通信情報保護學會論文誌
第 10 券 第 4 號, 2000. 12

윈도우 방법과 인수 방법을 혼합한 빠른 멱승 알고리즘*

박 희 진**, 박 근 수**, 조 유 근**

A Fast Exponentiation Algorithm Using A Window Method and a Factor Method

Heejin Park**, Kunsoo Park**, Yookun Cho**

요 약

윈도우 방법과 인수 방법을 혼합 적용하면 멱승 연산에 사용되는 곱셈 연산의 횟수를 줄일 수 있다. 지수가 512비트일 때 윈도우의 크기가 5인 윈도우 방법은 607번 정도의 곱셈 연산을 필요로 하는 데 반해 윈도우와 인수 방법을 혼합한 방법은 599번 정도의 곱셈연산을 필요로 한다. 이는 현실적으로 가능한 멱승 연산 중에서 가장 적은 수의 곱셈 연산을 요구하는 방법이다.

ABSTRACT

We show how to reduce the number of multiplications required for an exponentiation by using a window method and a factor method. This method requires 599 multiplications for a 512-bit integer exponent while the window method with window size 5 requires 607 multiplications. This method requires fewest multiplications among practical exponentiation algorithms.

keyword : *exponentiation, addition-chain, RSA cryptosystem*

Ⅰ. 서 론

멱승 연산은 주어진 양의 정수 X와 E에 대해서 X^E를 구하는 연산이고 모듈러 멱승 연산은 주어진 양의 정수 X와 E와 M에 대해서 $X^E \bmod M$을 구하는 연산이다[1]. RSA 암호시스템[2]과 Elgamal 서명 시스템[3] 그리고 DSS[4]와 같이 널리 쓰이고 있는 암호 알고리즘들이 매우 큰 지수에 대한 모듈러 멱승 연산을 포함하고 있기 때문에 모듈러 멱승 연산을 빠르게 수행하는 것은 암호학 분야에서 아주 중요하다. 모듈러 멱승 연산을 빠르게 하기 위한 노력은 크게 두 가지 방법으로 나눌 수 있다. 하나는 모듈러 멱승 연산을 수행하는데 사용되는 모듈러 곱셈 연산의 횟수를 줄이려는 방법이고[1,5,6,7~10] 또 하나는 모듈러 곱셈 연산 자체를 빠르게 하려는 방법이다[10~14]. 모듈러 곱셈 연산 자체를 빠르게 하려는 방법은 모듈러 멱승 알고리즘이 적용되는 하드웨어나 사용되는 상황에 따라 최적화의 방향이 달라질 수 있는 반면 모듈러 멱승 연산에 사용되는 모듈러 곱셈 연산의 횟수를 줄이는 방법은 하드웨어나 주변

* 이 논문은 2000년도 두뇌 한국21사업에 의하여 지원되었음.
** 서울대학교 컴퓨터공학부 ({hjpark, kpark}@theory.snu.ac.kr, cho@cse.snu.ac.kr)

상황에 상관없이 최적화될 수 있는 방법이며 모듈러 멱승 연산뿐만 아니라 멱승연산에도 사용될 수 있는 범용적인 것이다. 따라서 본 논문에서는 멱승 연산에 사용되는 곱셈 연산의 횟수를 줄이는 방법에 대해서 살펴보겠다.

멱승 연산에 사용되는 곱셈 연산의 횟수를 줄이려는 방법 중 가장 기본적인 방법은 이진 방법이다[1]. 이진 방법은 평균적으로 $1.5(\log(E+1)-1)$ 번의 곱셈 연산을 필요로 한다. 이진방법을 일반화하면 m진 방법[1]이 되는데 이 m진 방법은 평균적으로 $m-2+\log(E+1)-\log m+\log(E+1)*(1-1/m)/\log m$ 번의 곱셈 연산을 필요로 하고 지수 E가 커질수록 이진 방법보다 좀 더 좋은 성능을 보인다. 하지만 이진 방법 보다 약 m개 정도의 추가적인 메모리를 필요로 한다. m진 방법에서 메모리 사용량을 절반 정도로 줄이고 사용하는 곱셈연산의 횟수도 조금 더 줄인 방법이 윈도우 방법이다. 지수가 512비트 정수일 경우 m진 방법 중 가장 좋은 성능을 보이는 32진 방법이 636번 정도의 곱셈 연산을 필요로 하는 반면 윈도우의 크기가 5인 윈도우 방법은 607번의 곱셈 연산을 필요로 한다.

