msec)

암호시스템의 안전성을 보장하기 위하여 암호 알고리즘의 개선보다는 좀더 큰 수를 사용하는 데 중점을 두고 있다. 예를 들면, 상업적으로 사용하는 인증시스템에서 512비트 공개 키 암호방식은 이미 안전하지 못하다고 판명되었다. 최소한 768비트(또는 1024비트)의 비밀 키를 사용하도록 권고하고 있다. 이는 RSA 형태의 시스템에서 사용하는 소인수 분해의 모듈러(moduli)나 ElGamal 형태의 시스템에서 사용되는 소수(prime)의 경우에 동시에 적용된다.

하지만, 암호시스템에서 사용하는 큰 수의 지수 연산은 많은 계산량을 요구한다. 특히, 세션 키를 생성하기 위한 지수연산에 대한 계산량은 비트 수가 증가할수록 급격히 증가한다. 1024비트의 지수 연산은

512비트 지수 연산보다 약 7배의 연산 시간이 요구된다. 산술적으로 계산을 하면, Classic은 35.18초, Addition Chain은 27.96초, Brickel은 5.98초, Lee는 4.14초, 제안한 방법은 11.76초가 걸린다. 그러나, Brickel과 Lee의 방법은 밑수의 고정이외에도 사전계산 값을 저장하기 위한 메모리가 많이 요구되는 문제점이 발생한다.

특히 DSP 프로세서를 TMS320C52-80MHz를 사용하여 설계하는 경우 최소 4배의 감소효과가 있으므로 평균 2.94초가 걸리게 되며, addition chain이나 사전계산방법을 연동하여 사용할 수 있으므로 더욱 효과적이며 실시간 구현이 가능하게 된다.

V. 결 론

효과적인 암호시스템도 디지털 서명과 암호화하는 과정에서 시간을 많이 소비하게 되면 무용지물이다. 특히 스마트카드나 저가형 암호시스템과 같이 속도와 메모리가 제한된 시스템의 경우에는 기존의 방법을 사용하게되면 속도나 메모리에서 문제점이 발생한다.

본 논문에서는 sparse 소수를 사용하여 지수연산의 부담을 줄이는 방법을 제안함으로써 하드웨어와 소프트웨어의 설계를 매우 간단하게 처리할 수 있을 뿐 아니라, 암호시스템을 설계하는 데 상당한 도움을 줄 수 있다.

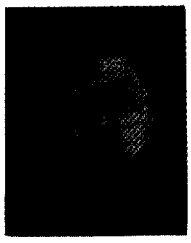
향후 소인수 분해방식의 시스템에서도 사용 가능한 sparse 모듈러 감소 연산 방법, β의 크기가 큰 경우의 적용, 안전성, N의 크기와 속도와의 관계 등이 연구돼야 할 과제이다.

참 고 문 헌

1. W.Diffie and M.E.Hellman, "New directions in cryptography," *IEEE Transaction on Information Theory*, VOL. IT-22. NO. 6, November 1976, pp. 644-654.
2. T.ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transaction on Information Theory*, VOL. IT-31. NO. 4, July 1985, pp. 469-472.
3. R.Rivest, A.Shamir and I.Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communication of ACM*, 21(2), 1978, pp. 120-126.
4. C.H.Lim and P.J.Lee, "Sparse RSA Secret Keys and Their Generation," *SAC96, workshop record*, 1996, pp. 117-131.
5. H. Morita, C. H. Yang, "A Modular-Multiplication Algorithm using Lookahead Determination", *IEICE TRAND. FUNDAMENTALS*, VOL. E76-A, NO. 1 JANUARRY 1993.
6. P. Montgomary, "Modular Multiplication without Trial Division," *Maths. of Computation*, vol. 44, no. 170, pp. 519-521, Apr. 1985.
7. S. Kawamura and K. Hirano, "A Fast Modular-Arithmetic Algorithm Using a Residue Table", *Eurocrypt 88*, pp. 245-260.
8. 박종헌, 고행석, 이주형, "곱셈 알고리즘 연구", CESD-409-971168, *국방과학연구소 기술 보고서*, pp. 5-7, 1997.
9. J. Bos, M. Coster, "Addition Chain Heuristics," *Proceedings CRYPTO'89*, Springer-Verlag Lecture Notes in Computer Science, pp. 400-407.
10. E.F.Brickell, D.M.Gordon, K.S.McCurley, and D.B.Wilson, "Fast Expontiation with Precomputation(Extended Abstract)", *In Proc. Eurocrypt'92*, Balatonfured, Hungary, NCS 839. Springer Verlag, 1992. pp. 200-207.
11. C.H.Lim and P.J.Lee "More Flexible Exponentiation with Precomputation," *In Advances in Cryptology-Crypto'94*, LNCS 839. Springer Verlag, 1994, pp. 95-107.
12. S.A.Vanstone and R.J.Zuccherato, "Short RSA Keys and Thir Generation", *J.Cryptography*, 1995, pp. 101-114.
13. 김인중, 박봉주, 고재영, "스마트카드의 비밀 계산 속도 개선에 관한 연구," *제 7회 정보보호와 암호에 관한 학술대회 연구집(WISC '95)*, pp. 200-215.
14. D.E.Knuth, *The Art of Computer Programming*, Vol.2, Seminumerical Algorithms, 2nd Edition, Addison-Wesley, Reading, Mass., 1981.

15. P.D.Barrett, "Implementing the Revest Shamir and Adleman public key encryption lgorithm on a standard digital signal processor," *Advances in Cryptology Proc. Crypto'86*, LNCS 263, A. M. Odlyzko, Ed., Springer-Verlag, 1987, pp. 311-323.
16. S.B.Mohan and B.S.Adiga, "Fast algorithms for inplement RSA public key cryptosystem", *Electronics Letters*, 21(7), 1985, pp. 761.
17. G.Meister, "On an Implementation of the Mohan-Adiga Algorithm," *In Advances in Cryptology-Eurocrypt'90*, LNCS 473, Springer Verlag, 1991, pp. 496-500.
18. A.Bosselaers, R.Govaerts, J.Vandewalle, "Comparison of three modular reduction function," *In Advances in Crypto'93*, Springer Verlag, 1993, pp. 176-185.
19. H.R.Chivers, "A Prectical Fast Exponetiation Algorithm for Public Key," *Secure Communcation Ststem*, 23 Feb, 1984, pp. 54-58.

고 재 영(Jae Young Koh) 정회원
국방과학연구소 재직
한국통신학회 논문지 제23권 3호 참조



박 봉 주(Bong Joo Park) 정회원
1986년 2월 : 서강대학교 수학과 졸업(이학사)
1988년 2월 : 서강대학교 대학원 수학과 졸업(이학석사)
1988년 3월~현재 : 국방과학연구소 재직
※주관심분야 : 정보보호, 컴퓨터 통신

김 인 중(In Jung Kim) 정회원
1990년 2월 : 충남대학교 전자공학과 졸업(공학사)
1992년 2월 : 충남대학교 대학원 전자공학과 졸업(공학석사)
1992년 3월~현재 : 국방과학연구소 재직
※주관심분야 : 컴퓨터 네트워크, 정보보호, 데이터 통신