

제곱-나눗셈 모듈러 지수연산법

이 상 운*

Square-and-Divide Modular Exponentiation

Sang-Un, Lee *

요 약

암호학의 암호 생성과 해독, 소수판별법의 성능은 대부분 $a^b \pmod{m}$ 의 모듈러 지수연산의 효율적 구현여부로 결정된다. 모듈러 지수연산법에는 제곱-곱셈 방식의 표준 이진법이 최선의 선택으로 알려져 있다. 그러나 큰 자리수의 b 에 대해서는 사전처리를 하는 $n-ary$, ($n \geq 2$)이 보다 효율적으로 적용된다. 본 논문에서는 모듈러 지수 나눗셈 방법을 적용한 제곱-나눗셈법과 사전처리 없는 $n-ary$ 제곱-나눗셈법을 제안하였다. 제곱-나눗셈법은 b 가 $2^k + 2^{k-1}$ 에 근접한 값 또는 2^{k+1} 에 근접한 경우 수행횟수 측면에서 가장 효율적임을 알 수 있었다. 나머지 값들에 대해서는 사전처리 없는 $n-ary$ 제곱-나눗셈법을 적용하는 것이 사전처리를 하는 일반적인 $n-ary$ 법에 비해 수행횟수 측면에서 효율적임을 보였다.

▶ Keywords : 모듈러 지수연산 법, 이진법, n -항법, 제곱-곱셈법, 제곱-나눗셈법

Abstract

The performance and practicality of cryptosystem for encryption, decryption, and primality test are primarily determined by the implementation efficiency of the modular exponentiation of $a^b \pmod{m}$. To compute $a^b \pmod{m}$, the standard binary squaring (square-and-multiply) still seems to be the best choice. However, in large b bits, the preprocessed $n-ary$, ($n \geq 2$) method could be more efficient than binary squaring method. This paper proposes a square-and-divide and unpreprocessed $n-ary$ square-and-divide modular exponentiation method. Results confirmed that the square-and-divide method is the most efficient of trial number in a case where the value of b is adjacent to $2^k + 2^{k-1}$ or to 2^{k+1} . It was also proved that for b out of the beforementioned range, the unpreprocessed $n-ary$ square-and-divide method yields higher efficiency of trial number than the general preprocessed $n-ary$ method.

•제1저자 : 이상운

•투고일 : 2013. 2. 5, 심사일 : 2013. 3. 18, 게재확정일 : 2013. 3. 25.

* 강릉원주대학교 멀티미디어공학과 (Dept. of Multimedia Eng., Gangneung-Wonju National University)

▶ Keywords : modular exponentiation, binary method, n -ary method, square-and-multiply, square-and-divide

I. 서론

a^b 연산을 지수연산 (exponentiation)이라 하며, $a^b \pmod m$ 은 a^b 의 값을 m 으로 나눈 나머지를 구하는 것으로 모듈러 지수연산 (modular exponentiation)이라 한다. 모듈러 지수연산은 암호학 (cryptography) 분야의 암호 생성과 해독과 더불어 소수판별법 (primality test)에도 널리 사용되고 있다[1,2].

대표적인 모듈러 지수 연산법에는 $a^b \pmod m$ 의 $b = b_k b_{k-1} b_{k-2} \dots b_1 b_{0(2)}$ 에 대해 LSB (least significant bit)에서 MSB (most significant bit)로 n 비트씩 분할 (n -진법으로 변환)한 $n_l n_{l-1} \dots n_1 n_0$, ($l = \lceil k/n \rceil$)에 대해 n_l 은 l 비트의 값을 초기치 a^l 로 설정하고, n_{l-1} 부터 n_0 까지 각 비트에 대해 d 회 곱셈과 각 비트 값에 대해 지수항 덧셈 (곱셈)을 수행하는 n 항 법 (n -ary method)이 있다. n -ary법은 각 비트 값이 가질 수 있는 가능한 경우수인 2^n 개 중에서 0,1을 제외한 $2^n - 2$ 개를 사전에 계산하여 저장한 후 비트 값을 곱할 때 사용한다[3,4]. 지수연산은 $(a^b)^c = a^{b \times c}$, $a^b \times a^c = a^{b+c}$ 로 계산된다. 이진법 (binary method)은 1-ary로 곱셈-곱셈 (square-and-multiply) 방식이다. 2-ary는 각 비트에 대해 $(a^2)^2 = a^{2 \times 2} = a^4$ 의 2회 곱셈과 비트 값에 대한 곱셈을 수행한다. 따라서 n -ary법은 n 회 곱셈 (n 승)을 수행하고 비트 값을 곱셈하는 방식이라 할 수 있다. 이외에도 추가 사슬 (addition chain)법[4]과 몽고메리 (Montgomery) 감소법[5] 등이 있다.

n -ary의 곱셈횟수는 $b = b_k b_{k-1} b_{k-2} \dots b_1 b_{0(2)}$ 에 대해 사전처리 (preprocessing) 곱셈 수행횟수 $2^n - 2$ 와 $\lceil k/n \rceil - 1$ 개의 n -진법 비트 값에 대한 곱셈횟수로 결정된다.

