Journal of The Korea Society of Computer and Information
Vol. 18, No. 8, August 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.8.087

최단 보폭-최장 보폭 이산대수 알고리즘의 변형

이 상 운*

Modified Baby-Step Giant-Step Algorithm for Discrete Logarithm

Sang-Un, Lee *

요 약

최단 보폭-최장 보폭 알고리즘은 n을 $m = \lceil \sqrt{n} \rceil$ 개의 원소를 가진 m개의 블록으로 분할하고 첫 번째 블록의 m개에 대해 $a^x \pmod n$ 값을 저장한다. 다음으로 m개의 블록에 대한 $\bmod n$을 계산하여 첫 번째 블록의 원소 값을 검색하여 일치하는 블록을 찾는 방법이다. 본 논문에서는 첫 번째로, $a^{\phi(n)/2} \equiv 1 \pmod n$과 $a^x \pmod n \equiv a^{\phi(n)+x} \pmod n$의 특징을 적용하여 m개의 원소를 가진 $\lceil m/2 \rceil$ 개의 블록으로 분할하는 방법을 적용하여 최장 보폭의 수행 횟수를 50% 감소시켰다. 두 번째로, $\lceil m/2 \rceil$ 개의 최단 보폭을 먼저 수행하여 저장하고, 첫 번째 블록의 m개 원소를 수행하는 최단 보폭을 수행하는 방법으로 최단 보폭-최장 보폭 알고리즘을 역으로 수행하는 방법을 제안하였다. 이 알고리즘은 최단 보폭-최장 보폭 알고리즘의 m개 저장과 검색을 $\lceil m/2 \rceil$ 개로 50% 감소시키는 특징이 있다.

▸ Keywords : 이산대수, 이산누승, 곱셈차수, 오일러 파이 함수, 최단 보폭-최장 보폭

Abstract

A baby-step giant-step algorithm divides n by m blocks that possess $m = \lceil \sqrt{n} \rceil$ elements, and subsequently computes and stores $a^x \pmod n$ for m elements in the 1st block. It then calculates $\bmod n$ for m blocks and identifies each of them with those in the 1st block of an identical elemental value. This paper firstly proposes a modified baby-step giant-step algorithm that divides $\lceil m/2 \rceil$ blocks with m elements applying $a^{\phi(n)/2} \equiv 1 \pmod n$ and $a^x \pmod n \equiv a^{\phi(n)+x} \pmod n$ principles. This results in a 50% decrease in the process of the giant-step. It then suggests a reverse baby-step giant step algorithm that performs and saves $\lceil m/2 \rceil$ blocks firstly and computes $a^x \pmod n$ for m elements. The proposed algorithm is found to successfully halve the

•제1저자 : 이상운
•투고일 : 2013. 5. 14, 심사일 : 2013. 5. 23, 게재확정일 : 2013. 6. 13
* 강릉원주대학교 멀티미디어공학과 (Dept. of Multimedia Eng., Gangneung-Wonju National University)

memory and search time of the baby-step giant step algorithm.

▸ Keywords : Discrete logarithm, Discrete exponentiation, Multiplicative order, Euler's totient function, Baby-step giant-step

I. 서 론

유암호의 공개키 n은 합성수 (composite number)로 유사하거나 동일한 길이의 2개 소수 (prime number) p,q를 선택하여 곱한 값인 $n=pq$로 결정된다. 메시지 m은 $e<n$, $\gcd(e,\phi(n))=1$을 선택하여 암호화된 코드 $c=m^e \pmod n$로 변환되어 전송되며, 수신자는 개인키 d를 이용하여 메시지 $m=c^d \pmod n$로 해독한다. $ed \pmod{\phi(n)}=1$을 선택하며, (e,n)은 공개된다. 여기서 $\phi(n)$은 n과 서로소인 원소의 개수로 오일러의 totient (phi) 함수이며, $d=e^{-1} \pmod{\phi(n)}=e^{\phi(n)-1} \pmod{\phi(n)}$이다. 수신된 암호를 해독하기 위해 $m=c^d \pmod n$에서 m,c,n이 주어졌을 때 d를 직접 구하는 방법으로 이 문제를 이산로그 또는 이산대수 (discrete logarithm)라 한다[1].

