The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 16, No. 2, pp.41-47, Apr. 30, 2016. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2016.16.2.41
JIIBC 2016-2-5

가변길이 분할 기법을 적용한 모듈러 지수연산법

Modular Exponentiation Using a Variable-Length Partition Method

이상운*

Sang-Un Lee*

요 약 암호학의 암호 생성과 해독의 곱셈 횟수는 대부분 $a^b \pmod m$ 모듈러 지수연산의 효율적 구현여부로 결정된다. 표준 모듈러 지수연산법으로는 $1-ary$법인 이진법이 있으며, $n-ary(2 \le n \le 6)$법이 많이 적용되고 있다. $n-ary(1 \le n \le 6)$법은 $b=b_k b_{k-1} \cdots b_1 b_{0(2)}$에 대해 R-L 방향으로 n비트로 고정된 분할을 하고, n회 제곱과 비트값 곱셈을 수행하는 제곱-곱셈법이다. 본 논문에서는 $b_{k-1} \cdots b_1 b_{0(2)}$에 대해 L-R 방향으로 가변길이로 분할하는 방법을 적용한다. 또한, 개변길이의 제곱과 곱셈 또는 나눗셈을 적용한다. 제안된 가변길이 분할법은 고정길이 분할법인 $n-ary$법에 비해 곱셈 수행횟수를 감소시킬 수 있었다.

Abstract The times of multiplication for encryption and decryption of cryptosystem is primarily determined by implementation efficiency of the modular exponentiation of $a^b \pmod m$. The most frequently used among standard modular exponentiation methods is a standard binary method, of which $n-ary(2 \le n \le 6)$ is most popular. The $n-ary(2 \le n \le 6)$ is a square-and-multiply method which partitions $b=b_k b_{k-1} \cdots b_1 b_{0(2)}$ into n fixed bits from right to left and squares n times and multiplies bit values. This paper proposes a variable-length partition algorithm that partitions $b_{k-1} \cdots b_1 b_{0(2)}$ from left to right. The proposed algorithm has proved to reduce the multiplication frequency of the fixed-length partition $n-ary$ method.

Key Words : Modular exponentiation, Binary method, $n-ary$ method, Fixed-length, Variable-length

Ⅰ. 서 론

$a^b \pmod m$ 은 a^b의 값을 m으로 나눈 나머지를 구하는 모듈러 지수연산 (modular exponentiation)이라 하며 암호학 분야에 널리 사용되고 있다.[1,2]

대표적인 모듈러 지수 연산법에는 $a^b \pmod m$ 의 $b=b_k b_{k-1} b_{k-2} \cdots b_1 b_{0_{(2)}}$를 R-L (right-to-left)로 n비트씩 분할 (2^n-진법)한 $n_l n_{l-1} \cdots n_1 n_0$에 대해 n_l 비트의 값을 초기치 a^l로 설정하고, n_{l-1}부터 n_0까지 n회 제곱 후 비트 값을 지수항 덧셈 (곱셈)을 수행하는 n항법 ($n-ary$ method)이 있다. $n-ary$법은 각 비트 값이 가질 수 있는 가능한 경우수인 2^n개 중에서 0,1을 제외한 2^n-2개를 사전 처리 (preprocessing)하여 저장한 후 비트 값 곱셈에 사용한다. 이외에도 몽고메리 감소법 (Montgomery reduction)과 추가사슬 (addition chain)법 등이 있다.[3-8]

$n-ary$법은 $b=b_k b_{k-1} \cdots b_1 b_{0(2)}$에 대해 항상 고정된 길이 n비트씩 분할한다. 이를 통신 분야에서 데이터

*정회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
접수일자 : 2015년 8월 28일, 수정완료 : 2016년 3월 2일
게재확정일자 : 2016년 4월 8일

Received: 28 August, 2015 / Revised: 2 March, 2016 / Accepted: 8 April, 2016
*Corresponding Author: sulee@gwnu.ac.kr
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

압축시 적용되는 기법인 고정길이 부호 기법 (fixed-length coding method)으로 설명될 수 있다. 데이터 압축의 효율성을 증대시키는 방법으로 가변길이 부호 기법 (variable-length coding method)을 일반적으로 적용하고 있다.[9]