일반적으로 암호와 복호에 사용되는 큰 자리수에 대해서는 n -ary, ($n > 1$)법이 이진법보다 효율적인 것으로 알려져 있다[3,6-10].

본 논문은 n -ary의 곱셈 횟수 k 를 감소시키는 방법을

제안한다. 적용된 방법은 $k+1$ 또는 k 회의 곱셈을 수행하고, 곱셈 또는 나눗셈을 수행하는 방법으로 일반적으로 “곱셈-나눗셈 법”이라 칭한다.

2장에서는 n -ary법을 고찰한다. 3장에서는 지수 나눗셈 방법을 제안하고 곱셈-나눗셈법과 사전처리 없는 n -ary법에 곱셈 또는 나눗셈을 하는 방법을 접목시킨 알고리즘을 제안하고 적합성을 검증한다.

II. n -ary 모듈러 지수연산법

본 장에서는 적용이 복잡한 추가 사슬법[4]과 몽고메리 감소법은 제외하고, 일반적으로 적용되고 있는 n -ary법과 제안된 곱셈-나눗셈법의 성능을 비교하기 위해 n -ary법에 대해 사례를 통해 수행횟수를 고찰해 본다.

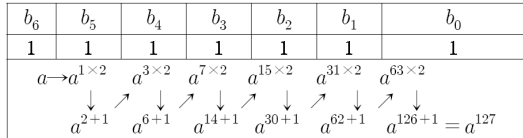
모듈러 지수연산에 일반적으로 적용되고 있는 n -ary법은 그림 1과 같이 수행된다[4]. 여기서는 일반적으로 알려진 n -ary법을 간략히 표현하였다. 일반적으로 알려진 방법은 초기치를 $a_1 = 1$, $c = 1$ 로 하여 그림 1보다 수행횟수가 1회 추가된다.

```

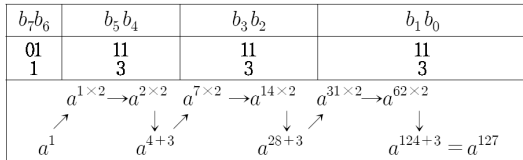
 $a^b \pmod m, b = b_k b_{k-1} \dots b_1 b_{0(2)}$ 
Preprocessing (사전 처리)
 $a_1 = a$ 
if  $n \geq 3$  then /* 1-ary와 2-ary는 수행 않음 */
for  $i = 2$  to  $(2^n - 1)$  do
 $a_i = (a_{i-1} \times a) \pmod m$ 
end
end
Modular Exponentiation ( $a, b, m$ )
 $c = a_i$  /*  $b_k$  값에 해당하는  $a_i$  값. 만약,  $b_k = 10$ 이면  $a * /$ 
for  $i = k-1$  down to 0 do
for  $j = 1$  to  $d$  do
 $c = (c \times c) \pmod m$ 
end
 $c = (c \times a_i) \pmod m$  /*  $b_i$  값에 해당하는  $a_i$  값 */
end
return  $c$ 
    
```

그림 1. 일반적인 n -ary ($n \geq 1$) 지수연산법
Fig. 1. General n -ary ($n \geq 1$) Modular Exponentiation