홍성민, 오상엽, 윤현수는 인수 방법을 이용하여 윈도우 방법의 성능향상을 시도하였다[10]. 그들은 512비트 지수에 대해서 평균적으로 602개의 곱셈 연산을 필요로 하는 방법을 개발했으며 우리는 이 방법을 더욱 발전시켜 평균적으로 599번의 곱셈 연산을 필요로 하는 알고리즘을 제시한다.

Ⅱ장에서는 기존의 멱승방법들에 대해서 설명하며 Ⅲ장에서는 윈도우 방법과 인수 방법을 혼합한 방법에 대해서 설명한다. Ⅳ장에서는 윈도우 방법과 인수 방법을 혼합한 방법을 기존의 방법과 비교하며 Ⅴ장에서 결론을 맺는다.

Ⅱ. 기초적인 멱승방법

2.1 이진방법[1]

n을 $\lfloor \log(E+1) \rfloor$ 로 정의하면 지수 E를 다음과 같이 표현할 수 있다.

$$E=\sum_{i=0}^{n-1} e_i 2^i \quad (e_i=0 \text{ 또는 } 1)$$

즉 E의 이진 표현은 $e_{n-1}e_{n-2}\ldots e_0$이 되며 이진 방법에서는 E의 이진 표현을 왼쪽에서 오른쪽으로 탐색하거나 오른쪽에서 왼쪽으로 탐색하면서 X^E를 구한다. 알고리즘은 [그림 1, 2]와 같다.

```
A=X; i=n-2;
while i≥0
  A=A*A
  if e_i=1 then A=A*X
  i=i-1
return A
```

[그림 1] 왼쪽에서 오른쪽으로 탐색하는 이진 방법

```
if e_i=1 then A=X
else A=1; B=X; i=1;
while i≤n-1
  B=B*B
  if e_i=1 then A=A*B
  i=i+1
return A
```

[그림 2] 오른쪽에서 왼쪽으로 탐색하는 이진 방법

위의 알고리즘을 분석해보면 이진 방법은 제곱 연산을 $n-1$ 번 수행하고 제곱이 아닌 곱셈 연산을 평균적으로는 $0.5(n-1)$ 번, 최악의 경우에는 $n-1$ 번 수행한다. 따라서 이진 방법을 수행할 때 사용되는 총 곱셈 연산의 횟수는 평균적으로 $1.5(n-1)$ 번, 최악의 경우에는 $2(n-1)$ 번이 된다.

2.2 m진 방법[1]

지수 E의 m진 표현을 $f_{l-1}f_{l-2}\ldots f_0$ ($l=\lfloor \log_m (E+1) \rfloor$)라 하면 다음과 같은 식이 성립한다.

$$E=\sum_{i=0}^{l-1} f_i m^i$$

m진 방법으로 X^E를 구하기 전에 먼저 $X^2, X^3, X^4, \ldots, X^{m-1}$을 모두 계산하여 테이블에 저장한 다음 이 테이블의 값을 이용하여 X^E를 구한다. 이진 방법과 마찬가지로 지수를 왼쪽에서 오른쪽으로 탐색하거나 오른쪽에서 왼쪽으로 탐색하면서 X^E를 구한다. 왼쪽에서 오른쪽으로 탐색하면서 X^E를 구하는 알고리즘은 [그림 3]과 같다.

일반적으로 $m=2^k$가 되도록 잡으며 이 경우 초기 테이블을 계산하는데 2^k-2번의 곱셈 연산이 필요하고 $\log(E+1)-k$번의 제곱 연산이 필요하며 평균적으로 $\frac{\log(E+1)}{k}(1-2^{-k})$ 번의 제곱이 아닌

```
A = f_{l-1};  i = l-2;
while  i≥0
    A = A^m
    if  f_i≠0  then  A = A*f_i
    i = i-1
return  A
```

〔그림 3〕 왼쪽에서 오른쪽으로 탐색하는 m진 방법

곱셈 연산이 필요하다.