본 논문은 $n-ary$의 곱셈 횟수를 감소시키는 방법으로 가변길이 부호 기법을 적용한다. 알고리즘의 차이점은 첫 번째로, $b=b_kb_{k-1}\cdots b_1b_{0(2)}$에 대해 $n-ary$법은 R-L로 n비트씩 일정하게 분할하는 기법으로 n-블록부호 (block code)라 할 수 있다. 반면에, 제안된 방법은 b_k를 제외한 $b_{k-1}b_{k-2}\cdots b_1b_0$에 대해 L-R (left-to-right) 가변길이로 비트를 분할하는 기법을 적용한다. 따라서 제안된 방법은 고정 길이 기법과 상반되는 개념이다. $n-ary$법을 "고정길이 분할법"이라 하고, 제안된 방법을 "가변길이 분할법"이라 하자. 두 번째로, 고정길이 분할법은 사전처리를 수행하는데 반해 가변길이 분할법은 사전처리를 하지 않고, 비트 값 계산은 알고리즘 수행과정에서 얻은 값을 활용한다. 세 번째로, 고정길이 분할법은 비트 값 계산을 항상 곱셈으로 얻는다. 반면에 가변길이 분할법에서는 곱셈 또는 나눗셈을 적용한다.

2장에서는 고정길이 분할법을 고찰한다. 3장에서는 가변길이 분할법을 제안하고, 고정길이 분할법과 가변길이 분할법의 수행 횟수를 비교하여 본다.

II. 고정길이 분할법

일반적으로 알려진 $n-ary$법은 그림 1과 같이 수행된다.[1,4] 그림 1의 이진법과 $n-ary$법을 통합하여 일반화시켜 표현하면 그림 2와 같이 수행되며, 그림 1 방법에 비해 곱셈을 1회 감소시킬 수 있다.

$b=1933=11110001101_{(2)}$인 경우의 이진법, $2-ary$법과 $3-ary$법의 계산 방법은 그림 3에 제시하였다. 이진법은 사전처리 횟수(0)+제곱횟수(10)+비트 값 곱셈횟수(6)=16회를 수행하였다. $2-ary$법은 2+5*2+4= 16회, $3-ary$법은 6+3*3+3=18회를 수행한다.

$b=127=1111111_{(2)}$인 경우, 이진법은 0+2+2+2+2+2+2=12회 곱셈을 수행한다. $2-ary$법은 $01\ 11\ 11\ 11=1333_{(4)}$에 대해 2+0+3+3+3=11회 수행한다. $3-ary$법은 $001\ 111\ 111=177_{(8)}$에 대해 6+4+4=14회를 수행한다. 결국, $b=127$인 경우 $2-ary$법이 최선의 방법임을 알 수 있다.

$a^b \pmod n$
Modular-Exponentiation (a,b,n)
  $c=1$, $b=b_kb_{k-1}\cdots b_1b_{0_{(2)}}$
  for $i=k$ down to 0 do
    $c=(c\times c) \bmod n$ /* square
    if $b_i=1$ then $c=(c\times a) \bmod n$
  return c

(a) Binary method

$a^b \pmod n, b=b_kb_{k-1}\cdots b_1b_{0_{(2)}}$
Preprocessing (사전 처리)
  $a_0=1$
  for $i=1$ to (2^k-1) do
    $a_i=(a_{i-1}\times a) \bmod n$
$c=1$
Modular-Exponentiation (a,b,n)
  for $i=n$ down to 0 do
    for $j=0$ to $k-1$
      $c=(c\times c) \bmod n$ /* power
    if $b_i\neq 0$ then $c=(c\times a_{b_i}) \bmod n$
  return c

(b) $n-ary(n>1)$ method

그림 1. 전형적인 $n-ary$ 지수연산법
Fig. 1. Typical $n-ary$ modular exponentiation

$a^b \pmod m, b=n_kn_{k-1}\cdots n_1n_{0(n)}$
Preprocessing (사전 처리)
  $a_1=a$
  if $n\geq 3$ then /* $1-ary$와 $2-ary$는 수행 않음.
    for $i=2$ to (2^n-1) do
      $a_i=(a_{i-1}\times a) \bmod m$
    end
  end
Modular-Exponentiation (a,b,m)
  $c=a_k$ /* n_k값의 a_k 값. 만약, $n_k=1$이면 a.
  for $i=k-1$ down to 0 do
    for $j=1$ to n do
      $c=(c\times c) \bmod m$
    end
    $c=(c\times a_i) \bmod m$ /* n_i값의 a_i 값.
  end
  return c