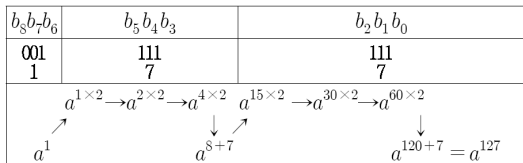
$b = 127 = 1111111_{(2)}$ 인 경우, $n-ary$ 를 적용하는 방법은 그림 2에 제시하였다. 이진법은 $b_6 = 1$ 에서 초기치로 a 를 설정하고 $b_5 = 1$ 에서 $a^1 \rightarrow (a^1)^2$ 의 제곱과 비트 값 1 ($1 = a^1$)에 대해 $(a^2)(a^1) = a^3$ 으로 a^3 이 된다. 이를 $b_4 = 1, b_3 = 1, \dots, b_0 = 1$ 에 계속 적용하면 총 곱셈횟수는 12회를 수행한다. 2-ary법은 $b = 127 = 0111111_{(2)} = 1333_{(4)}$ 에 대해 사전에 $10 = a \rightarrow a^2, 11 = (a^2)a^1 = a^3$ 의 2회 곱셈으로 값을 저장한다. 다음으로 $b_7b_6 = 01$ 비트 값에 대해 $(a)^1 = a$ 로 곱셈을 수행하지 않고 초기치로 설정되며, $b_5b_4 = 11 = 3$ 에 대해 $(a^2) \rightarrow (a^2)^2 = a^4$ 의 2회 제곱과 비트 값 $11 = 3$ 에 대해 $a^4 \times a^3 = a^{4+3} = a^7$ 의 1회 곱셈을 수행한다. 마찬가지로 $b_3b_2 = 11$ 와 $b_1b_0 = 11$ 에 대해서도 각각 3회의 곱셈을 수행하므로 총 곱셈 수행횟수는 $2+3+3+3=11$ 회이다. 3-ary를 적용하면 $127 = 001111111 = 177_{(8)}$ 에 대해 $2^k - 2 = 6$ 회의 곱셈으로 a^2, a^3, \dots, a^7 을 사전에 계산하며, $b_8b_7b_6 = 001 = a, b_5b_4b_3 = 111$ 에 대해 $(a)^2 \rightarrow (a^2)^2 \rightarrow (a^4)^2$ 의 3회 제곱과 $111 = 7$ 에 대해 $a^8 \times a^7 = a^{8+7} = a^{15}$ 의 1회 곱셈을 수행한다. 이런 과정을 거치면 총 곱셈 수행횟수는 $6+0+4+4=14$ 회를 수행한다. 결국, $b = 127$ 인 경우 $d-ary$ 중에서 2-ary의 11회가 가장 좋은 결과를 얻을 수 있다.



(a) 이진법



(b) 2-ary법



(c) 3-ary법

그림 2. a^{127} 계산 $n-ary$ 법

Fig. 2. $n-ary$ Method for a^{127} Computation

$n-ary$ 는 $2^n - 2$ 회의 사전처리, $(\frac{k}{n} - 1) \cdot n = k - n$ 회 제곱과 $(\frac{k}{n} - 1) \cdot (\frac{2^n - 1}{2^n})$ 회 곱셈을 수행한다. $(\frac{2^n - 1}{2^n})$ 은 2^n 개의 가능한 비트 값 중에서 0을 제외한 $2^n - 1$ 개에 대해서만 곱셈이 수행됨을 의미한다. 이 공식을 적용하면 이론적으로는 이진 자리수 $l(k)$ 에 대해 최적인 n 를 결정할 수 있다. $2 \leq k \leq 2048$ 에 대해 계산한 결과 최적의 $l(k)$ 범위는 표 1과 같이 얻었다. 따라서 $n-ary$ 를 적용할 경우 b 의 2진 자리수 $l(k)$ 에 따라 표 1을 적용하면 곱셈 수행횟수를 감소시킬 수 있다. 그러나 실제로는 $b_i = 1$ 의 개수에 영향을 받는다.

표 1. $l(k)$ 범위에 대해 최적인 $n-ary$

Table 1. Optimal $n-ary$ for Ranges

$l(k)$	$n-ary$
(2, 6]	1-ary
(7, 34]	2-ary
(35, 121]	3-ary
(122, 368]	4-ary
(369, 1043]	5-ary
(1044, 2048]	6-ary

III. 제공-나눗셈 모듈러 지수연산법

지수 나눗셈은 $a^{b-c} = \frac{a^b}{a^c}$ 으로 계산된다. 그러나 모듈러

지수 나눗셈 $a^{(b-c)} \pmod{m} = \frac{a^b \pmod{m}}{a^c \pmod{m}}$ 을 계산하는

방법은 제안되지 않고 있다. 만약, 모듈러 나눗셈 지수연산법을 적용할 수 있다면 $n-ary$ 법의 곱셈 수행횟수를 크게 감소시킬 수도 있을 것이다. 따라서 본 절에서는 모듈러 나눗셈 지수연산법을 제안하여 제공-나눗셈으로 지수연산을 수행하는 방법을 제안한다. 모듈러 나눗셈은 그림 3의 방법을 적용한다. 왜냐하면 모듈러 나눗셈은 n 이 일반적으로 십진수가 아니기 때문에 직접 나눗셈을 할 수 없으며, n 의 임의의 배수를 더한 값을 나누어 정수가 되어야 되기 때문이다.

