Journal of The Korea Society of Computer and Information
Vol. 20 No. 8, pp. 29-33, August 2015

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.8.029

Multiple Parallel-Pollard's Rho Discrete Logarithm Algorithm

Sang-Un Lee *

Abstract

This paper proposes a discrete logarithm algorithm that remarkably reduces the execution time of Pollard's Rho algorithm. Pollard's Rho algorithm computes congruence or collision of $\alpha^a\beta^b \equiv \alpha^A\beta^B \pmod p$ from the initial value $a=b=0$, only to derive γ from $(a+b\gamma)=(A+B\gamma), \gamma(B-b)=(a-A)$. The basic Pollard's Rho algorithm computes $x_i=(x_{i-1})^2, \alpha x_{i-1}, \beta x_{i-1}$ given $\alpha^a\beta^b \equiv x \pmod p$, and the general algorithm computes $x_i=(x_{i-1})^2$, Mx_{i-1}, Nx_{i-1} for randomly selected $M=\alpha^m, N=\beta^n$. This paper proposes 4-model Pollard Rho algorithm that seeks $\beta_\gamma=\alpha^\gamma, \beta_{\gamma'}=\alpha^{(p-1)/2+\gamma}$, and $\beta_{\gamma^{-1}}=\alpha^{(p-1)-\gamma})$ from $m=n=\lceil \sqrt{n} \rceil$, $(a,b)=(0,0),(1,1)$. The proposed algorithm has proven to improve the performance of the $(0,0)-$ basic Pollard's Rho algorithm by 71.70% .

▸ Keyword : discrete logarithm, Euler's totient function, Pollard Rho algorithm

I. Introduction

암호는 대칭암호인 비밀키와 비대칭 암호인 공개키로 구분된다. 대표적인 비대칭 암호인 RSA의 공개키 n은 합성수(composite number)로 2개 소수 p,q를 선택하여 $n=p\times q$ 로 쉽게 계산될 수 있다. 비대칭암호의 공개키 n으로부터 역으로 p,q를 구하는 소인수분해하기 어렵다는 수학의 난제에 기반하고 있다[1,2]

대표적인 대칭암호인 DES (Data Encryption Standard) 또는 AES (Advanced Encryption Standard)는 소수 (prime number)를 사용하며, $\alpha^\gamma \equiv \beta \pmod p$에서 α,β,p가 주어졌을 때 비밀 키 γ을 구하는 이산대수 (discrete logarithm)의 수학적 난제에 기반하고 있다[2].

본 논문에서는 $\alpha^\gamma \equiv \beta \pmod p$에서 γ을 구하는 이산대수 알고리즘을 제안한다. 이산대수 알고리즘으로는 아기걸음-거인걸음 (baby-step giant-step), Pollard의 캥거루와 Rho (ρ), Pohlig-Hellman, Index calculus, Number Field Sieves, Function Field Sieve 등이 있다. 그러나 어떠한 알고리즘도 빠른 시간 내에 해를 구하지 못하고 있다[2].

1971년에 제안된 Shank의 아기걸음-거인걸음 알고리즘은 거인걸음의 보폭 $m=\lceil \sqrt{n} \rceil$개의 데이터를 저장해야 하는 단점을 갖고 있으며, 수행 복잡도는 $O(\sqrt{n})$이다[3]. 반면에, 1978년에 Pollard가 Rho 알고리즘을 제안하였으며, 이 알고리즘은 아기걸음-거인걸음의 $m=\lceil \sqrt{n} \rceil$개의 데이터 저장 공간 필요성을 개선한 장점이 있는 반면, 수행 복잡도는 $O(\sqrt{n})$이다[4,5]. 이후에는 대부분 Pollard Rho 알고리즘에 대한 개선 연구가 수행되었으며, 아기걸음-거인걸음에 대해서는 연구가 활발히 이루어지지 않았고, 이로 인해 1990년대 이후로 이산대수 분야에서는 큰 진전이 없는 상황이다[4-7]. 최근들어 Lee[8]은 아기걸음-거인걸음 알고리즘을 아기걸음-성인걸음 방법으로 보다 단축시키는 방법을 제안하였다. 그럼에도 불구하고, 지금까지는 Pollard 알고리즘이 대칭키의 암호를 해독하는데 가장 효율적인 방법으로 알려져 있다.