그림 2. 일반화된 $n-ary$ 지수연산법
Fig. 2. Generalized $n-ary$ modular exponentiation

구분		이진법		$2-ary$		$3-ary$	
		제곱	곱셈	제곱	곱셈	제곱	곱셈
사전 처리		–	–	10 11	$a\times a$ $a^2\times a$	010 011 100 101 110 111	$a\times a$ $a^2\times a$ $a^3\times a$ $a^4\times a$ $a^5\times a$ $a^6\times a$
b_{10}	1	–	a	–	a	–	a^3
b_9	1	$(a^1)^2$	a^{2+1}	$(a^1)^2$	a^{4+3}		
b_8	1	$(a^3)^2$	a^{6+1}	$(a^2)^2$		$(a^3)^2$ $(a^6)^2$ $(a^{12})^2$	a^{24+6}
b_7	1	$(a^7)^2$	a^{14+1}	$(a^7)^2$	a^{28+2}		
b_6	0	$(a^{15})^2$	–	$(a^{14})^2$			
b_5	0	$(a^{30})^2$	–	$(a^{30})^2$	–	$(a^{30})^2$ $(a^{60})^2$ $(a^{120})^2$	a^{240+1}
b_4	0	$(a^{60})^2$	–	$(a^{60})^2$			
b_3	1	$(a^{120})^2$	a^{240+1}	$(a^{120})^2$	a^{480+3}		
b_2	1	$(a^{241})^2$	a^{482+1}	$(a^{240})^2$		$(a^{241})^2$ $(a^{482})^2$ $(a^{964})^2$	a^{1928+5}
b_1	0	$(a^{483})^2$	–	$(a^{483})^2$	a^{1932+1}		
b_0	1	$(a^{966})^2$	a^{1932+1}	$(a^{966})^2$			

그림 3. a^{1933} 계산 $n-ary$법
Fig. 3. a^{1933} Computation using $n-ary$ method

$b=b_k b_{k-1}\cdots b_1 b_{0_{(2)}}$에 대해 $n-ary$법은 2^n-2회 사전처리, $n(\left\lceil \frac{k+1}{n} \right\rceil -1)$회 제곱과 $(\left\lceil \frac{k+1}{n} \right\rceil -1)(\frac{2^n-1}{2^n})$회 곱셈을 수행한다. 이 공식을 적용하면 이론적으로 최적인 $n-ary$법을 결정할 수 있다. RSA는 일반적으로 이진수 512, 1024 또는 2,048 비트를 적용한다. 따라서 $2\le k\le 2048$에 대해 계산한 결과 k 범위별 최적의 $n-ary$법은 그림 4에 제시하였다. 그러나 실제로는 $b_i=1$의 개수에 따라 최적의 $n-ary$법이 달라질 수 있다.

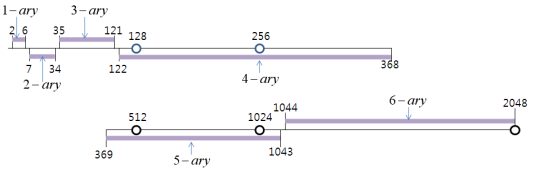

그림 4. 최적의 $n-ary$법의 이진자리수 k 범위
Fig. 4. k range of binary digits number for optimal $n-ary$ method

Ⅲ. 가변길이 분할법

모듈러 지수 연산법에서 가변길이로 분할하는 방법은 연구되지 않고 있다. 본 장에서 제안하는 가변길이 분할법은 다음의 2가지 기법을 추가적으로 접목시킨다.

(1) $n-ary$법은 2^n-2회의 곱셈 사전처리를 수행하는데 반해 가변길이 분할법은 사전 처리를 수행하지 않고 알고리즘 수행 과정에서 얻은 지수 값을 활용한다.

$b=48=110000_{(2)}$인 경우, 사전처리를 하지 않으면 $2\to4\to8\to16\to32\to(48=32+16)\to48$의 7회 곱셈을 수행한다. 만약, $2-ary$를 적용한다면, 사전에 $2\to3$을 구한 후 $1,2\to3(2+1)\to6\to12\to24\to48$을 수행하여 8회의 곱셈을 수행하는데 반해 사전 처리된 값을 곱셈에 전혀 사용하지 않는다.

(2) $n-ary$법은 비트 값에 대해 곱셈 (지수 항 덧셈)만을 수행한다. 반면에 가변길이 분할법은 곱셈 또는 나눗셈을 적용한다.