while $j = 1$ to $e = \text{정수}$

$e = \frac{a^{2^{k+1}} \pmod{m} + (m \times j)}{a^{2^{k+1}-b} \pmod{m}}, e = \text{정수}$

$j = j + 1$

end

그림 3. 모듈러 나눗셈 방법

Fig. 3. Modular Division Method

그림 3의 모듈러 나눗셈 방법을 채택한 제곱-나눗셈 지수 연산법은 그림 4에 제시하였다. 그림 4는 $2^k < b < 2^{k+1}$ 에서 $2^k + 2^{k-1} < b$ 에 존재하는 경우 2^{k+1} 을 기준으로 비트 값을 지수 뺄셈한 경우이며, $2^k < b < 2^k + 2^{k-1}$ 에 대해서는 2^k 를 기준으로 곱셈 또는 나눗셈을 수행하는 방법을 적용하면 된다.

```

 $a^b \pmod n, b = (b_k, b_{k-1}, \dots, b_1, b_0)_2$ 
Modular-Exponentiation ( $a, b, m$ )
 $c = a$ 
for  $i = 1$  to  $k+1$  do
     $c = (c \times c) \pmod m$ 
end
while  $j = 1$  to  $e = \text{정수}$  do
     $e = \frac{a^{2^{j+1}} \pmod m + (m \times j)}{a^{2^{j+1}-b} \pmod m}$ 
     $j = j + 1$ 
end
return  $e$ 
    
```

그림 4. 제곱-나눗셈법
Fig. 4. Square and Divide Method

[7,1023]의 $b = 2^{k+1} - 1$ 에 대해 곱셈 횟수를 계산한 결과는 표 2에 제시되어 있다. $b = 127 = 1111111_2$ 의 경우, 이진법은 12회, 2-ary는 11회, 3-ary는 14회의 곱셈을 수행한다. 반면에, 제곱-나눗셈법을 적용하면 $a^{127} = a^{128-1}$ 로 $a, a^2 \rightarrow a^4 \rightarrow a^8 \rightarrow a^{16} \rightarrow a^{32} \rightarrow a^{64} \rightarrow a^{128}$ 의 7회 제곱과 $a^{128} \rightarrow a^{127}$ 의 1회 나눗셈으로 8회로 단축시킬 수 있다. $a^{127} \pmod m = a^{128-1} \pmod m = \frac{a^{128} \pmod m}{a^1 \pmod m}$ 으로 계산된다.

표 2. $b = 2^k - 1$ 값에 따른 알고리즘의 곱셈 수행횟수 비교
Table 2. Compare of the Multiply Number of Algorithms in accordance with $b = 2^k - 1$

b	수행횟수				
	이진법	2-ary	3-ary	제곱-나눗셈법	
7	111	2*2=4	2+3*1=5	-	3+1=4
15	1111	2*3=6	2+3*1=5	6+4*1=10	4+1=5
31	11111	2*4=8	2+3*2=8	6+4*1=10	5+1=6
63	111111	2*5=10	2+3*2=8	6+4*1=10	6+1=7
127	1111111	2*6=12	2+3*3=11	6+4*2=14	7+1=8
255	11111111	2*7=14	2+3*3=11	6+4*2=14	8+1=9
511	111111111	2*8=16	2+3*4=14	6+4*2=14	9+1=10
1023	1111111111	2*9=18	2+3*4=14	6+4*3=18	10+1=11

$4^b \pmod{497}$ 에 대해 표 2의 b 값인 $a^{2^k-1} \pmod m$ 을 계산한 결과는 표 3에 제시되어 있다.

또한, $32 \leq b \leq 64$ 의 범위에 대해 곱셈 수행횟수를 비교한 결과는 표 4에 제시되어 있다. 표에서 제곱-나눗셈법은 b 가 2^{k+1} 에 근접한 값 또는 $b < 2^k + 2^{k-1}$ 에 근접한 경우 사전처리를 하는 일반적인 d -ary법에 비해 곱셈 횟수를 감소시킬 수 있음을 보여주고 있다.