2.3 윈도우 방법[5,7]

윈도우의 크기가 m인 윈도우 방법은 m진 방법을 개량한 방법이다. m진 방법이 지수 E의 이진 표현을 일률적으로 잘라 무조건 크기가 m인 윈도우를 만드는 반면에 윈도우 방법은 윈도우가 항상 1인 비트에서 시작하고 끝나도록 함으로써 초기에 테이블에 저장하는 값의 개수를 줄이고 좀 더 적은 수의 윈도우가 생기도록 하는 방법이다. 예를 들어

$E = 10001\ 00001\ 10000$

인 경우 32진 방법으로 E를 분할하면 10001 00001 10000와 같이 분할되지만 크기가 5인 윈도우 방법을 사용하면 E가 10001 00001 10000와 같이 분할된다. 이렇게 분할하면 두 가지 이점이 생기는데 그 중 하나는 초기에 테이블에 저장되어 있는 값을 계산하는데 절반정도의 곱셈 연산만 필요하다는 것이다. 윈도우의 끝나는 지점이 항상 1이므로 초기테이블에 $X^3, X^5, X^7, \ldots, X^{m-1}$과 같이 지수가 m보다 작고 홀수인 X의 멱승만을 구하면 되기 때문이다. 또 하나의 이점은 윈도우의 수가 줄어들어 제곱이 아닌 곱셈 연산의 수가 줄어든다는 것이다. 앞의 예에서 보면 32진 방법에서는 3개의 윈도우를 만들지만 크기가 5인 윈도우 방법은 2개의 윈도우를 만든다.

2.4 인수 방법[1]

$E = pq$ 이면 $X^E = (X^p)^q$이므로 $Y = X^p$를 먼저 구하고 Y^q를 구하는 방법이다. 이 방법은 p나 q의 값이 작을 때 효율적인 방법이며 p나 q가 더 이상 인수 분해 되지 않거나 p와 q의 값이 아주 커서 실질적으로 인수 분해가 불가능할 때는 사용할 수 없다. 따라서 인수 방법이 단독으로 사용되는 경우는 드물고 다른 방법과 함께 사용되어야 한다[9,14].

2.5 덧셈 사슬 방법[6]

어떤 정수 n에 대한 덧셈 사슬이란 각 $a_i\ (i \geq 1)$에 대하여 $a_i = a_j + a_k\ (k \leq j < i)$를 만족하는 다음의 정수 열을 구하는 것이다.

$a_0\ (=1),\ a_1, \ldots, a_r\ (=n)$

어떤 정수 X와 E에 대해서 X^E를 구하는데 사용되는 곱셈연산의 횟수를 줄이는 문제는 E의 짧은 덧셈 사슬을 구하는 문제로 변환될 수 있다. 주어진 정수에 대해서 가장 짧은 덧셈 사슬을 구하는 문제는 매우 어려운 문제이다. 주어진 정수가 작은 경우에는 exhaustive 탐색을 사용하여 가장 짧은 덧셈 사슬을 구할 수 있지만 주어진 정수가 커지면 커질수록 exhaustive 탐색에는 많은 시간이 소요된다. 따라서 짧은 덧셈 사슬을 구하기 위한 여러 가지 휴리스틱이 개발되어 왔으며 위에서 제시한 이진 방법, m진 방법, 윈도우 방법, 인수 방법들도 이런 휴리스틱 중의 하나이다. 윈도우 방법을 사용하여 구한 E에 대한 덧셈 사슬은 〔그림 4〕와 같다.

$$
\begin{aligned}
&1,2,3,5,\cdots,2^k-3,2^k-1,\\
&2w_{l-1},4w_{l-1},\cdots,2^{g_{l-2}}w_{l-1},2^{g_{l-2}}w_{l-1}+w_{l-2},\\
&2(2^{g_{l-2}}w_{l-1}+w_{l-2}),4(2^{g_{l-2}}w_{l-1}+w_{l-2}),\cdots,\\
&2^{g_{l-2}+g_{l-3}}w_{l-1}+2^{g_{l-3}}w_{l-2}+w_{l-3}\\
&\qquad\vdots\\
&2^{\sum_{i=1}^{l-2}g_i}w_{l-1}+2^{\sum_{i=1}^{l-3}g_i}w_{l-2}+\cdots+w_0(=E)
\end{aligned}
$$