$b=127=1111111_{(2)}$에 대한 곱셈 수행 횟수는 이진법은 12회, $2-ary$법은 11회, $3-ary$법은 14회임을 2장에서 보였다. 만약, $a^{127}=a^{128-1}$로 모듈러 지수 나눗셈을 계산할 수 있다면 $a^2\to a^4\to a^8\to a^{16}\to a^{32}\to a^{64}\to a^{128}\to a^{127}$로 7회의 제곱과 1회의 나눗셈으로 8회로 단축시킬 수 있다. 모듈러 지수 나눗셈 $a^{(b-c)}(\mathrm{mod}\,m)=\frac{a^b(\mathrm{mod}\,m)}{a^c(\mathrm{mod}\,m)}$을 직접 계산하는 방법은 제안되지 않고 있다. 모듈러 지수 나눗셈은 n의 임의의 배수를 더한 값을 나누어 정수가 되어야 되기 때문에 본 장에서는 모듈러 나눗셈 $a^b(\mathrm{mod}\,m)=\frac{a^{2^{b_{k+1}}}(\mathrm{mod}\,m)}{a^{2^{b_{k+1}}-b}(\mathrm{mod}\,m)}$은 그림 5의 방법을 적용한다. 즉, $a^{127}(\mathrm{mod}\,m)=\frac{a^{128}(\mathrm{mod}\,m)}{a^1(\mathrm{mod}\,m)}$으로 계산될 수 있으며, 이 경우 j 값을 찾아야 한다.

while $j=1$ to $e=$정수
$e=\frac{a^{2^{b_{k+1}}}(\mathrm{mod}\,m)+(m\times j)}{a^{2^{b_{k+1}}-b}(\mathrm{mod}\,m)}$, $e=$정수
or
$e=\frac{a^{2^{b_k+b_{k-1}}}(\mathrm{mod}\,m)+(m\times j)}{a^{2^{b_k+b_{k-1}}-b}(\mathrm{mod}\,m)}$, $e=$정수
$j=j+1$
end

그림 5. 모듈러 나눗셈 방법
Fig. 5. Modular divide method

[7,1023]의 $b=2^{b_{k+1}}-1$에 대해 곱셈 횟수를 계산한 결과는 표 1에 제시되어 있다. $b=2^{b_{k+1}}-1$에 대해 값이 증가할수록 나눗셈법이 곱셈법에 비해 수행횟수를 크게 감소시킬 수 있음을 알 수 있다.

표 1. $b=2^k-1$값에 따른 수행횟수 비교
Table 1. Compare of trial number follow the $b=2^k-1$ value

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**

$4^b(\mathrm{mod}\,497)$에 대해 표 1의 b값인 $a^{2^k-1}(\mathrm{mod}\,m)$을 계산한 결과는 표 2에 제시되어 있다.

표 2. $a^{2^k-1}(\mathrm{mod}\,n)$ 계산 결과
Table 2. Computation results of $a^{2^k-1}(\mathrm{mod}\,n)$

b	4^{2^k}와 $4^{2^k-1}(\mathrm{mod}\,m)$	모듈러 나눗셈	j
7	$\frac{4^7(\mathrm{mod}\,497)=480}{4^8(\mathrm{mod}\,497)=429}$	$(429+497\times3)/4=480$	3
15	$\frac{4^{15}(\mathrm{mod}\,497)=162}{4^{16}(\mathrm{mod}\,497)=151}$	$(151+497\times1)/4=162$	1
31	$\frac{4^{31}(\mathrm{mod}\,497)=109}{4^{32}(\mathrm{mod}\,497)=436}$	$(436+497\times1)/4=109$	1
63	$\frac{4^{63}(\mathrm{mod}\,497)=309}{4^{64}(\mathrm{mod}\,497)=242}$	$(242+497\times2)/4=309$	2
127	$\frac{4^{127}(\mathrm{mod}\,497)=228}{4^{128}(\mathrm{mod}\,497)=415}$	$(415+497\times1)/4=228$	1
255	$\frac{4^{255}(\mathrm{mod}\,497)=190}{4^{256}(\mathrm{mod}\,497)=263}$	$(263+497\times1)/4=190$	1
511	$\frac{4^{511}(\mathrm{mod}\,497)=270}{4^{512}(\mathrm{mod}\,497)=86}$	$(86+497\times2)/4=270$	2
1023	$\frac{4^{1023}(\mathrm{mod}\,497)=358}{4^{1024}(\mathrm{mod}\,497)=438}$	$(438+497\times2)/4=358$	2

가변길이 분할법의 핵심은 b의 이진수에 대해 비트수를 가변적으로 분할하는 기준이다. $b=b_kb_{k-1}\cdots b_1b_{0_{(2)}}$에 대해 $b_k=1$이므로 초기치 a를 설정하고, $b_{k-1}\cdots b_1b_0$ 에 대해 L-R 방향으로 가급적 다음 기준에 의거 분할한다.