표 3. $a^{2^k-1} \pmod n$ 계산 결과

Table 3. The Result of $a^{2^k-1} \pmod n$

b	4^{2^k} 와 $4^{2^k-1} \pmod m$	방법	j
7	$\frac{4^7 \pmod{497} = 480}{4^8 \pmod{497} = 429}$	$(429 + 497 \times 3) / 4 = 480$	3
15	$\frac{4^{15} \pmod{497} = 162}{4^{16} \pmod{497} = 151}$	$(151 + 497 \times 1) / 4 = 162$	1
31	$\frac{4^{31} \pmod{497} = 109}{4^{32} \pmod{497} = 436}$	$(436 + 497 \times 1) / 4 = 109$	1
63	$\frac{4^{63} \pmod{497} = 309}{4^{64} \pmod{497} = 242}$	$(242 + 497 \times 2) / 4 = 309$	2
127	$\frac{4^{127} \pmod{497} = 228}{4^{128} \pmod{497} = 415}$	$(415 + 497 \times 1) / 4 = 228$	1
255	$\frac{4^{255} \pmod{497} = 190}{4^{256} \pmod{497} = 263}$	$(263 + 497 \times 1) / 4 = 190$	1
511	$\frac{4^{511} \pmod{497} = 270}{4^{512} \pmod{497} = 86}$	$(86 + 497 \times 2) / 4 = 270$	2
1023	$\frac{4^{1023} \pmod{497} = 358}{4^{1024} \pmod{497} = 438}$	$(438 + 497 \times 2) / 4 = 358$	2

표 4. $32 \leq b \leq 64$ 곱셈 수행횟수

Table 4. Trial Number of Multiply for $32 \leq b \leq 64$

b	2진수	수행횟수	
		이진법 (제곱-곱셈법)	곱셈-나눗셈법
32	100000	$a \rightarrow a^2 \rightarrow a^4 \rightarrow a^8 \rightarrow a^{16} \rightarrow a^{32}$ (5회)	
33	100001	32+1 (6회)	
34	100010	32+2 (6회)	
35	100011	32+2+1 (7회)	
36	100100	32+4 (6회)	
37	100101	32+4+1 (7회)	
38	100110	32+4+2 (7회)	
39	100111	32+4+2+1 (8회)	32+8-1 (7회)
40	101000	32+8 (6회)	
41	101001	32+8+1 (7회)	
42	101010	32+8+2 (7회)	
43	101011	32+8+2+1 (8회)	
44	101100	32+8+4 (7회)	
45	101101	32+8+4+1 (8회)	
46	101110	32+8+4+2 (8회)	32+16-2 (7회)
47	101111	32+8+4+2+1 (9회)	32+16-1 (7회)

48	110000	32+16 (6회)	
49	110001	32+16+1 (7회)	
50	110010	32+16+2 (7회)	
51	110011	32+16+2+1 (8회)	
52	110100	32+16+4 (7회)	
53	110101	32+16+4+1 (8회)	
54	110110	32+16+4+2 (8회)	64-(8+2) (8회)
55	110111	32+16+4+2+1 (9회)	64-(8+1) (8회)
56	111000	32+16+8 (7회)	64-8 (7회)
57	111001	32+16+8+1 (8회)	64-(8-1) (8회)
58	111010	32+16+8+2 (8회)	64-(4+2) (8회)
59	111011	32+16+8+2+1 (9회)	64-(4+1) (8회)
60	111100	32+16+8+4 (8회)	64-4 (7회)
61	111101	32+16+8+4+1 (9회)	64-(2+1) (8회)
62	111110	32+16+8+4+2 (9회)	64-2 (7회)
63	111111	32+16+8+4+2+1 (10회)	64-1 (7회)
64	100000 0	$a \rightarrow a^2 \rightarrow a^4 \rightarrow a^8 \rightarrow a^{16} \rightarrow a^{32} \rightarrow a^{64}$ (6회)	

기존의 $n-ary$, ($n \geq 3$)은 $2^n - 2$ 개의 비트 값을 사전에 구하는 방식이다. 반면에 사전처리를 하지 않고 알고리즘 수행 과정에서 얻어진 값들을 기준으로 $n-ary$ 에 곱셈 또는 나눗셈을 적용하는 방법을 고찰해 보자.

사전처리 없는 $3-ary$ 와 $4-ary$ 에 곱셈 또는 나눗셈 방법을 접목시켜 $32 \leq b \leq 64$ 의 수행횟수를 구한 결과는 표 5에 제시되어 있다. 사전처리 없는 $3-ary$ 와 $4-ary$ 는 51과 54에서는 제공-나눗셈보다 좋은 결과를 나타냈으며, 62와 63은 제공-나눗셈법이 가장 좋은 결과를 나타내었다. 따라서 대체적으로 2^{k+1} 에 근접한 값에 대해서는 제공-나눗셈법을, 나머지 값들에 대해서는 사전처리 없는 $n-ary$ 법에 제공-나눗셈법을 접목시킨 방법을 적용하는 것이 효율적임을 알 수 있다.