이산대수 알고리즘으로는 최단 보폭-최장 보폭, Pollard의 캥거루와 rho, Pohlig-Hellman, Index calcurus, Number Field Sieves, Function Field Sieve 등이 있다. 그러나 어떠한 알고리즘도 다항시간 내에 해를 구하지 못하고 있다[2].

대표적인 최단 보폭-최장 보폭 알고리즘은 n을 k개의 원소를 갖고 있는 k개의 블록 (block)으로 분할하고, 최단 보폭 단계에서는 첫 번째 블록의 k개에 대해 모듈러 연산을 수행한다. 다음으로 블록 결정 기준이 되는 $a^k \pmod n$을 계산하고 $(a^k)^{-1} \pmod n$을 구한다. 마지막으로 최장 보폭 단계에서는 k개의 블록을 대상으로 모듈러 연산을 수행하여 해당되는 블록과 블록 내의 위치를 결정한다. 따라서 $2k+1$회의 모듈러 연산을 수행한다[3-5].

본 논문은 최단 보폭-최장 보폭 알고리즘의 모듈러 연산 횟수를 $(k-1)+\lceil k/2 \rceil$로 줄일 수 있는 범위 축소 방법과 최단 보폭-최장 보폭 알고리즘을 역으로 수행하는 최장 보폭-최단 보폭 알고리즘을 제안한다. 2장에서는 암호 해독 방법에 관한 관련 연구를 고찰한다. 3장에서는 최단 보폭-최장 보폭 알고리즘을 효율적으로 수행하는 변형된 형태를 제안한다. 4장에서는 결론을 맺는다.

II. 최단 보폭-최장 보폭 알고리즘

컴퓨터에서 사용하는 암호는 비대칭키인 RSA (Rivest, Shamir, and Adleman)와 대칭키인 DES (data encryption standard)를 혼합하여 사용하고 있다. DES는 빠른 속도를 갖는 반면에 키를 해독하기가 쉽다. 반면에, RSA는 해독이 불가한 반면에 암호와 복호 시간이 과다하게 소요된다. 이러한 두 암호체계의 장점만을 채택하여 DES는 평문을 암호화 하고 복호화하는데 사용되며, DES 키는 RSA로 사용자에게 전달한다. 따라서, RSA의 암호체계로 전송되는 DES 키를 해독하기 위해서는 이산대수 문제를 보다 쉽게 풀어야 한다.

이산대수 문제를 푸는 가장 일반적인 방법인 최단 보폭-최장 보폭 (Baby-step Giant-step, BSGS) 알고리즘은 그림 1에 제시되어 있다. 이 알고리즘은 $a^b \equiv c \pmod n$에서 b를 찾기 위해 주어진 수 n을 m개의 원소를 갖는 m개의 블록 $(n=m \times m)$으로 분할하고 어느 블록의 몇 번째 순서에 존재하는지를 결정한다[4-5].

입력 : a,c,n, 출력 : b (j번째 블록의 i번째 위치)
$a^b \equiv c \pmod n$, $\gcd(a,n)=1$. /* $m=c^d \pmod n$
$m=\lceil \sqrt{n} \rceil$ /* 블록 개수 결정
for $i=0$ to $m-1$ /* Baby-step (첫 번째 블록의 모든 원소들에 대한 값을 구함)
$c_i=a^i \pmod n$ 계산, (i,c_i) 저장.
end
$d=a^m \pmod n$에 대해 $e=(a^m)^{-1} \pmod n$ 계산
/* 블록 찾기 기준
$dx+ny=1$로 계산 /* 유클리드 알고리즘 적용
또는 $dx \pmod n=1$로 계산
for $j=0$ to $m-1$ /* Giant-step (블록과 블록 내 위치 결정)
$c_j=c \times d^j \pmod n$ 계산
if $c_j=c_i$ then $b=mj+i$
else if $c_j \neq c_i$ then continue.
end