[기준 1] "0"또는 "1"만 존재하는 경우 분할하지 않는다. 예로, $b=1111111_{(2)}$인 경우, 111111로 $b=10000000_{(2)}$의 경우, 0000000으로 분할하지 않는다.

[기준 2] 1개의 "1"에 이어서 "0"연속된 이후 "1"이 1개만 존재하는 경우, "1"이 b_0이면 분할하지 않으며, b_0가 아니면 "0"까지 분할한다. 예로, $b=110001_{(2)}$인 경우, 10001로 분할하지 않는다. $b=11000100_{(2)}$인 경우, 1000-100으로 분할된다.

[기준 3] "1"이 2개 연속된 경우, 이어서 연속되는 "0"이 종료되는 비트까지 분할한다. 예로, $b=1110000100_{(2)}$인 경우 110000-100으로 분할된다. $b=11100101_{(2)}$인 경우, 1100-101로 분할된다.

[기준 4] "1"이 3개 이상 연속된 경우, "1"이 종료되는 비트까지 분할한다. 예로, $b=1111000100_{(2)}$인 경우 111-000-100으로 분할된다.

[기준 5] "0"이 2개 이상 연속되고 이어서 "1"이 1개만 존재하고 b_0이면 분할하지 않으며, b_0가 아니면 "0"이 끝나는 비트에서 분할한다. 예로, $b=1000000100_{(2)}$인 경우 000000-100 으로 분할한다. $b=10000001_{(2)}$인 경우 0000001로 분할하지 않는다.

[기준 6] "1010…"또는 "0101…"인 경우, 3비트씩 분할한다. 예로, $b=1110110100_{(2)}$인 경우 11-011-0100로 분할된다.

[기준 7]두 번째 이후의 분할 비트수는 오로지"0"만 존재하지 않을 경우, 첫 번째 분할의 비트수를 초과하지 않도록 재분할한다. 예로, $b=3243679=1100010111111\,0100111111_{(2)}$인 경우, 1000-10-111111-0100- 11111로 분할하였다면 첫 번째 분할의 길이 4비트를 초과하는 경우가 발생하므로 1000-10-111-111-0100-111-11로 재분할한다.

가변길이 분할법은 그림 6에 제시하였다.

Ⅳ. 적용 결과 및 분석

$32\le b\le 64$범위에 대해 이진법과 가변길이 분할법의 횟수를 비교한 결과는 표 3에 제시되어 있다.

$2^5 \le b \le 2^6$에 대해 이진법과 가변길이 분할법을 비교한 결과, $2^k+2^n, (k=5, n=1,2,\cdots,k)$는 항상 동일한 곱셈 횟수를 나타낸다. 반면에, $2^k+(2^{k-2}-1)=39$, $2^k+(2^{k-1}-1)=47$과 $2^k+(2^{k-1}-2)=46$에서 나눗셈을, $2^k+2^{k-1}+2^{k-2}+2^{k-3}-1=55$와 $2^k+2^{k-1}+2^{k-2}-1=59$에서는 곱셈을 적용하여 수행횟수를 감소시킬 수 있었다. 또한, $2^{k+1}-l, (l=1,2,3,4)$인 60, 61, 62와 63에서 나눗셈을 적용하여 수행횟수를 줄일 수 있었다.

$a^b \pmod n, b=(b_k b_{k-1}\cdots b_1 b_0)_2$
Modular-Exponentiation (a, b, m)
  $c=a$
  $b=b_{k-1}b_{k-2}\cdots b_1b_0$ 을 l개로 분할. /* 분할 기준 참조
  for $i=1$ to l do
    for $j=1$ to p_i do/* p_i: 각 분할의 비트 수
      $c=(c\times c) \pmod m$
    end
    if 비트 값 = 이전 $a^{b_1+b_2}$ then
      $c_a=(c_1\times c_2) \pmod m$ /* $c_i=a^{b_i} \pmod m$
      $c=(c\times c_a) \pmod m$
    else if 비트 값 = 2^k-1 then
      while $j=1$ to c_a =정수 do
        /* $c_k = 2^k \pmod m$
        $c_a=\dfrac{c_k+(m\times j)}{a} \pmod m$
        $j=j+1$
      end
      $c=(c\times c_a) \pmod m$
    end if
    end
  return c