Pollard Rho 알고리즘은 거북이는 기본 속도로, 토끼는 거북이의 2배 속도로 달려가면서 $\alpha^a\beta^b \equiv \alpha^A\beta^B$의 합동이 되는 값을 몇 번째 스텝에서 얻는 가로 결정된다. 그러나 본 논문에서는 Pollard Rho 알고리즘을 다양한 방법으로 변형시키면 Pollard Rho의 수행횟수 이전이라도 충돌지점을 빨리 찾을 수 있다는 점에 착안하여 Pollard Rho 알고리즘의 기본형을 확장

• First Author: Sang-Un Lee, Corresponding Author: Sang-Un Lee
*Sang-Un Lee (sulee@gwnu.ac.kr), Dept. of Multimedia Engineering, Gangneung-Wonju National University
• Received: 2015. 03. 25, Revised: 2015. 05. 17, Accepted: 2015. 06. 12.
• This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the CPRC(Communication Policy Research Center) support program (IITP-2015-R0880-15-1007) supervised by the IITP(Institute for Information & communication Technology Promotion)

한 다양한 일반형 알고리즘을 제안한다. 제안된 알고리즘은 초기치 $(a,b)=(0,0),(1,1)$과 $\beta_\gamma=\alpha^\gamma$, $\beta_{\gamma'}=\alpha^{(p-1)/2+\gamma}, \beta_{\gamma^{-1}}=\alpha^{(p-1)-\gamma}$을 적용하여 $m=n=\lceil\sqrt{n}\rceil$, $M=\alpha^m, N=\beta^n$ 에 대한 $x_i=(x_{i-1}^2, Mx_{i-1}, Nx_{i-1})$과 $x_i=(x_{i-1}^2, \alpha Mx_{i-1}, \beta Nx_{i-1})$의 충돌을 찾는 일반형 알고리즘이다. 2장에서는 이산대수 알고리즘을 고찰한다. 3장에서는 일반형 Pollard Rho 알고리즘을 제안하고, 성능을 검증하여 본다

II. 이산대수법

$\alpha^\gamma \equiv \beta \pmod p$에서 γ을 찾는 아기걸음-거인걸음 알고리즘은 그림 1에 제시되어 있다. 이 알고리즘은 p를 $m=\lceil\sqrt{p}\rceil$ 보폭으로 m 걸음으로 분할하고, 아기걸음 단계에서는 첫 번째 거인보폭 m개에 대해 보폭 1인 아기걸음으로 $\alpha^i \pmod p$, $i=[0,m-1]$의 모듈러 지수연산을 수행한다. 다음으로 거인걸음을 역으로 걷기 위해 $\alpha^m \pmod p$의 역함수 $\alpha^{-m} \pmod p$을 유클리드 알고리즘으로 구한다. 마지막으로 거인걸음 단계에서는 보폭 $m=-m$으로 j번째 걸음에서 i번째 값과 일치하면, $\gamma=jm+i$로 결정한다[6,7].

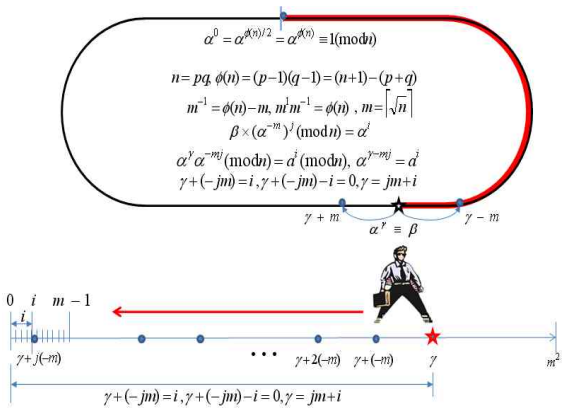

Fig. 1. Baby-step Giant-step Algorithm

아기걸음-거인걸음 알고리즘의 문제점은 아기걸음단계에서 구한 $m=\lceil\sqrt{p}\rceil$개의 모듈로 값을 계산 및 저장하고 탐색하는데 과다한 시간과 메모리를 필요로 한다는 점이다. p가 큰 수이면 이 데이터를 모듈로 계산과 저장 및 탐색에는 현실적으로 불가능하다고 할 수 있다. 이러한 데이터 저장 문제점을 해결한 알고리즘이 Pollard Rho 알고리즘이다.