그림 1. 최단 보폭-최장 보폭 알고리즘
Fig. 1. Baby-step Giant-step Algorithm

최단 보폭 단계에서는 첫 번째 블록의 모든 원소들에 대해 $c_i = a^i \pmod n$을 계산하여 (i, c_i)를 저장한다. 여기서 i는 나중에 해당 블록내의 순서를 결정한다. 다음으로 $a^m \pmod n$을 계산하여 $d = (a^m)^{-1} \pmod n$으로 블록을 찾는 기준을 결정한다. 마지막으로 최장 보폭 단계에서는 $c_j = c \times d^j \pmod n$을 계산하여 $c_j = c_i$이면 $b = mj + i$로 b가 j번째 블록의 i번째 위치라고 결정한다.

그림 1에서 $d = (a^m)^{-1} \pmod n$ 역-모듈러 곱셈 (modular multiplicative inverse) 계산은 $ax + by = \gcd(a,b)$에 의거 $aa^{-1} \equiv 1 \pmod n$, $ax + ny = 1$로 $x = a^{-1}$이 된다. 이는 일반적으로 확장된 유클리드 알고리즘 (extended Euclidean algorithm)을 적용하여 구하고 있다. 즉, 최단 보폭-최장 보폭 알고리즘은 최단 보폭에서 계산된 $\sqrt{n}$ 개의 데이터를 저장해야 하며, 최장 보폭에서 $\sqrt{n}$ 회를 수행하기 때문에 매우 큰 자리수인 경우 메모리 저장용량 문제가 발생한다. 따라서, 최단 보폭에서의 메모리 저장용량 감소와 더불어 최장 보폭에서의 수행횟수를 감소시킬 수 있는 알고리즘이 절실히 요구되고 있다.

최단 보폭-최장 보폭 알고리즘에 대해 Coron et al.[6]의 해밍거리를 이용한 변형된 방법과 Oh et al.[7]의 타원곡선에 대한 변형된 연구 결과가 존재한다. 그러나, 이산대수 문제를 보다 빠르게 푸는 획기적인 방법에 대해서는 연구가 진행되지 않고 있으며, 모듈러 지수연산법에 대해서는 Lee[8]가 있다.

III. 최단 보폭-최장 보폭 알고리즘의 변형

본 장에서는 최단 보폭-최장 보폭 알고리즘에 비해 메모리 저장 용량 최소화와 연산횟수 단축을 할 수 있는 이산대수 방법을 제안한다.

3.1 범위 축소 최단 보폭-최장 보폭 알고리즘

첫 번째로, 최단 보폭-최장 보폭 알고리즘이 왜 m개의 블록으로 나누어 $2m+1$회의 모듈러 연산을 수행하는가? m개의 블록이 모듈러 연산을 최소화 시키는 최적 분할 방법인가? 이에 대한 해답은 표 1의 모듈러 연산 횟수로 살펴볼 수 있다.

$n = 33$은 $\sqrt{n} = 5.7446$으로 $m = 6$으로 결정된다. 따라서 블록을 m개로 결정할 경우 모듈러 연산 횟수가 최단이 된다. 따라서 이 기준을 적용한다.

표 1. $n = 33$ 인 경우 블록 크기 별 모듈러 연산횟수
Table 1. Number of Computation for each block size of $n = 33$

블록 원소 (최단 보폭)	블록 결정	블록 수 (최장 보폭)	모듈러 연산 총 수행횟수
1	1	33	35
2	1	17	20
3	1	11	15
4	1	9	14
5	1	**7**	**13**
6	1	**6**	**13**
7	1	**5**	**13**
8	1	5	14
9	1	4	14
10	1	4	15
11	1	3	15
12	1	3	16
13	1	3	17
14	1	3	18
15	1	3	19
16	1	3	20
17	1	2	20
18	1	2	21
19	1	2	22
20	1	2	23

두 번째로, 최단 보폭-최장 보폭 알고리즘은 최단 보폭 단계에서 $0 \le i < m-1$에 대해 $c_i = a^i \pmod n$을 계산한다. 그러나 $a^0 \pmod n = 1$, $a^1 \pmod n = a$로 모듈러 연산을 수행할 필요가 없다. 따라서 $0 \le i \le m-1$을 $2 \le i \le m-1$로 범위를 축소시킨다.