그림 6. 가변길이 분할법
Fig. 6. Variable-length partition method

표 3. $32 \le b \le 64$ 곱셈 수행횟수
Table 3. Trial number of multiplication for $32 \le b \le 64$

b	2진수	수행횟수 (이진법/가변길이 분할법)
32	1-00000	2→4→8→16→32 (5회)
	1-00000	2→4→8→16→32 (5회)
33	1-00001	2→4→8→16→32→33 (6회)
	1-00001	2→4→8→16→32→33(+1) (6회)
34	1-00010	2→4→8→16→17→34 (6회)
	1-000-10	2→4→8→16→32→34(+2) (6회)
35	1-00011	2→4→8→16→17→34→35 (7회)
	1-000-11	2→4→8→16→32→(3=2+1)→35 (7회)
36	1-00100	2→4→8→9→18→36 (6회)
	1-00-100	2→4→8→16→32→36(+4) (6회)
37	1-00101	2→4→8→9→18→36→37 (7회)
	1-00-101	2→4→8→16→32→(5=4+1)→37 (7회)
38	1-00110	2→4→8→9→18→19→38 (7회)
	1-00-11-0	2→4→8→16→(3=2+1)→19→38 (7회)
39	1-00111	2→4→8→9→18→19→38→39 (8회)
	1-00-111	2→4→8→16→32→(7=8−1)→39 **(7회)**
40	1-01000	2→4→5→10→20→40 (6회)
	1-01-000	2→4→5(+1)→10→20→40 (6회)
41	1-01001	2→4→5→10→20→40→41 (7회)
	1-01-001	2→4→5(+1)→10→20→40→41(+1) (7회)
42	1-01010	2→4→5→10→20→21→42 (7회)
	1-01-010	2→4→5(+1)→10→20→40→42(+2) (7회)
43	1-01011	2→4→5→10→20→21→42→43 (8회)
	1-01-011	2→4→5(+1)→10→20→40→(3=2+1)→43 (8회)
44	1-01100	2→4→5→10→11→22→44 (7회)
	1-011-00	2→4→8→(3=2+1)→11→22→44 (7회)
45	1-01101	2→4→5→10→11→22→44→45 (8회)
	1-011-01	2→4→8→(3=2+1)→11→22→44→45(+1) (8회)
46	1-01110	2→4→5→10→11→22→23→46 (8회)
	1-0111-0	2→4→8→16→(7=8−1)→23→46 **(7회)**
47	1-01111	2→4→5→10→11→22→23→46→47 (9회)
	1-01111	2→4→8→16→32→(15=16−1)→47 **(7회)**
48	1-10000	2→3→6→12→24→48 (6회)
	1-10000	2→4→8→16→32→48(+16) (6회)
49	1-10001	2→3→6→12→24→48→49 (7회)
	1-10001	2→4→8→16→32→(17=16+1)→49 (7회)
50	1-10010	2→3→6→12→24→25→50 (7회)
	1-100-10	2→4→8→12(+4)→24→48→50(+2) (7회)
51	1-10011	2→3→6→12→24→25→50→51 (8회)
	1-100-11	2→4→8→12(+4)→24→48→(3=2+1)→51 (8회)
52	1-10100	2→3→6→12→13→26→52 (7회)
	1-10-100	2→4→6(+2)→12→24→48→52(+4) (7회)
53	1-10101	2→3→6→12→13→26→52→53 (8회)
	1-10-101	2→4→6(+2)→12→24→48→(5=4+1)→53 (8회)
54	1-10110	2→3→6→12→13→26→27→54 (8회)
	1-10-11-0	2→4→6(+2)→12→24→(3=2+1)→27→54 (8회)
55	1-10111	2→3→6→12→13→26→27→54→55 (9회)
	1-10-111	2→4→6(+2)→12→24→48→(7=6+1)→55 **(8회)**
56	1-11000	2→3→6→7→14→28→56 (7회)
	1-11-000	2→4→(3=2+1)→7→14→28→56 (7회)
57	1-11001	2→3→6→7→14→28→56→57 (8회)
	1-11-001	2→4→(3=2+1)→7→14→28→56→57(+1) (8회)
58	1-11010	2→3→6→7→14→28→29→58 (8회)
	1-11-01-0	2→4→(3=2+1)→7→14→28→29(+1)→58 (8회)
59	1-11011	2→3→6→7→14→28→29→58→59 (9회)
	1-11-011	2→4→(3=2+1)→7→14→28→56→59(+3) **(8회)**
60	1-11100	2→3→6→7→14→15→30→60 (8회)
	1-111-00	2→4→8→(7=8−1)→15→30→60 **(7회)**
61	1-11101	2→3→6→7→14→15→30→60→61 (9회)

	1-111-01	2→4→8→(7 = 8 − 1)→15→30→60→61(+1) **(8회)**
62	1-11110	2→3→6→7→14→15→30→31→62 (9회)
	1-1111-0	2→4→8→16→(15 = 16 − 1)→31→62 **(7회)**
63	1-11111	2→3→6→7→14→15→30→31→62→63 (10회)
	1-11111	2→4→8→16→32→64→(63 = 61 − 1)→63 **(8회)**
64	1-000000	2→4→8→16→32→64 (6회)
	1-000000	2→4→8→16→32→64 (6회)

추가적으로 $64 \le b \le 128$에 대해 몇 개의 숫자에 대해 이진법, $2-ary$법과 가변길이 분할법을 적용한 사례는 표 4에 제시되어 있다.