〔그림 4〕 윈도우의 크기가 k인 윈도우 방법으로 구한 E의 덧셈 사슬. $w_i\ (0 \leq i \leq l-1)$는 E의 각 nonzero 윈도우이며 $g_i\ (0 \leq i \leq l-2)$는 w_{i+1}의 가장 오른쪽 비트와 w_i의 가장 오른쪽 비트 사이의 거리이다.

Ⅲ. 윈도우 방법과 인수 방법을 혼합한 멱승 방법

본 논문에서는 제시하는 윈도우 방법과 인수 방법을 혼합한 멱승 연산 알고리즘은 다음과 같다. 윈도우의 크기는 m으로 표시한다.

step 1. 특정 상수 Z에 대하여 Z보다 작은 모든 홀수에 대해서 가장 짧은 덧셈 사슬을 구한다. Z를 그리 크지 않은 수로 정하면 exhaustive 탐색

$$1,2,4,\cdots,2^{g_{l-1}}+w_{l-1},$$
$$2(2^{g_{l-1}}+w_{l-1}),4(2^{g_{l-1}}+w_{l-1}),\cdots,$$
$$2^{g_{l-1}+g_{l-2}}+2^{g_{l-2}}w_{l-1}+w_{l-2},$$
$$\vdots$$
$$2^{\sum_{i=1}^{l-1}g_i}+2^{\sum_{i=0}^{l-2}g_i}w_{l-1}+\cdots+w_0(=Y_1)$$

〔그림 5〕 윈도우의 크기가 k인 변형된 윈도우 방법으로 구한 Y_1의 덧셈 사슬

을 이용하여 구할 수 있다.

step 2. 주어진 지수 E를 홀수가 될 때까지 2로 나누어 얻어진 수를 E'이라 하자 그러면 어떤 정수 $c\geq 0$에 대해서 $E=E'2^c$가 성립하고 E'와 E사이의 덧셈 사슬은 E', $2E'\ldots E$가 된다. E'에 대한 덧셈 사슬을 구하기 위해서 Y를 E'으로 초기화하고 step 3과 4를 반복 적용한다.

step 3. 모든 $0\leq s<\lfloor \log(Y+1)\rfloor$에 대해서 Y를 $Y_1(=Y \bmod 2^s)$과 $Y_2(=\lfloor Y/2^s\rfloor)$로 나눈 다음 Y_1의 덧셈 사슬의 길이와 Y_2의 덧셈 사슬의 길이의 합이 가장 작은 Y_1과 Y_2를 찾는다. Y_1의 각 nonzero 윈도우를 $w_i(0\leq i\leq l-1)$라고 하고, w_{i+1}의 가장 오른쪽 비트와 w_i의 가장 오른쪽 비트 사이의 거리를 g_i $(0\leq i\leq l-2)$, w_{l-1}의 비트 수를 g_{l-1}이라 하면 Y_1의 덧셈 사슬은 〔그림 5〕와 같이 변형된 윈도우 방법으로 구할 수 있다.

Y_2에는 다음의 step 3.1과 3.2의 방법을 적용하여 덧셈 사슬을 구하고 그 중 가장 짧은 덧셈 사슬을 Y_2의 덧셈 사슬로 한다.

step 3.1. Y_2에 윈도우 방법을 적용하여 만든 덧셈 사슬의 길이

step 3.2. Z보다 작은 모든 홀수로 Y_2를 나누어 보고 Y_2를 나누는 모든 홀수 O에 대해서 다음을 수행한다. Y_2/O에 윈도우 방법을 적용하여 얻어지는 덧셈 사슬의 길이에 step 1에서 구한 O에 대한 덧셈 사슬의 길이를 더한다.

step 4. step 3에서 구한 Y_1과 Y_2에 대해서 다음을 수행한다. Y_2와 Y사이의 덧셈 사슬은 〔그림 6〕과 같이 구할 수 있다.