Pollard의 Rho 알고리즘은 $\alpha^\gamma \equiv \beta \pmod n$에서 γ를 구하기 위해 식 (1)을 찾으며, $\alpha^a\beta^b \equiv \alpha^A\beta^B$의 합동 (congruence) 또는 충돌 (collision)을 구하기 위해 그림 2와 같이 거북이는 정상속도로, 토끼는 거북이의 2배 속도로 달려가면서 토끼와 거북이의 값이 같으면 종료하는 방식으로 수행된다[4,5]. 이 방법을 기본형이라 하자. 여기서 x_i와 X_i는 식 (2)와 같이 계산된다.

$$\alpha^a\beta^b \equiv \alpha^A\beta^B,\ (B-b)\gamma=(a-A) \qquad (1)$$
$$\alpha^a(\alpha^\gamma)^b \equiv \alpha^A(\alpha^\gamma)\beta^B, \alpha^{a+\gamma b} \equiv \alpha^{A+\gamma B},$$
$$(a+\gamma b)=(A+\gamma B), \gamma(B-b)=(a-A)$$

```
입력 : α, β, n, 출력 : γ
α^γ ≡ β (mod n)
int  main(void) {
     x_0 = 1, a_0 = 0, b_0 = 0, X_0 = 1, A_0 = 0, B_0 = 0
     for  (i = 1; i < n; ++i) {
          xab(x, a, b);
          XAB(X, A, B); XAB(X, A, B);
          if  (x_i = X_i) BREAK;
     }
     return 0;
}
void xab(x, a, b) {
     x_{i-1} (mod 3) = k
     if  k = 0 then x_i = x_{i-1}^2 (mod n), a_i = 2a_{i-1}
         (mod |G|), b_i = 2b_{i-1} (mod |G|)
     else if k = 1 then x_i = αx_{i-1} (mod n), a_i = a_{i-1}
         + 1 (mod |G|), b_i = b_{i-1}
     else if k = 2 then x_i = βx_{i-1} (mod n), a_i = a_{i-1},
         b_i = b_{i-1} + 1 (mod |G|)
}
void XAB(X, A, B) {
     X_{i-1} (mod 3) = k
     if  k = 0 then X_i = X_{i-1}^2 (mod n), A_i = 2A_{i-1}
         (mod |G|), B_i = 2B_{i-1} (mod |G|)
     else if k = 1 then X_i = αX_{i-1} (mod n), A_i = A_{i-1}
         + 1 (mod |G|), B_i = B_{i-1}
     else if k = 2 then X_i = βX_{i-1} (mod n), A_i = A_{i-1},
         B_i = B_{i-1} + 1 (mod |G|)
}
```

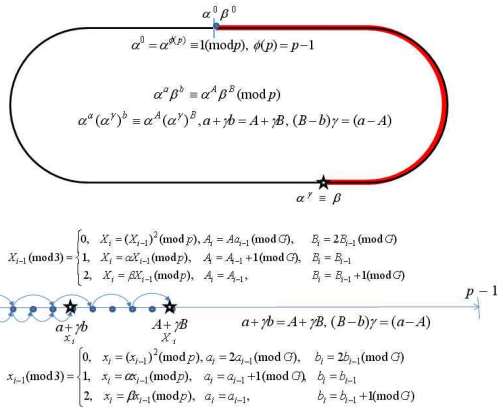

Fig. 2. Pollard's Rho Basic Algorithm

$$f_p(x)=\begin{cases} x_{i-1}^2 \pmod p, & a_i=2a_{i-1} \pmod G, b_i=2b_{i-1} \pmod G \\ \alpha x_{i-1} \pmod p, & a_i=a_{i-1}+1 \pmod G, b_i=b_{i-1} \\ \beta x_{i-1} \pmod p, & a_i=a_{i-1}, b_i=b_{i-1}+1 \pmod G \end{cases} \qquad (2)$$

기본형 Pollard의 Rho 알고리즘의 성능을 향상시키기 위해 제안한 방법을 일반형 Pollard의 Rho 알고리즘이라 하자. 이 알고리즘은 x_i와 X_i를 식 (3)으로 계산한다. 여기서 $M=\alpha^m$, $N=\beta^n$으로 m,n을 임의로 설정한다[5,6].

$$f_p(x)=\begin{cases} x_{i-1}^2 \pmod p, & a_i=2a_{i-1} \pmod G, b_i=2b_{i-1} \pmod G \\ Mx_{i-1} \pmod p, & a_i=a_{i-1}+m \pmod G, b_i=b_{i-1} \\ Nx_{i-1} \pmod p, & a_i=a_{i-1}, b_i=b_{i-1}+n \pmod G \end{cases} \qquad (3)$$

일반형의 대표적인 사례로 Teske[6]가 있다. Teske는 m,n 각각에 대해 선형 걸음 (linear walk)로 20회 곱셈을, 결

합 걸음 (combined walk)로 16회 곱셈과 4회 제곱을 적용하였다. 또한, Cheon et al.[9]은 Tag 함수와 랜덤 걸음을 적용하였다.