표 4에서는 이진 자리수가 7개 이상으로 이론상으로는 $2-ary$법이 이진법에 비해 효율적인 것으로 알려져 있지만 127을 제외하면 실제로는 이진법이 더 효율적임을 알 수 있다. 또한, 가변길이 분할법은 표 3에서의 결과와 동일하게 모든 b값에 대해 $n-ary(1 \le n)$법과 동일하거나 보다 좋은 결과를 나타내고 있다.

표 4. $64 \le b \le 128$ **곱셈 수행횟수**
Table 4. Trial number of multiplication for $64 \le b \le 128$

b	2진수	수행횟수 (이진법/$2-ary$/가변길이 분할법)
	1-0-0-0-0-0-0	1,2→4→8→16→32→64 (6회)
64	01-00-00-00	2→3, 1,2→4→8→16→32→64 (8회)
	000000	2→4→8→16→32→64 (6회)
	1-0-0-0-0-0-1	1,2→4→8→16→32→64→65(+1) (7회)
65	01-00-00-01	2→3,1,2→4→8→16→32→64→65(+1) (9회)
	000001	2→4→8→16→32→64→65(+1) (7회)
	1-0-0-0-1-1-1	1,2→4→8→16→17(+1)→34→35(+1)→70→71(+1) (9회)
71	01-00-01-11	2→3,1,2→4→8→16→17(+1)→34→68→71(+3) (10회)
	000-111	2→4→8→16→32→64→(7 = 8 − 1)→71 (8회)
	1-0-1-1-1-1-1	1,2→4→5(+1)→10→11(+1)→22→23(+1)→46→47(+1)→94→95(+1) (11회)
95	01-01-11-11	2→3,1,2→4→5(+1)→10→20→23(+3)→46→92→95(+3) (11회)
	011111	2→4→8→16→32→64→(31 = 32 − 1)→95 (8회)
	1-1-0-0-0-0-0	1,2→3(+1)→6→12→24→48→96 (7회)
96	01-10-00-00	2→3,1,2→4→6(+2)→12→24→48→96 (9회)
	100000	2→4→8→16→32→64→96(+32) (7회)
	1-1-1-1-1-1-1	1,2→3(+1)→6→7(+1)→14→15(+1)→30→31(+1)→62→63(+1)→126→127(+1) (12회)
127	01-11-11-11	2→3,1,2→4→7(+3)→14→28→31(+3)→62→124→127(+3) (11회)
	1111111	2→4→8→16→32→64→(63 = 64 − 1)→127 (8회)
	1-0-0-0-0-0-0	1,2→4→8→16→32→64→128 (7회)
128	10-00-00-00	2→3,2,4→8→16→32→64→128 (8회)
	0000000	2→4→8→16→32→64→128 (7회)

만약, $b = 761 = 1011111001_{(2)}$인 경우, 이진법은 0+9+6=15회, $2-ary$는 2+3*4=14회, $3-ary$는 6+4*3=18회의 곱셈을 수행한다. 가변길이 분할법을 적용하여 011111-001로 분할하면 $a^2 \to a^4 \to a^8 \to a^{16} \to a^{32} \to a^{64} \to (31 = 32-1) \to a^{95} \to a^{190} \to a^{380} \to a^{760} \to a^{761}(+1)$로 12회로 줄일 수 있다.

만약, $b = 3243679 = 1100010111111010011111_{(2)}$인 경우, 이진법은 35회, $2-ary$는 31회, $3-ary$는 34회의 곱셈을 수행한다. 가변길이 분할법을 적용하여 1000-10-111111-0100-11111을 다시 1000-10-111-111-0100-111-11로 재분할하면 $a^2 \to a^4 \to a^8 \to a^{16} \to a^{24}(+8) \to a^{48} \to a^{96} \to a^{98}(+2) \to a^{196} \to a^{392} \to a^{784} \to (7 = 8-1) \to a^{791} \to a^{1582} \to a^{3164} \to a^{6328} \to a^{6335}(+7) \to a^{12670} \to a^{25340} \to a^{50680} \to a^{101360} \to a^{101364}(+4) \to a^{202728} \to a^{405456} \to a^{810912} \to a^{810919}(+7) \to a^{1621838} \to a^{3243676} \to (3 = 2+1) \to a^{3243679}$로 30회로 줄일 수 있다.