세 번째로, 최장 보폭 단계에서는 먼저, $0 \le j \le m-1$로 m개의 블록에 대해 $c_j = c \times d^j \pmod n$을 계산한다. 그러나 첫 번째 블록은 최단 보폭에서 검증되었기 때문에 수행할 필요가 없어 $1 \le j \le m-1$로 축소될 수 있다. 다음으로 최대값 $m-1$에 관한 사항이다. 이는 n이 소수인 경우 최대 $\phi(p) = p-1$, $a^{p-1} \equiv 1 \pmod p$로 m번째 블록까지 확인해야 한다. 그러나 보안에 사용되는 공개키 n은 합성수이며, 합성수는 $a^{\phi(n)/2} \equiv 1 \pmod n$이 최대값이 되며, 이를 만족하는 j는 $\lceil m/2 \rceil$이다. 따라서 $1 \le j \le m-1$이 다시 $1 \le j \le \lceil m/2 \rceil$로 축소된다.

합성수가 $a^{\phi(n)/2} \equiv 1 \pmod n$이 최대값이 되는지 여부는 곱셈차수 (multiplicative order)로 판단할 수 있다. 곱셈차수는 Z_n의 순환 군 (cyclic group)을 결정하는데 적용된다. 예로, $n = 143 (11 \times 13)$, $\phi(n) = 120$, $m = 12$에

대해 곱셈차수를 구한 결과는 표 2와 같다.

표 2. $a^b \equiv 1 (\mathrm{mod}\, 143), 1 \le a \le n-1$인 경우 b
Table 2. b value for $a^b \equiv 1 (\mathrm{mod}\, 143), 1 \le a \le n-1$

a	b	a	b	a	b	a	b
0	0	**36**	**30**	**72**	**60**	108	30
1	1	**37**	**60**	**73**	**20**	109	4
2	60	**38**	**10**	**74**	**30**	110	-
3	15	**39**	**-**	**75**	**30**	111	12
4	30	**40**	**10**	**76**	**12**	112	20
5	20	**41**	**60**	**77**	**-**	113	30
6	60	**42**	**15**	**78**	**-**	114	30
7	60	**43**	**6**	**79**	**10**	115	60
8	20	**44**	**-**	**80**	**60**	116	10
9	15	**45**	**12**	**81**	**15**	117	-
10	6	**46**	**60**	**82**	**30**	118	10
11	-	**47**	**20**	**83**	**20**	119	60
12	**2**	**48**	**15**	84	60	120	6
13	**-**	**49**	**30**	85	60	121	-
14	**5**	**50**	**60**	86	20	122	4
15	**60**	**51**	**10**	87	6	123	60
16	**15**	**52**	**-**	88	-	124	60
17	**30**	**53**	**5**	89	12	125	20
18	**20**	**54**	**12**	90	10	126	15
19	**60**	**55**	**-**	91	-	127	30
20	**60**	**56**	**6**	92	5	128	60
21	**4**	**57**	**20**	93	60	129	10
22	**-**	**58**	**60**	94	30	130	-
23	**6**	**59**	**60**	95	30	131	2
24	**60**	**60**	**20**	96	20	132	-
25	**10**	**61**	**30**	97	60	133	3
26	**-**	**62**	**30**	98	12	134	30
27	**5**	**63**	**60**	99	-	135	20
28	**60**	**64**	**10**	100	3	136	60
29	**30**	**65**	**-**	101	30	137	60
30	**30**	**66**	**-**	102	60	138	20
31	**20**	**67**	**12**	103	10	139	30
32	**12**	**68**	**30**	104	-	140	30
33	**-**	**69**	**30**	105	10	141	60
34	**4**	**70**	**20**	106	60	142	-1, +1
35	**30**	**71**	**60**	107	30		