$2^{10} \equiv 5 (\mathrm{mod}\, 1019)$에서 $b=10$을 아기걸음-거인걸음과 Pollard의 Rho 알고리즘으로 찾는 횟수를 비교하여 보자.

아기걸음-거인걸음은 $m = \lceil \sqrt{1019} \rceil = 32$로, 아기걸음 단계에서 $[0,31]$에 대해 $2^i \equiv c_i (\mathrm{mod}\, n), c_i = c_{i-1}^2 (\mathrm{mod}\, n)$을 구하는 과정에서 $2^{10} \equiv 5 (\mathrm{mod}\, 1019)$을 찾아 거인걸음 단계를 수행하지 않고 10회 수행으로 해를 찾는다. 반면에, Pollard의 Rho 알고리즘은 표 1과 같이 51회 수행으로 $2^{681}5^{378} = 1010 = 2^{301}5^{416} (\mathrm{mod}\, 1019)$을 찾아 $(416-378)\ \gamma = (681-301) (\mathrm{mod} 1018), 38\gamma = 380, \gamma = 380/38 = 10$ 을 구한다.

III. 다중 병렬 수행 Pollard's Rho 알고리즘

Pollard Rho 기본형 알고리즘은 사이클 검출에서 최선의 방법으로 알려진 Floyd 알고리즘[10]을 적용하고 있다. Floyd 알고리즘은 거북이는 정상 속도로, 토끼는 거북이의 2배의 속도로 걸어가면서 충돌을 검출하는 방법이다. 이 방법은 표 1에서 알 수 있듯이 일정한 수행횟수까지는 충돌을 검출하지 못함을 알 수 있다. 따라서 충돌 검출 시점을 가능한 빨리 찾는 것이 이산대수 문제를 해결하는 목표가 될 수 있다. 기본형의 이러한 문제를 해결하기 위해 랜덤하게 걸어가는 일반형 알고리즘이 제안되었다. 또 다른 연구는 고정된 간격에 checkpoint를 위치시키는 방법과 스택을 적용한 Nivasch 알고리즘이 있다[11]. 이러한 방법은 모두 초기치가 $(0,0)$에서 시작한다.

본 장에서는 Pollard Rho 기본형의 거북이는 정상 속도, 토끼는 거북이의 2배 속도로 걸어가면서 충돌 (합동)이 발생하는 지점을 찾는데 시간이 많이 소요되며, 이 지점까지 도달하기 이전에도 이 지점의 값과 동일한 값을 갖는 지점이 존재한다는 점에 착안하여 Pollard Rho 기본형을 다양하게 변형시킨 방법을 적용하였다.

제안된 방법은 표 2와 같이 A_1, A_2, A_3, A_4의 4가지 유형을 적용한다. 초기치 $(a,b) = (0,0)$인 경우 $\alpha^0 \equiv 1 (\mathrm{mod}\, p)$에서,