$$Y_2,2Y_2,4Y_2,\cdots,2^{g_{l-1}}Y_2,2^{g_{l-1}}Y_2+w_{l-1},$$
$$2(2^{g_{l-1}}Y_2+w_{l-1}),4(2^{g_{l-1}}Y_2+w_{l-1}),\cdots,$$
$$2^{g_{l-1}+g_{l-2}}Y_2+2^{g_{l-2}}w_{l-1}+w_{l-2}$$
$$\vdots$$
$$2^{\sum_{i=1}^{l-1}g_i}Y_2+2^{\sum_{i=0}^{l-2}g_i}w_{l-1}+\cdots+w_0(=Y)$$

〔그림 6〕 Y_2와 Y사이의 덧셈 사슬

그리고 step 3에서 Y_2에 대한 덧셈 사슬을 만든 방법에 따라 step 4.1이나 4.2를 수행한다.

step 4.1. step 3.1에서 Y_2의 덧셈 사슬을 구했을 경우 : Y_2에서 맨 오른쪽의 윈도우를 제거한 다음 홀수가 될 때까지 2로 계속 나누어 얻어진 수를 Y_2'이라 하자. 이를 식으로 표현하면 다음과 같다.

$$Y_2=Y_2'*2^p+Q$$
$$(p\geq k,\ 1\leq Q\leq m-1\text{인 홀수})$$

따라서, Y_2'과 Y_2사이의 덧셈 사슬은 Y_2', $2Y_2'$, $4Y_2'$, ..., $2^pY_2'$, $2^pY_2'+Q\,(=Y_2)$가 된다. Y_2'의 덧셈 사슬은 Y를 Y_2'으로 놓고 step 3을 다시 수행하여 구한다.

step 4.2. step 3.2에서 Y_2의 덧셈 사슬을 구했을 경우 : O에 대한 덧셈 사슬 $b_0(=1)$, b_1, ... $b_r(=O)$에 Y_2/O를 곱해서 Y_2/O와 Y_2사이의 덧셈 사슬 Y_2/O, b_1Y_2/O,..., $b_rY_2/O(=Y_2)$를 만든다. Y_2/O의 덧셈 사슬은 Y를 Y_2/O로 놓고 step 3을 다시 수행하여 구한다

본 논문의 방법은 윈도우 방법과 여러 개의 정수로 나누어 보는 인수 방법을 혼합하여 사용하기 때문에 기존의 윈도우 방법[7]보다 짧은 덧셈 사슬을 생성하게 된다. 윈도우의 크기가 5일 때 1010001에 본 논문의 방법과 기존의 윈도우 방법을 적용하여 얻어지는 덧셈 사슬을 비교하면 다음과 같다.

1010001에 대한 덧셈 사슬을 구하기 전에 테이블을 초기화하기 위하여 다음과 같은 덧셈 사슬이 필요하다.

1,10,11,101,111,1001,1011,1101,1111,10001,
10011,10101,10111,11001,11011,11101,11111

초기화한 테이블의 값을 이용하여 1010001에 윈도우의 크기가 5인 윈도우 방법[7]과 본 논문의 방법을 적용하면 각각 다음과 같은 덧셈 사슬을 얻는다.

윈도우 방법
101 1010 10100 101000 1010000 1010001

본 논문의 방법
11011 10*11011 11*11011(=1010001)

덧셈 사슬의 총 길이는 윈도우의 크기가 5인 윈도우 방법의 경우 22(=16+6)이고 본 논문의 방법의 경우 19(=16+3)이다.

Ⅳ. 성능비교

지수가 512비트일 때와 1024비트일 때 이진 방법, m진 방법, 윈도우 방법, 홍성민외 2인의 방법, 그리고 본 논문에서 제시한 방법을 비교하였다.