$a^b \equiv 1 (\mathrm{mod}\, n)$인 곱셈차수는 $a^g (\mathrm{mod}\, n) = a^{kb+g} (\mathrm{mod}\, n), (k = 1,2,\cdots)$를 의미하므로 곱셈차수까지만 모듈러 연산 결과를 알 수 있으면 이산대수의 값을 구할 수 있다. 곱셈차수는 $\phi(n)$의 약수에 대한 진부분집합 값에서 발생한다. $\phi(n) = 120$의 경우 $\phi(n) = 2^s d, (d = odd)$, $0 \le r \le s$인 120→60→30→15에 대해 $120 = 2 \times 60, 3 \times 40, 4 \times 30, 5 \times 24, 6 \times 20, 10 \times 12$, $60 = 2 \times 30, 3 \times 20, 4 \times 15, 5 \times 12, 6 \times 10$, $30 = 2 \times 15, 3 \times 10, 5 \times 6$, $15 = 3 \times 5$의 모든 약수들의 진부분 집합 {2,3,4,5,6,10,} 20,24,30,40,60} 중에서 곱셈차수가 발생함을 알 수 있다. 또한, 최대값은 $\phi(n) = 120$이 아닌 약수의 진부분 집합 최대값인 $\phi(n)/2 = 60$으로 이 조건을 만족하는 j의 최대 블록은 $\lceil m/2 \rceil$이다. 또한, 최단 보폭단계에서 $m = 12$, $2 \le i \le 12$를 수행하면 $\gcd(a,n) = 1$인 118개중 38개가 이 영역에 속해 있어 32.20%는 최장 보폭 단계를 수행하지 않아도 됨을 알 수 있다. 또한, 사전에 $\gcd(a,n) = 1$ 여부를 계산하지 않는 장점이 있으며, $\gcd(a,n) \neq 1$인 $p+q-1$개는 최단 보폭 단계의 $2 \le i \le m$ 내에서 $c_i = a$가 발생하기 때문에 b의 존재 여부를 간단히 구할 수 있다. 또 다른 특징은 소인수 p,q의 배수 kp, kq에 대해 $kp \pm 1$, $kq \pm 1$에서는 대부분 곱셈차수가 $2 \le i \le m$, m과 $2m$으로 나타나며, $\phi(n)/2$은 그 외의 부분 (소수 포함)에서 발생

입력 : a,c,n
출력 : b (k번째 블록의 i번째 위치)
$a^b \equiv c (\mathrm{mod}\, n)$
$b = \infty$
if $c = 1$ then $b = 0$, 알고리즘 종료
else if $c = a$ then $b = 1$, 알고리즘 종료
else $m = \lceil \sqrt{n} \rceil$ /* 블록 개수 결정
  for $i = 2$ to m /* Baby-step
    $c_i = a^i (\mathrm{mod}\, n)$ 계산
    if $c_i = c$ then $b = i$, 알고리즘 종료
    else if $c_i = a$ then 이산대수 없음, 알고리즘 종료
    else if $c_i = 1, b = \infty$ then 이산대수 없음, 알고리즘 종료
    else if $c_i \neq c, i < m, c_i \neq a$ then (i, c_i)저장
    else if $c_i \neq c, i = m$ then
      $d = a^m (\mathrm{mod}\, n)$
  end
  $d = a^m (\mathrm{mod}\, n)$
  for $j = 1,2,3,\cdots$
    if $jn+1 (\mathrm{mod}\, d) = 0$ then
      $e = (jn+1)/d$
    else $j = j+1$, continue
  end
  /* e : 블록 찾기 기준
  for $k = 1$ to $\left\lceil \frac{m}{2} \right\rceil$ /* Giant-step
    $c_k = c \times e^k (\mathrm{mod}\, n)$ 계산
    if $c_k = c_i$ then $b = mk+i$, 알고리즘 종료
    else continue
  end

그림 2. 범위 축소 최단 보폭-최장 보폭 알고리즘
Fig. 2. Reduced Range Baby-step Giant-step Algorithm

한다. 이와 같이 변형된 알고리즘은 그림 2에 제시되어 있으며 "범위 축소 최단 보폭-최장 보폭(reduced range BSGS, RBSGS) 알고리즘"이라 하자.