Table 1. Pollard's Rho Algorithm for $2^{10} \equiv 5 (\mathrm{mod}\, 1019)$

i	$x (\mathrm{mod}\, 3)$	x	a	b	결과	$X (\mathrm{mod}\, 3)$	X	A	B	결과
초기치	-	1	0	0	$2^0 5^0 = 1$	-	1	0	0	$2^0 5^0 = 1$
1	1	2	1	0	$2^1 5^0 = 2$	1 2	2 10	1 1	0 1	$2^1 5^1 = 10$
2	2	10	1	1	$2^1 5^1 = 10$	1 2	20 100	2 2	1 2	$2^2 5^2 = 100$
3	1	20	2	1	$2^2 5^1 = 20$	1 2	200 1000	3 3	2 3	$2^3 5^3 = 1000$
4	2	100	2	2	$2^2 5^2 = 100$	1 0	981 425	4 8	3 6	$2^8 5^6 = 425$
5	1	200	3	2	$2^3 5^2 = 200$	2 0	87 436	8 16	7 14	$2^{16} 5^{14} = 436$
6	2	1000	3	3	$2^3 5^3 = 1000$	1 2	872 284	17 17	14 15	$2^{17} 5^{15} = 284$
7	1	981	4	3	$2^4 5^3 = 981$	2 2	401 986	17 17	16 17	$2^{17} 5^{17} = 986$
8	0	425	8	6	$2^8 5^6 = 425$	2 2	854 194	17 17	18 19	$2^{17} 5^{19} = 194$
...	...	...	...	...	...	...	...	...	...	...
50	0	505	680	378	...	2 2	224 101	300 300	414 415	$2^{300} 5^{415} = 101$
51	1	**1010**	**681**	**378**	$2^{681} 5^{378} = 1010$	2 1	505 **1010**	300 **301**	416 **416**	$2^{301} 5^{416} = 1010$

Table 2. Compare with application methods of Pollard Rho algorithm

구분	기존 방식	**제안 방식**
초기치	$(a,b) = (0,0)$	$(a,b) = (0,0), (1,1)$
α	$1 \le \alpha \le p-1$	2,5
β	β_γ	$\beta_\gamma, \beta_{\gamma'}, \beta_{\gamma^{-1}}$
기본형	$A_0 : (x_{i-1}^2, \alpha x_{i-1}, \beta x_{i-1})$	$A_1 : (x_{i-1}^2, \alpha^2 x_{i-1}, \beta^2 x_{i-1})$ $A_2 : (x_{i-1}^4, \alpha^2 x_{i-1}, \beta^2 x_{i-1})$
일반형	$(x_{i-1}^2, \alpha^m x_{i-1}, \beta^n x_{i-1})$ m,n = 랜덤 선택	$A_3 : (x_{i-1}^2, \alpha^m x_{i-1}, \beta^n x_{i-1})$ $A_4 : (x_{i-1}^2, \alpha^{m+1} x_{i-1}, \beta^{n+1} x_{i-1})$ $m=n=\lceil \sqrt{n} \rceil$ 고정

$(a,b) = (1,1)$이면 $\alpha\beta = \alpha^1 \alpha^\gamma = \alpha^{\gamma+1}$에서 시작하는 방법이다. 기본형은 β_γ을 적용하여 $x_i = (x_{i-1}^2, \alpha x_{i-1}, \beta x_{i-1})$의 충돌을 찾는데 반해, 제안된 방법은 $\beta_\gamma, \beta_{\gamma'}, \beta_{\gamma^{-1}}$을 적용한다. 여기서, $\beta_{\gamma'} = \alpha^{\gamma+(p-1)/2}$로, $\alpha^{(p-1)/2} \equiv -1 (\mathrm{mod}\, p)$이면 $\beta_\gamma + \beta_{\gamma'} = p$, $\alpha^{(p-1)/2} \equiv 1 (\mathrm{mod}\, p)$이면 $\beta_\gamma = \beta_{\gamma'}$가 된다. $\beta_{\gamma^{-1}} = \alpha^{(p-1)-\gamma}$은 β_γ의 역함수로 $\beta_\gamma \times \beta_{\gamma^{-1}} \equiv 1 \pmod{p}$이다. 이와 같이 다양한 형태를 적용하면 Pollard Rho 기본형 수행 횟수에 도달하기 이전이라도 x_i와 X_i가 충돌하는 지점을 빨리 찾을 수 있다.

기존의 기본형 $x_i = (x_{i-1}^2, \alpha x_{i-1}, \beta x_{i-1})$을 A_0라 하자. 제안된 방법은 4가지로, $A_1 : x_i = (x_{i-1}^2, \alpha^2 x_{i-1}, \beta^2 x_{i-1})$, $A_2 : x_i = (x_{i-1}^4, \alpha^2 x_{i-1}, \beta^2 x_{i-1})$의 변형된 기본형과 $m=n=\lceil \sqrt{n} \rceil$에 대해 $A_3 : x_i = (x_{i-1}^2 \alpha^m x_{i-1}, \beta^n x_{i-1})$와 $A_4 : x_i = (x_{i-1}^2, \alpha^{m+1} x_{i-1}, \beta^{n+1} x_{i-1})$의 변형된 일반형이다.

본 장에서는 대칭키는 $\alpha = 2$ 또는 5를 일반적으로 적용하기 때문에 이를 활용한다.