지수가 512비트와 정수일 때 이진 방법, 32진 방법, 윈도우의 크기가 5인 윈도우 방법, 윈도우의 크기가 5인 홍성민외 2인의 방법, 윈도우의 크기가 5인 윈도우 방법과 Z가 100인 인수 방법을 혼합한 본 논문의 방법을 비교하였다. m진 방법 중 32진 방법을 사용하고 윈도우 방법에서 윈도우의 크기를 5로 정한 이유는 지수가 512비트 정수일 때를 기준으로 최적화 하였기 때문이다. 이진 방법과 32진 방법은 이론적으로 분석이 가능하며 평균적으로 각각 768번과 636번의 곱셈 연산을 필요로 한다. 윈도우의 크기가 5인 윈도우 방법도 좀 복잡하기는 하지만 이론적으로 분석이 가능하며[7,15] 약 607개 정도의 곱셈 연산을 필요로 한다.

하지만 홍성민외 2인의 방법과 본 논문의 방법을 혼합한 방법은 이론적으로 분석하는 것이 간단하지 않으므로 난수를 10000개 발생시켜서 실제로 윈도우 방법과 윈도우 방법과 인수 방법을 혼합한 방법을 적용시켜서 나온 결과 값들의 평균을 내어 보았다. 윈도우 방법은 이론적으로 분석이 가능함에도 실제로 적용시켜 본 이유는 실험의 정확성을 판단하기 위해서였다. 그 결과 윈도우 방법 (윈도우의 크기 5)은 607번 정도의 곱셈 연산을 필요로 하였으며 홍성민외 2인의 방법은 602번 정도의 곱셈 연산을 필요로 하였으며 우리의 윈도우 방법 (윈도우의 크기 5)과 인수 방법 (Z=100)이 혼합된 방법은 599번 정도의 곱셈 연산을 필요로 하였다. 이 내용을 〔표 1〕로 정리하였다.

〔표 1〕 지수가 512비트일 때 곱셈 횟수 비교

멱승 방법	곱셈횟수	분석 방법
이진 방법	768	확률적 분석
32진 방법	636	확률적 분석
윈도우 방법 (크기=5)	607	확률적 분석
홍성민외 2인의 방법	602	실험 결과
본 논문의 방법	599	실험 결과

〔표 2〕 지수가 1024비트일 때 곱셈 횟수 비교

멱승 방법	곱셈횟수	분석 방법
이진 방법	1535	확률적 분석
64진 방법	1244	확률적 분석
윈도우 방법 (크기=6)	1195	확률적 분석
홍성민외 2인의 방법	1189	실험 결과
본 논문의 방법	1186	실험 결과

지수가 1024비트 정수일 때 이진 방법, 64진 방법, 윈도우의 크기가 6인 윈도우 방법, 윈도우의 크기가 6인 윈도우 방법과 Z가 300인 인수 방법을 혼합한 방법을 비교하여 〔표 2〕로 정리하였다. m진 방법 중 64진 방법을 사용하고 윈도우 방법에서 윈도우의 크기를 6으로 정한 이유는 지수가 1024비트 정수일 때를 기준으로 최적화 하였기 때문이다. 이진 방법, 64진 방법, 윈도우 방법은 이론적으로 분석을 하였으며 홍성민외 2인의 방법과 본 논문의 방법은 10000개의 난수를 발생시켜서 실제로 적용시킨 다음 평균을 구한 것이다.

Ⅴ. 결 론

본 논문에서 제시한 윈도우 방법과 인수 방법을 혼합한 방법은 아무래도 기존의 방법으로 덧셈 사슬을 구할 때보다는 시간이 많이 걸린다. Supersparc 167MHZ에서 실험한 결과 지수가 512비트인 경우 하나의 덧셈 사슬을 구하는데 평균적으로 2.5초 정도가 소요된다. 그러므로, 오프라인으로 지수의 덧셈 사슬을 구해놓고 입력으로 밑이 들어오면 미리 구해진 덧셈 사슬을 따라 멱승 연산을 수행하는 것이 바람직하다. 따라서 Elgamal이나 DSS처럼 밑이 고정되고 지수가 변하는 암호시스템 보다는 RSA처럼 지수가 고정되고 밑이 변하는 시스템이 이 방법을 적용하기에 적합하다.

본 논문에서는 난수를 발생시켜서 결과를 비교했지만 이론적인 분석이 필요하며 윈도우 방법을 분석한 방법을[7,15] 확장시켜 분석을 시도해 볼만하다.

참 고 문 헌

[1] D.E. Knuth, "The Art of Computer Programming: Seminumerical Algorithms", Addison-Wesley, U.S.A. 1981.