$n=143, m=12$인 경우, $15^b \equiv 124 \pmod n$의 $b=59$를 구하여 보자. 최단 보폭-최장 보폭 알고리즘은 최단보폭단계에서 $15^0 \pmod n$부터 $15^{11} \pmod n$의 12개를 계산하여 저장한다. 다음으로, 블록탐색 기준을 얻기 위해 $15^{12} \pmod n = 27$을 계산하고, $27x \pmod n = 1$인 $x=53$을 얻어 54회를 수행한다. 최장보폭단계에서는 $c_j = c \times d^j \pmod n$에서 $j=4$, $i=11$인 $15^{11} \pmod n = 59$를 얻어 4회 수행되고 $b=12\times4+11=59$를 구하였다. 결국, 12+54+4=70회를 수행하였다. 반면에 법위 축소 최단 보폭-최장 보폭 알고리즘은 최단보폭단계에서 $15^2 \pmod n$부터 $15^{11} \pmod n$인 10개를 계산하고, 12개를 저장한다. 블록탐색기준을 얻기 위해 $d=15^{12} \pmod n = 27$을 계산하고, $jn+1 \pmod d = 0$을 계산한 결과 $j=10$에서 $e=1431/27=53$을 얻어 11회를 수행한다. 최장보폭단계에서는 4회를 수행하였다. 결국, 제안된 방법은 10+11+4=25회로 수행횟수를 획기적으로 감소시킬 수 있었다.

3.2 최장 보폭-최단 보폭 알고리즘

본 절에서는 최단 보폭-최장 보폭 알고리즘을 역으로 수행하는 최장 보폭-최단 보폭 (Giant-step Baby-step, GSBS)방법으로 그림 3을 제안한다. 최단 보폭-최장 보폭 알고리즘은 m회의 최단 보폭으로 진행한 후 최장 보폭을 하는 방법인데 반해, 그림 3은 $\lceil m/2 \rceil - 1$회의 최장 보폭을 진행한 후 최단 보폭을 수행하는 방법이다. 왜냐하면 $\phi(n)=(p-1)(q-1)=pq-(p+q-1)=n-(p+q-1)$이 되며, $p+q-1>2\lfloor \sqrt{n} \rfloor$으로 $\phi(n)<n-2\lfloor \sqrt{n} \rfloor$이다. 또한, $\gcd(a,n)=1$에 대해 $a^{\phi(n)} \equiv 1 \pmod n, a^{\phi(n)/2} \equiv 1 \pmod n$이 성립하므로 $\lceil m/2 \rceil - 1$개의 블록 내에 $\phi(n)$이 존재하므로 이 블록들만 탐색하면 된다. 따라서 최단 보폭-최장 보폭 알고리즘의 m개의 (i, c_i)를 저장하는 방식을 최장 보폭-최단 보폭 알고리즘은 $\lceil m/2 \rceil - 1$개의 (j, c_j)를 저장하는 것으로 50%를 축소시키는 효과를 얻는다. 또한, 최단 보폭 단계에서 만약, $[2, m-1]$의 초반에 위치하면 수행 횟수를 보다 단축시킬 수도 있다.

$n=307, a=43, m=18, 43^b \equiv 140 \pmod{307}$에 서 $b=87$을 비교하여 보자. 최단 보폭-최장 보폭 알고리즘은 (0,1), (1,43), (2,7), (3,301), (4,49), (5,265), (6,36

입력 : a, c, n
출력 : b (k번째 블록의 i번째 위치)
$a^b \equiv c \pmod n$
$b = \infty$
if $c=1$ then $b=0$, 알고리즘 종료
else if $c=a$ then $b=1$, 알고리즘 종료
else $m = \lceil \sqrt{n} \rceil$ /* 블록 수
$d = a^m \pmod n$ 계산.
for $j=1,2,3,\cdots$
if $jn+1 \pmod d = 0$ then
$e=(jn+1)/d$ /* 블록 찾기 기준
else $j=j+1$, continue
end
for $k=1$ to $\left\lceil \frac{m}{2} \right\rceil - 1$ /* Giant-step
$c_k = c \times e^k \pmod n$ 계산
if $c_k = a$ then $i=1, b=mk+i$,
알고리즘 종료
else if $c_k \neq a$ then (k, c_k) 저장
end
for $i=2$ to $m-1$
$c_i = a^i \pmod n$ 계산
if $c_i = c_k$ then $b=mk+i$, exit
else if $c_i = c$ then $b=i$
else continue
end