만약, $b = 761 = 1011111001_{(2)}$ 인 경우, 이진법은 $0 + 9 + 6 = 15$ 회, $2-ary$ 는 $2 + 3 * 4 = 14$ 회, $3-ary$ 는 $6 + 4 * 3 = 18$ 회의 곱셈을 수행한다. $2-ary$ 로 $10 - 11 - 11 - 10 - 01$ 로 분할하고, 사전처리 없는 $4-ary$ 로 $10 - 1111 - 1001$ 에 대해 제공-나눗셈을 처리한다고 가정하면 $a^2, a^4 \rightarrow a^8 \rightarrow a^{16} \rightarrow a^{32} \rightarrow (15 = 16 - 1) \rightarrow a^{47} \rightarrow a^{94} \rightarrow a^{188} \rightarrow a^{376} \rightarrow a^{752} \rightarrow (9 = 8 + 1) \rightarrow a^{761}$ 로 13회로 줄일 수 있다.

만약, $b = 3243679 = 1100010111111010011111_{(2)}$ 인 경우, 이진법은 35회, $2-ary$ 는 31회, $3-ary$ 는 34회의 곱셈을 수행한다. $3-ary$ 로 $1 - 100 - 010 - 111 - 111010 - 011 - 111$ 로 분할하고, 사전처리 없는 $6-ary$ 로 좌에서 우로 처리한다면 $1 - 100010 - 111111 - 010011 - 111$ 은 $a^2 \rightarrow a^4 \rightarrow a^8 \rightarrow a^{16} \rightarrow a^{32} \rightarrow a^{64} \rightarrow (34 = 32 + 2) \rightarrow$

$a^{98} \rightarrow a^{196} \rightarrow a^{392} \rightarrow a^{784} \rightarrow a^{1568} \rightarrow a^{3136} \rightarrow a^{6272} \rightarrow (63 = 64 - 1) \rightarrow a^{6335} \rightarrow a^{12670} \rightarrow a^{25340} \rightarrow a^{50680} \rightarrow a^{101360} \rightarrow a^{202720} \rightarrow a^{405440} \rightarrow (19 = 16 + 2 + 1) \rightarrow a^{405459} \rightarrow a^{810918} \rightarrow a^{1621836} \rightarrow a^{3243672} \rightarrow (7 = 8 - 1) \rightarrow a^{3243679}$ 로 30회로 줄일 수 있다. 여기에는 비트 값을 처리할 때 곱셈 또는 나눗셈을 적용하였다.

표 5. $32 \leq b \leq 64$ 의 사전처리 없는 $n-ary$ 수행횟수
Table 5. $n-ary$ Trial Number for $32 \leq b \leq 64$ without Preprocessing