제안된 알고리즘을 $p = 1019$, $\alpha = 2,5$에 대해 Pollard Rho의 기본형 알고리즘이 충돌을 검출하기까지의 수행횟수를 비교하여 본다. 여기서는 Pollard Rho의 일반형은 m과 n을 랜덤하게 결정하는 방식으로 인해 제안된 알고리즘의 $\sqrt{n}$ 고정형 방식과의 차이점으로 인해 비교는 수행하지 않았다.

기본형 알고리즘실험에는 $\gamma = 5,100,200,300,\cdots,1000$의 11개 값을 구였으며, $A_0 : (0,0) - \beta_\gamma$ 기본형과 제안된 A_1, A_2, A_3, A_4 알고리즘을 수행한 결과는 표 3에, 수행횟수 비교 결과는 표 4에 종합하여 제시하였다. 제안된 알고리즘은 A_1, A_2, A_3, A_4가 동시에 병행으로 수행되어 최적의 결과를 갖는 방법을 해를 결정하는 방식이다. 따라서 표 4에서는 표 3의 A_1, A_2, A_3, A_4의 4개방식이 동시에 수행되어 이들 중 가장 먼저 충돌 시점을 찾은 값을 최적 해로 결정하여 Pollard Rho의 A_0와 비교하였다.

제안된 다중 방식 (A_1, A_2, A_3, A_4동시 수행의 4중 방식)을 적용할 경우 Pollard Rho의 $A_0 : (0,0) - \beta_\gamma$ 기본형에 비해 수행횟수를 71.70% 감소시킬 수 있었다.

본 논문은 이산대수를 계산하는 Pollard Rho 기본형 알고리즘의 수행횟수를 크게 감소시킨 다중 병렬 알고리즘을 제안하였다.

제안된 방법은 $\beta = \beta_\gamma, \beta_{\gamma'}, \beta_{\gamma^{-1}}$을 찾는 $A_1 : (x_{i-1}^2, \alpha^2 x_{i-1}, \beta^2 x_{i-1})$과 $A_2 : (x_{i-1}^4, \alpha^2 x_{i-1}, \beta^2 x_{i-1})$의 기본형 변형 2가지와, $A_3 : (x_{i-1}^2, \alpha^m x_{i-1}, \beta^n x_{i-1})$와 $A_4 : (x_{i-1}^2, \alpha^{m+1} x_{i-1}, \beta^{n+1} x_{i-1})$, $m = n = \lceil \sqrt{n} \rceil$의 일반형 변형 2가지 방법을 적용하였다. 제안된 4가지 방법을 동시에 수행하여 최적 해를 갖는 방법으로 해를 결정하는 다중 병렬 수행 방법은 $(0,0)$ 초기치로 $x_i = (x_{i-1}^2, \alpha x_{i-1}, \beta x_{i-1})$, $\beta = \beta_\gamma$의 충돌을 찾는 기본형 Pollard Rho 알고리즘에 비해 수행횟수를 71.70% 감소시켰다.