그림 3. 최장 보폭-최단 보폭 알고리즘
Fig. 3. Giant-step Baby-step Algorithm

), (7,13), (8,252), (9,91), (10,229), (11,23), (12,68), (13,161), (14,169), (15,206), (16,262), (17,214)의 18개를 저장한다. 블록탐색 기준을 얻기 위해 $(43^{18})^{-1} \pmod{307} = 115$로 116회 계산한다. 최장보폭단계에서는 $c_j = 140 \times (115)^j \pmod{307}$에서 $j=4, c_j=206$을 구하고 이 때 $i=15$를 검색하였다. 따라서 $b=18\times4+15=87$을 얻으며, 18+1+115+15=149회의 모듈러 연산을 수행한다. 반면에, 최장 보폭-최단 보폭 알고리즘은 $d=43^{18} \pmod n$을 이진법 지수연산을 수행하면 $18=10010_{(2)}$으로 제곱 4회, 곱셈 1회로 5회 계산된다. 다음으로, $(307\times112+1) \pmod{299}=0$, $(307\times112+1)/299=115$를 얻어 112회 수행되었다. 최장보폭단계에서는 $c_k = 140 \times (115)^k \pmod{307}, (1 \le k \le 8)$을 계산하여 (1,136), (2,290), (3,14), (4,206), (5,51), (6,32), (7,303), (8,154)을 저장한다. 최단 보폭단계에서는 (2,7)부터 시작하여 (15,206)까지 14회 수행되어 $c_i = c_k = 206$을 얻었으며,

$k=4$, $i=15$이다. 따라서 모듈러 연산 횟수는 5+112+8+14=139회이다. 이는 최단 보폭-최장 보폭 알고리즘에 비해 10회를 감소시키는 결과를 얻는다. 또한, 최단 보폭 단계의 m개 데이터 저장에 비해 $\lceil m/2 \rceil - 1$개로 축소시켜 저장 및 검색하는 장점이 있어 보다 효율적이다. 본 예제는 $[0, m-1]$에서 $(m-1)/2 < i$인 경우이다.

$i < (m-1)/2$에 존재하는 경우를 고찰해 보자. $n=143$, $m=12$인 경우 $50^{49} \equiv 24 \pmod{143}$을 구하면, 최단 보폭-최장 보폭 알고리즘은 (0,0),(1,50), (2,69), (3,18),(4,42),(5,98),(6,38), (7,41), (8,48), (9,112),(10,23), (11,6)을 저장하고, (12,14)에 대해 $(14)^{-1} \equiv 92 \pmod{307}$로 92회 수행되었으며, $24 \times 92^{j} \pmod{307}$로 $j=4, c_j=50, i=1$을 얻어 $b=12\times4+1=49$를 얻는다. 이 경우 모듈로 수행횟수는 12+1+92+4=109회이다. 반면에, 최장 보폭-최단 보폭 알고리즘은 (1,63), (2,76), (3,128),(4,50),(5,24)을 계산하는 과정에서 $k=4, c_k=50=a$으로 $b=12\times4+1=49$를 얻는다. 단지, $50^{12} \pmod{n}$, $12=1100_{(2)}$를 계산하는데 3회의 제곱과 1회의 곱셈이 수행되었다. 따라서 4+4=8회의 모듈로 연산만을 수행하여 결과를 얻을 수 있다.

실험 방법은 임의의 값들에 대해 BSGS, RBSGS와 GSBS의 메모리 저장 데이터 건수와 연산횟수를 비교분석하였다. 본 실험 사례들에 대한 알고리즘 성능을 비교한 결과는 표 3에 제시되어 있다.

RBSGS 알고리즘은 BSGS 알고리즘에 비해 메모리 저장 용량을 차이가 없으며, 단지 연산횟수를 62% 줄일 수 있었다. GSBS 알고리즘은 BSGS 알고리즘에 비해 평균적으로 메모리 저장용량은 57%를, 연산횟수는 50%를 감소시켰다.