[2] R.L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) 1978, pp. 120~126.

[3] T. Elgamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions on Information Theory 31 (4) 1985, pp. 469~472.

[4] National Institute for Standards and Technology, Digital Signature Standard (DSS), Federal Register 56 169, 1991.

[5] J. Bos and M. Coster, Addition chain heuristics, Proc. Crypto'89, Lecture Notes in Computer Science Vol. 435, pp. 400~407, Springer-Verlag, 1990.

[6] P. Downey, B. Leong and R. Sethi, Computing sequences with addition chains, SIAM Journal on Computing 10 (3) 1981, pp. 638~646.

[7] C.K. Koc, Analysis of sliding window techniques for exponentiation, Computers & Mathematics with Applications 30 (10) 1995, pp. 17~24.

[8] Y. Yacobi, Exponentiating faster with addition chain, In Proc. Eurocrypt'90, Lecture Notes in Computer Science Vol. 473, Springer-Verlag, 1990, pp. 222~229.

[9] C. D. Walter, Exponentiation Using Division Chains, IEEE transactions on computers 47 (7) 1998, pp. 757~765.

[10] 홍성민, 오상엽, 윤현수, 모듈라 멱승 연산의 빠른 수행을 위한 덧셈 사슬 휴리스틱과 모듈라 곱셈 알고리즘들, 통신정보보호학회논문지, 제7권, 제2호, 1997, pp. 73~91.

[11] S. Hong, S. Oh, and H. Yoon, New modular multiplication algorithms for fast modular exponentiation, Proc. Eurocrypt'96, Lecture Notes in Computer Science Vol. 1070, Springer-Verlag, 1996, pp. 166~177.

[12] P. L. Montgomery, Modular multiplication without trial division, Mathematics of Computation 44 (170) 1985, pp. 519~521.

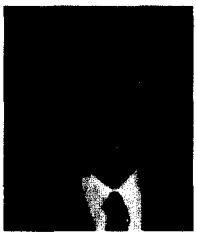
[13] H. Morita, A fast modular-multiplication algorithm based on a higher radix, Proc. Crypto'89, Lecture Notes in Computer Science Vol. 435, Springer-Verlag, 1990, pp. 387~399.

[14] C. D. Walter, Faster modular multiplication by operand scaling. Proc. Crypto '91, Lecture Notes in Computer Science Vol. 576, Springer-Verlag, 1992, pp. 313~323

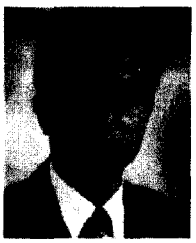
[15] H. Park, K. Park, and Y. Cho, Analysis of Variable Length Nonzero Window method for Exponentiation, Computers & Mathematics with applications 37, 7, 1999, pp. 21~29.

〈著者紹介〉

박 희 진 (Heejin Park)
1994년 2월 : 서울대학교 컴퓨터공학과 학사
1996년 2월 : 서울대학교 컴퓨터공학과 석사
1996년 3월~현재 : 서울대학교 컴퓨터공학부 박사과정
〈관심분야〉 컴퓨터이론, 암호학, 병렬계산



박 근 수 (Kunsoo Park) 종신회원
1983년 : 서울대학교 컴퓨터공학과 학사
1985년 : 서울대학교 컴퓨터공학과 석사
1991년 : 미국 Columbia University 전산학 박사
1991년~1993년 : 영국 University of London, King's College 조교수
1993년~현재 : 서울대학교 컴퓨터공학부 부교수
〈관심분야〉 컴퓨터이론, 암호학, 병렬계산



조 유 근 (Yookun Cho)
1971년 : 서울대학교 건축공학과 학사
1978년 : 미국 University of Minnesota 전산학 박사
1979년~현재 : 서울대학교 컴퓨터공학부 교수
1984년~1985년 : 미국 University of Minnesota 교환교수
1993년~1995년 : 서울대학교 중앙교육연구전산원장
1995년 : 한국정보과학회 부회장
1999년~현재 : 서울대학교 공과대학 부학장
〈관심분야〉 운영체제, 알고리즘 설계 및 분석, 암호학