IV. 결론

Table 3. The number of collision detection for $n = 1019$

α	γ	충돌 검출 횟수 $(\beta_\gamma, \beta_{\gamma'}, \beta_{\gamma^{-1}})$									
		A_0		A_1		A_2		A_3		A_4	
		(0,0)	(1,1)	(0,0)	(1,1)	(0,0)	(1,1)	(0,0)	(1,1)	(0,0)	(1,1)
2	$2^{10} \equiv 5$	(**51**,57,81)	(34,57,54)	(47,47,26)	(47,47,**13**)	(23,23,48)	(23,46,24)	(28,28,30)	(28,39,30)	(32,30,36)	(32,27,18)
	$2^{100} \equiv 548$	(**19**,34,40)	(19,17,40)	(105,105,41)	(14,14,31)	(24,24,36)	(**12**,**12**,36)	(25,25,53)	(28,25,53)	(44,28,57)	(44,19,57)
	$2^{200} \equiv 718$	(60,**34**,37)	(60,34,37)	(86,86,16)	(43,43,27)	(24,24,**10**)	(34,34,36)	(29,29,30)	(29,-,40)	(31,24,32)	(31,11,65)
	$2^{300} \equiv 130$	(**20**,63,82)	(20,63,82)	(18,18,46)	(54,54, **7**)	(40,40,26)	(60,30,39)	(12,12,15)	(40,18,36)	(35,44,75)	(14,44,45)
	$2^{400} \equiv 929$	(79,74,**40**)	(79,74,40)	(60,60,24)	(20,**10**,20)	(15,15,29)	(46,23,30)	(30,30,50)	(**10**,30,-)	(82,27,50)	(82,54,20)
	$2^{500} \equiv 611$	(**36**,36,74)	(36,36,74)	(**15**,**15**,36)	(39,**15**,36)	(19,19,26)	(19,19,26)	(40,40,76)	(62,32,76)	(34,16,79)	(23,32,79)
	$2^{600} \equiv 596$	(**33**,76,45)	(33,76,45)	(53,53,22)	(53,53,22)	(36,36,42)	(12,12,42)	(**8**, **8**,23)	(-,13,23)	(40,49,25)	(45,98,25)
	$2^{700} \equiv 528$	(46,**31**,46)	(46,31,46)	(21,21,30)	(63,42,30)	(46,46,28)	(23,23,28)	(14,14,43)	(**10**,14,43)	(12,44,60)	(36,39,40)
	$2^{800} \equiv 967$	(108,**41**,84)	(108,41,84)	(42,42,68)	(42,42,**17**)	(63,63,32)	(21,42,32)	(20,20,46)	(28,-,46)	(38,80,40)	(38,60,32)
	$2^{900} \equiv 36$	(**20**,30,60)	(20,30,60)	(32,32,28)	(32,32,28)	(27,27,15)	(**6**,30,30)	(7, 7,34)	(19,-,34)	(32,70,40)	(32,70,20)
	$2^{1000} \equiv 367$	(70,72,**28**)	(70,72,28)	(27,27,50)	(18, **9**,50)	(54,54,50)	(54,27,50)	(32,32,19)	(-,32,19)	(56,36,60)	(14,23,20)
5	$5^{10} \equiv 548$	(60,-,**25**)	(60,-,25)	(20,-,15)	(20,-,23)	(33,-,20)	(33,-,24)	(22,-, **4**)	(44,-,24)	(34,-,33)	(17,-,33)
	$5^{100} \equiv 367$	(**30**,-,44)	(15,-,44)	(49,-,27)	(49,-,27)	(12,-,15)	(22,-, **6**)	(33,-,42)	(11,-,42)	(14,-,42)	(14,-,42)
	$5^{200} \equiv 181$	(**28**,-,55)	(28,-,55)	(33,-,28)	(22,-,28)	(58,-,40)	(58,-,10)	(16,-,41)	(-,-,41)	(12,-,21)	(30,-, **6**)
	$5^{300} \equiv 192$	(**19**,-,36)	(19,-,36)	(40,-,25)	(40,-,15)	(18,-,32)	(**14**,-,32)	(16,-,48)	(36,-,24)	(56,-,45)	(56,-,75)
	$5^{400} \equiv 153$	(18,-, **9**)	(16,-, 9)	(18,-,52)	(18,-,10)	(28,-, **9**)	(28,-,36)	(18,-,21)	(36,-,20)	(56,-,33)	(63,-,11)
	$5^{500} \equiv 106$	(**20**,-,40)	(20,-,40)	(20,-,28)	(20,-,28)	(16,-,34)	(24,-,14)	(**3**,-,13)	(48,-,-)	(42,-,33)	(25,-,33)
	$5^{600} \equiv 1017$	(78,-,**35**)	(78,-,35)	(22,-,**10**)	(21,-,**10**)	(31,-,33)	(20,-,66)	(29,-,47)	(29,-,16)	(40,-,34)	(40,-,34)
	$5^{700} \equiv 844$	(27,-,**11**)	(27,-,11)	(45,-,60)	(54,-,60)	(13,-,34)	(26,-,34)	(39,-,24)	(39,-,48)	(**8**,-,17)	(15,-,17)
	$5^{800} \equiv 991$	(**38**,-,40)	(38,-,40)	(17,-,48)	(17,-,32)	(30,-,18)	(36,-,16)	(36,-,52)	(72,-,52)	(**8**,-,84)	(41,-,60)
	$5^{900} \equiv 933$	(50,-,**15**)	(25,-,15)	(24,-,12)	(**8**,-,36)	(21,-,17)	(21,-,17)	(25,-,31)	(**8**,-,31)	(33,-,13)	(12,-,13)
	$5^{1000} \equiv 27$	(**14**,-,48)	(14,-,48)	(34,-,16)	(34,-,48)	(**8**,-,10)	(18,-,22)	(24,-,17)	(24,-,22)	(49,-,18)	(49,-,36)