b	2진수	수행횟수 ($3-ary/4-ary$)
32	100-000	2→4, 8→16→32 (5회)
	10-0000	2, 4→8→16→32 (5회)
33	100-001	2→4, 8→16→32→33(+1) (6회)
	10-0001	2, 4→8→16→32→33(+1) (6회)
34	100-010	2→4, 8→16→32→34(+2) (6회)
	10-0010	2, 4→8→16→32→34(+2) (6회)
35	100-011	2→4, 8→16→32→(3 = 2 + 1)→35 (7회)
	10-0011	2, 4→8→16→32→(3 = 2 + 1)→35 (7회)
36	100-100	2→4, 8→16→32→36(+4) (6회)
	10-0100	2, 4→8→16→32→36(+4) (6회)
37	100-101	2→4, 8→16→32→(5 = 4 + 1)→37 (7회)
	10-0101	2, 4→8→16→32→(5 = 4 + 1)→37 (7회)
38	100-110	2→4, 8→16→32→(6 = 4 + 2)→38 (7회)
	10-0110	2, 4→8→16→32→(6 = 4 + 2)→38 (7회)
39	100-111	2→4, 8→16→32→(7 = 8 - 1)→39 (7회)
	10-0111	2, 4→8→16→32→(7 = 8 - 1)→39 (7회)
40	101-000	2→4→5, 10→20→40 (6회)
	10-1000	2, 4→8→16→32→40(+8) (6회)
41	101-001	2→4→5, 10→20→40→41(+1) (7회)
	10-1001	2, 4→8→16→32→(9 = 8 + 1)→41 (7회)
42	101-010	2→4→5, 10→20→40→42(+2) (7회)
	10-1010	2, 4→8→16→32→(10 = 8 + 2)→42 (7회)
43	101-011	2→4→5, 10→20→40→(3 = 2 + 1)→43 (8회)
	10-1011	2, 4→8→16→32→(3 = 2 + 1)→(11 = 8 + 3)→43 (8회)
44	101-100	2→4→5, 10→20→40→44(+4) (7회)
	10-1100	2, 4→8→16→32→(12 = 8 + 4)→44 (7회)
45	101-101	2→4→5, 10→20→40→(5 = 4 + 1)→45 (8회)
	10-1101	2, 4→8→16→32→(5 = 4 + 1)→(13 = 8 + 5)→45 (8회)
46	101-110	2→4→5, 10→20→40→(6 = 4 + 2)→46 (8회)
	10-1110	2, 4→8→16→32→(14 = 16 - 2)→46 (7회)
47	101-111	2→4→5, 10→20→40→(7 = 5 + 2)→47 (8회)
	10-1111	2, 4→8→16→32→(15 = 16 - 1)→47 (7회)
48	110-000	2→3→6, 12→24→48 (6회)
	11-0000	2→3, 6→12→24→48 (6회)
49	110-001	2→3→6, 12→24→48→49(+1) (7회)
	11-0001	2→3, 6→12→24→48→49(+1) (7회)
50	110-010	2→3→6, 12→24→48→50(+2) (7회)
	11-0010	2→3, 6→12→24→48→50(+2) (7회)
51	110-011	2→3→6, 12→24→48→51(+3) (7회)
	11-0011	2→3, 6→12→24→48→51(+3) (7회)
52	110-100	2→3→6, 12→24→48→(4 = 3 + 1)→52 (8회)
	2→4→6, 12→24→48→52(+4) (7회)	
53	11-0100	2→3, 6→12→24→48→(4 = 3 + 1)→52 (8회)
	110-101	2→3→6, 12→24→48→(5 = 3 + 2)→53 (8회)
54	11-0101	2→3, 6→12→24→48→(5 = 3 + 2)→53 (8회)
	110-110	2→3→6→12→24→48→54(+6) (7회)
54	11-0110	2→3, 6→12→24→48→54(+6) (7회)

55	110-111	2→3→6,12→24→48→(7=6+1)→55 (8회)
	11-0111	2→3,6→12→24→48→(7=6+1)→55 (8회)
56	111-000	2→3→6→7,14→28→56 (7회)
	11-1000	2→3,6→12→24→48→(8=6+2)→56 (8회)
57	111-001	2→3→6→7,14→28→56→57(+1) (8회)
	11-1001	2→3,6→12→24→48→(9=6+3)→57 (8회)
58	111-010	2→3→6→7,14→28→56→58(+2) (8회)
	11-1010	2→3,6→12→24→48→(10=12-2)→58 (8회)
59	111-011	2→3→6→7,14→28→56→59(+3) (8회)
	11-1011	2→3,6→12→24→48→(11=12-1)→59 (8회)
60	111-100	2→4→6→7,14→28→56→60(+4) (8회)
	11-1100	2→3,6→12→24→48→60(+12) (7회)
61	111-101	2→4→6→7,14→28→56→(5=4+1)→61 (9회)
	11-1101	2→3,6→12→24→48→(13=12+1)→61 (8회)
62	111-110	2→3→6→7,14→28→56→62(+6) (8회)
	11-1110	2→3,6→12→24→48→(14=12+2)→62 (8회)
63	111-111	2→3→6→7,14→28→56→63(+7) (8회)
	11-1111	2→3,6→12→24→48→(15=12+3)→63 (8회)
64	1-000-000	1,2→4→8→16→32→64 (6회)
	100-0000	2→4,8→16→32→64 (6회)

만약, $b = 18206927 = 1000101011101000011001111_{(2)}$ 인 경우, 이진법은 $24+12=36$ 회, $2-ary$ 는 $2+12*2+8=34$ 회, $3-ary$ 는 $6+8*3+6=36$ 회의 곱셈을 수행한다. 이를 사전 처리 없는 $6-ary$ 로 적용하되 제곱을 6회 수행하고 비트 값에 대해 곱셈하는 과정을 적용하면 $1-000101-011101-000011-001111$ 에 대해 $6+6+6+6=24$ 회의 제곱과 $000101=5(4+1)$ 의 2회 곱셈, $011101=29=16+5+8$ 의 3회, $000011=3(2+1)$ 의 2회, $001111=15(16-1)$ 의 2회로 총 31회로 줄일 수 있다.

실험 데이터들에 제안된 제곱-나눗셈법과 사전처리 없는 $n-ary$ 법을 이진법, $2-ary$, $3-ary$ 법과 수행횟수를 비교한 결과는 표 6에 제시되어 있다. 제안된 제곱-나눗셈법과 사전처리 없는 $n-ary$ 법은 지수연산에 있어 가장 빠른 방법임을 알 수 있다.