$b = 18206927 = 1000101011101000011001111_{(2)}$인 경우, 이진법은 24+12=36회, $2-ary$는 2+12*2+8=34회, $3-ary$는 6+8*3+6=36회의 곱셈을 수행한다. 가변길이 분할법을 적용하여 0001-010-111-010000-11-00-1111로 분할하였다면 0001-010-111-01-0000-11-00-1111로 재분할한다. 이는 $a^2 \to a^4 \to a^8 \to a^{16} \to a^{17}(+1) \to a^{34} \to a^{68} \to a^{136} \to a^{138}(+2) \to a^{276} \to a^{552} \to a^{1104} \to (7 = 8-1) \to a^{1111} \to a^{2222} \to a^{4444} \to a^{4445}(+1) \to a^{8890} \to a^{17780} \to a^{35560} \to a^{71120} \to a^{142240} \to \to a^{284480} \to a^{284483}(+3) \to a^{568966} \to a^{1137932} \to a^{2275864} \to a^{4551728} \to \to a^{9103456} \to a^{18206912} \to (15 = 16-1) \to a^{18206927}$로 32회로 줄일 수 있다.

V. 결론

본 논문은 가변길이 분할법을 적용한 모듈러 지수연산법을 제안하였다. $n-ary$법은 이진수 $b = b_k b_{k-1} \cdots b_1 b_{0(2)}$에 대해 R-L로 n비트씩 동일한 비트로 분할하는 고정길이 분할법이라 할 수 있다.

제안된 가변길이 분할법은 $b_{k-1} \cdots b_1 b_{0(2)}$에 대해 L-R로 가변길이 분할 기준을 적용하여 분할하는 방법을 적용하였다. 또한, $n-ary$법의 $2^n - 2$개의 가능한 비트 값에 대한 곱셈 사전처리를 수행하지 않으며, 비트값에 대해 곱셈 또는 나눗셈을 적용하는 특징이 있다.

가변길이 분할법은 고정길이의 $n-ary$법에 비해 수행횟수를 보다 감소시키는 효과를 얻었다.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms," 2nd Edition, McGrew-Hill Book Company, ISBN: 9780262033848, 2005.

[2] M. Alfred, P. C. Oorschot, AND S. A. Vanstone, "Handbook of Applied Cryptography," CRC Press, ISBN: 0-8493-8523-7, 1996.

[3] S. T. Klein, "Should One Always Use Repeated Squaring for Modular Exponentiation?," Information Processing Letters, Vol. 106, No. 6, pp. 232-237, doi:10.1016/j.ipl.2007.11.016, Jun. 2008.

[4] D. M. Gordon, "A Survey of Fast Exponentiation Methods," Journal of Algorithms, Vol. 27, No. 1, pp. 129-146, doi:10.1006/jagm.1997. 0913, Apr. 1998.

[5] P. Montgomery, "Modular Multiplication Without Trial Division," Math. Computation, Vol. 44, pp. 519 - 521, doi: 10.1090/S0025-5718-1985 -0777282-X, Apr. 1985.

[6] G. Saldamli and C. K. Koc, "Spectral Modular Exponentiation," Proc. of the 18th IEEE Symposium on Computer Arithmetic, pp. 123-132, doi: 10.1109/ARITH.2007.34, Jun. 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., Sep. 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] J. Berstel, D. Perrin, and C. Reutenauer, 'Codes and Automata," Cambridge University Press, ISBN-13: 978-0521888318, 2009.

저자 소개

이 상 운(정회원)

- 1987년 : 한국항공대학교 항공전자공학과 (학사)
- 1997년 : 경상대학교 컴퓨터과학과 (석사)
- 2001년 : 경상대학교 컴퓨터과학과 (박사)
- 2003년 : 강원도립대학 컴퓨터응용과 전임강사
- 2004년~2007년 2월 : 국립 원주대학 여성교양과 조교수
- 2007년 3월~2015년 3월 : 강릉원주대학교 멀티미디어공학과 부교수
- 2015년 4월~현재 : 강릉원주대학교 멀티미디어공학과 정교수

<주관심분야 :소프트웨어 프로젝트 관리, 개발 방법론, 분석과 설계 방법론, 시험 및 품질보증, 소프트웨어 신뢰성, 그래프 알고리즘>

- E-Mail : sulee@gwnu.ac.kr