Table 4. Compare of collision detection performance for $n = 1019$

α^γ	$A_0 - (0,0) - \beta_r$		$\min\{A_1, A_2, A_3, A_4\} - (\beta_\gamma, \beta_{\gamma'}, \beta_{\gamma^{-1}})$					
	충돌검출 수행횟수		충돌검출 수행횟수		$A_0 - (0,0) - \beta_r$ 대비			
					$\alpha = 2$		$\alpha = 5$	
	$\alpha = 2$	$\alpha = 5$	$\alpha = 2$	$\alpha = 5$	수행횟수 비율	성능 향상율	수행횟수 비율	성능 향상율
α^{10}	51	60	13	4	25.49 %	74.51 %	6.67 %	93.33 %
α^{100}	19	30	12	6	63.16 %	36.84 %	20.00 %	80.00 %
α^{200}	60	28	10	6	16.67 %	83.33 %	21.43 %	78.57 %
α^{300}	20	19	7	14	35.00 %	65.00 %	73.68 %	26.32 %
α^{400}	79	18	10	9	12.66 %	87.34 %	50.00 %	50.00 %
α^{500}	36	20	15	3	41.67 %	58.33 %	15.00 %	85.00 %
α^{600}	33	78	8	10	24.24 %	75.76 %	12.82 %	87.18 %
α^{700}	46	27	10	8	21.74 %	78.26 %	29.63 %	70.37 %
α^{800}	108	38	17	8	15.74 %	84.26 %	21.05 %	78.95 %
α^{900}	20	50	6	8	30.00 %	65.00 %	16.00 %	84.00 %
α^{1000}	70	14	9	8	12.86 %	87.14 %	57.14 %	42.86 %
평 균					27.20 %	**72.80 %**	29.40%	**70.60 %**
					71.70 %			

제안된 알고리즘은 임의로 설정된 가상의 값에 대해 이산대수 값을 Pollard Rho 알고리즘보다 빨리 찾을 수 있음을 보였다. 추후, 실제적으로 적용되고 있는 RSA에 대해서도 제안된 알고리즘의 효율성을 검증할 예정이다.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms, Section 31.7 The RSA Public-key Cryptosystem", 2nd Ed., MIT Press and McGraw-Hill. pp. 881-887, 2001.

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

[3] D. Shanks, "The Infrastructure of a Real Quadratic Field and its Applications," Proceedings of the 1972 Number Theory Conference, University of Colorado, Boulder, 1972.

[4] E. Teske, "Speeding Up Pollard's Rho Method for Computing Discrete Logarithms," Lecture Notes in Computer Science, Vol. 1423, pp. 541-554, Jun. 1998.

[5] S. Bai and R. P. Brent, "On the Efficiency of Pollard's Rho Method for Discrete Logarithms," Computing: The Australasian Theory Symposium (CATS), Vol. 77, pp. 125-131, Jan. 2008.

[6] 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.

[7] D. C. Terr, "A modification of Shanks' Baby-step Giant-step algorithm," Mathematics of Computation, Vol. 69, No. 230, pp. 767-773, Mar. 1999.

[8] S. U. Lee, "Baby-Step Adult-Step Algorithm for Discrete Logarithm," Journal of KIIT, Vol. 11, No. 10, pp. 121-128, Oct. 2013.

[9] J. H. Cheon, J. Hong, and M. K. Kim, "Accelerating Pollard's Rho Algorithm on Finite Fields", Journal of Cryptography, pp. 1-48, 2010.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms, Section 25.2, The Floyd-Warshall Algorithms", 2nd Ed., MIT Press and McGraw-Hill. pp. 629-632, 2001.

[11] A. Shamir, "Random Graphs in Cryptography", 7th Haifa Workshop on Interdisciplinary Applications of Graph Theory, Combinatorics and Algorithms, 2007.

Authors

Sang Un Lee received the B. Sc. degree in avionics from the Korea Aerospace University in 1997. He received the M. Sc. and Ph. D. degrees in Computer Science from Gyeongsang National University, Korea, in 1997 and 2001, respectively. He is currently Professor with the Department of Multimedia Science, Gangneung-Wonju National University, Korea. He is interested in software quality assurance and reliability modeling, software engineering, software project management, neural networks, and algorithm.