표 6. 알고리즘 수행횟수 비교
Table 6. Compare of the Trial Number for Algorithms

b	이진법	$2-ary$	$3-ary$	제곱-나눗셈법	사전처리 없는 $n-ary$ 제곱-나눗셈법
7	4	5	-	4	-
15	6	5	10	5	-
31	8	8	10	6	-
63	10	8	10	7	-
127	12	11	14	8	-
255	14	11	14	9	-
511	16	14	14	10	-
761	15	14	18	-	13
1023	18	14	18	11	-
3243679	35	31	34	-	30
18206927	36	34	36	-	31

결론적으로, b 가 2^{k+1} 에 근접한 값인 경우 제곱-나눗셈법을 적용하는 것이 가장 효율적이며, 나머지 b 에 대해서는 곱

셈 또는 나눗셈법을 접목시킨 사전처리를 하지 않는 $n-ary$ 법이 보다 효율적임을 나타내고 있다.

IV. 결론

본 논문은 암호학의 암호 생성과 해독, 소수 판별법에 적용되는 모듈러 지수연산법을 제안하였다. 기존의 $n-ary$ 법은 $2^n - 2$ 개의 가능한 비트 값에 대한 곱셈 사전처리 후 제곱-곱셈법 방법을 적용하였다. 반면에 제안된 방법은 사전처리를 수행하지 않는 제곱-나눗셈법을 적용하였다. 이를 위해 모듈러 나눗셈을 수행하는 방법을 제안하였다. 또한, 사전처리를 하지 않는 $n-ary$ 법에 대해 곱셈 또는 나눗셈을 적용한 방법도 제안하였다.

제곱-나눗셈법은 b 가 2^{k+1} 에 근접한 값인 경우 가장 효율적이며, 나머지 값들에 대해서는 사전처리를 하지 않는 $n-ary$ 법에 제곱-나눗셈을 적용하는 방법이 사전처리를 하는 일반적인 $n-ary$ 법에 비해 효율적임을 보였다.

참고문헌

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms," 2nd Edition, McGraw-Hill Book Company, 2005.
- [2] M. Alfred, P. C. Oorschot, and S. A. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1996.
- [3] S. T. Klein, "Should One Always Use Repeated Squaring for Modular Exponentiation?," Information Processing Letters, Vol. 106, Issue. 6, pp. 232-237, 2008.
- [4] D. M. Gordon, "A Survey of Fast Exponentiation Methods," Journal of Algorithms, Vol. 27, No. 1, pp. 129-146, 1998.
- [5] P. Montgomery, "Modular Multiplication Without Trial Division," Math. Computation, Vol. 44, pp. 519-521, 1985.
- [6] G. Saldamli and C. K. Koc, "Spectral Modular Exponentiation," Proc. of the 18th IEEE Symposium on Computer Arithmetic, pp. 123-132, 2007.
- [7] V. Gopal, J. Guilford, E. Ozturk, W. Feghali, G.

- Wolrich, and M. Dixon, "Fast and Constant-Time Implementation of Modular Exponentiation," 28th International Symposium on Reliable Distributed Systems, Niagara Falls, New York, U.S.A., 2009.
- [8] L. Zhong, "Modular Exponentiation Algorithm Analysis for Energy Consumption and Performance," Technical Report CE-01-ZJL, Dept. of Electrical Engineering, Princeton University, 2001.
- [9] N. Nedjah and L. M. Mourelle, "Efficient Pre-Processing for Large Window-Based Modular Exponentiation Using Ant Colony," Informatica, Vol. 29, pp. 151-161, 2005.
- [10] F. R. Henriquez, "Modular Exponentiation," Arithmética Computacional, <http://delta.cs.cinvestav.mx/~francisco/arith/expo.pdf>.

저 자 소 개



이 상 운(Sang-Un, Lee)
 1983년 ~ 1987년 :
 한국항공대학교 항공전자공학과 (학사)
 1995년 ~ 1997년 :
 경상대학교 컴퓨터과학과 (석사)
 1998년 ~ 2001년 :
 경상대학교 컴퓨터과학과 (박사)
 2003.3 ~ 현 재 :
 강릉원주대학교 멀티미디어공학과 부교수
 관심분야 : 소프트웨어 프로젝트 관리,
 소프트웨어 개발 방법론,
 소프트웨어 신뢰성,
 신경망, 뉴로-피지,
 그래프 알고리즘
 e-mail : sulee@gwnu.ac.kr