표 3. 알고리즘 성능 비교
Table 3. Comparison of Algorithm Performance

n	a	c	b	BSGS		RBSGS	
				저장	연산횟수	저장	연산횟수
143	14	124	59	12개	70회	12개	25회
n	a	c	b	BSGS		GSBS	
				저장	연산횟수	저장	연산횟수
307	43	140	87	18개	149회	8개	139회
143	50	24	49	12개	109회	5개	8회

IV. 결론

본 논문은 $\phi(n)$을 기준으로 $a^{x} \pmod{n} \equiv a^{\phi(n)+x} \pmod{n}$과 $a^{\phi(n)} \equiv 1 \pmod{n}$과 $a^{\phi(n)/2} \equiv 1 \pmod{n}$의 특징을 적용하여 변형된 최단 보폭-최장 보폭 알고리즘을 제안하였다.

최단 보폭-최장 보폭 알고리즘은 n을 $m = \lceil \sqrt{n} \rceil$개의 원소를 가진 $m = \lceil \sqrt{n} \rceil$개의 블록으로 분할하고 첫 번째 블록의 m개에 대해 $a^{x} \pmod{n}$값을 저장한다. 다음으로 m개의 블록에 대한 $\bmod n$을 계산하여 첫 번째 블록의 원소 값을 검색하여 일치하는 블록을 찾는 방법이다.

변형된 최단 보폭-최장 보폭 알고리즘은 2가지 형태를 제안하였다. 첫 번째로, $a^{\phi(n)/2} \equiv 1 \pmod{n}$과 $a^{x} \pmod{n} \equiv a^{\phi(n)+x} \pmod{n}$의 특징을 적용하여 m개의 원소를 가진 $\lceil m/2 \rceil$개의 블록으로 분할하는 방법을 적용하여 최장 보폭의 수행 횟수를 50% 감소시켰다. 두 번째로, $\lceil m/2 \rceil$개의 최단 보폭을 먼저 수행하여 저장하고, 첫 번째 블록의 m개 원소를 수행하는 최단 보폭을 수행하는 방법으로 역 최단 보폭-최장 보폭 알고리즘이다. 최단 보폭-최장 보폭 알고리즘의 m개 $a^{x} \pmod{n}$값을 저장하고 검색하는데 반해, $\lceil m/2 \rceil$개로 메모리 저장 용량 및 탐색시간을 50% 감소시키는 효과를 얻었다.

참고문헌

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms," 2nd Ed., MIT Press and McGraw-Hill. pp. 887-896, 2001.

[2] Wikipedia, "Discrete Logarithm," http://en.wikipedia.org/wiki/Discrete_logarithm, 2013.

[3] D. R. Stinson, "Cryptography: Theory and Practice," 3rd ed., London, CRC Press, 2006.

[4] A. Stein and E. Teske, "Optimized Baby step-Giant step Methods," Journal of the Ramanujan Mathematical Society, Vol. 20, No. 1, pp. 1-32, Jan. 2005.

[5] D. C. Terr, "A modification of Shanks' Baby-step Giant-step algorithm," Mathematics of Computation, Vol. 69, No. 230, pp. 767-773, Apr. 2000.

[6] J. S. Coron, D. Lefranc, and G. Poupard, "A new baby-step giant-step algorithm and some applications to cryptanalysis," 7th International Workshop, Vol. 3659, pp. 47-60, Aug. 2005.

[7] B. K. Oh, K. C. Ha, and J. H. Oh, "An Improved Baby-Step-Giant-Step Method For Certain Elliptic Curves," Journal of Applied Mathematics

& Computing, Vol. 20, No. 1-2, pp. 485-489, 2006.

[8] S. U. Lee, "Square-and-Divide Modular Exponentiation," Journal of Korean Society of Computer Information, Vol. 18, No. 4, pp. 123-129, Apr. 2013.

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-퍼지,
그래프 알고리즘
e-mail : sulee@gwnu.ac.kr