The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 17, No. 6, pp.19-25, Dec. 31, 2017. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2017.17.6.19
JIIBC 2017-6-3

다중 초기치 Pollards's Rho 소인수분해 알고리즘

Integer Factorization Algorithm of Pollard's Rho Based on Multiple Initial Values

이상운*

Sang-Un, Lee*

요 약 본 논문은 비트코인 채굴에 필요한 SHA-256 암호 해시 값(n)을 구성하는 2개의 소수(p,q)를 빠르게 해독하는 소인수분해법을 다룬다. 본 논문에서는 Pollard's Rho 소인수분해 알고리즘의 수행횟수를 월등히 감소시킨 알고리즘을 제안하였다. Rho (ρ) 알고리즘은 $(x_0,y_0)=(2,2)$ 초기치에 대해 $x_i=x_{i-1}^2+1 (\mathrm{mod}\, n)$과 $y_i=[(y_{i-1}^2+1)^2+1] (\mathrm{mod}\, n)$을 계산하여 $1<\gcd(|x_i-y_i|,n)<n$으로 소인수를 구한다. 이 알고리즘은 특정 합성수에 대해서는 소인수 분해에 실패할 수 있다. 제안된 알고리즘은 Pollard Rho 알고리즘에 $(x_0,y_0)=(2^k,2^k)$와 $(2^k,2), 2\le k\le 10$을 적용하였다. 그 결과 모든 합성수에 대해 소인수분해를 할 수 있었으며, Pollard Rho 알고리즘의 수행횟수를 67.94% 감소시켰다.

Abstract This paper deals with integer factorization of two prime p,q of SHA-256 secure hash value n for Bit coin mining. This paper proposes an algorithm that greatly reduces the execution time of Pollard's rho integer factorization algorithm. Rho(ρ) algorithm computes $x_i=x_{i-1}^2+1 (\mathrm{mod}\, n)$ and $y_i=[(y_{i-1}^2+1)^2+1] (\mathrm{mod}\, n)$ for intial values $(x_0,y_0)=(2,2)$ to find the factor $1<\gcd(|x_i-y_i|,n)<n$. It however fails to factorize some particular composite numbers. The algorithm proposed in this paper applies multiple initial values $(x_0,y_0)=(2^k,2^k)$ and $(2^k,2), 2\le k\le 10$ to the existing Pollard's Rho algorithm. As a results, the proposed algorithm achieves both the factorization of all the composite numbers and the reduction of the execution time of Pollard's Rho by 67.94%.

Key Words : Integer factorization, Greatest common divider (gcd), Pollard rho algorithm

Ⅰ. 서 론

요즘 인터넷을 활용한 전자화폐 채굴로 각광 받고 있는 비트코인(Bit coin) 채굴과정은 컴퓨터가 암호를 푸는 과정이라 할 수 있으며, 이는 간단한 암호원리와 SHA-256(256비트 이진수, 32 byte 16진수, RSA-2048)로 구성된 해시함수(hash function)로 이루어져 있다. RSA- 2048은 현재까지 소인수분해 값(두 개의 소수)이 알려져 있지 않다. 따라서 블록 생성 문제의 해답인 nonce를 얻어 비트코인을 획득하려면 SHA-256가 갖고 있는 22,562,256개의 해들 중 어느 하나를 찾기 위해 무한히 반복 수행되는 암호를 해독해야만 한다. 이러한 이유로 인해 비트코인 채굴시간을 보다 단축시키기 위해서는 암호를 해독하는 소인수분해 방법을 개선할 필요성이 대

*정회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
접수일자 : 2017년 10월 11일, 수정완료 : 2017년 11월 11일
게재확정일자 : 2017년 12월 8일

Received: 11 October, 2017 / Revised: 11 November, 2017 / Accepted: 8 December, 2017
*Corresponding Author: sulee@gwnu.ac.kr
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

두되었다. 이러한 필요성에 의해 본 논문에서는 기존에 알려진 암호해독의 소인수분해법을 개선한 방법을 제안한다.

대표적인 비대칭 암호인 *RSA*의 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]

소인수분해 방법에는 나눗셈시행법 (Trial division), 제곱합동법 (Congruence of squares), Pollard's Rho, Quadratic Sieve (Q Sieve), General Number Field Sieve (GNFS) 등 다양하게 존재하고 있다.[3] 나눗셈 시행법은 $O(\sqrt{n})$ 복잡도로 $n/p, 3\le p<\sqrt{n}, (p=odd)$으로 거의 사용하지 않는다. Pollard의 Rho 알고리즘[4]은 10자리 이하의 수에 대한 소인수분해에 최적으로 알려져 있으며 $O(\sqrt{p})\approx O(\sqrt[4]{n})$이다. 제곱합동법 $a^2\equiv b^2 \pmod n, a=(p+q)/2, b=(q-p)/2, p<q$에 기반한 Q Sieve는 130자리수 이하의 수에 적합하며, RSA-129가 Q Sieve로 해독되었다. GNFS는 130자리 이상의 수에 적합하며, RSA-140과 RSA-155가 이 방법으로 해독되었다.[5,6]

Brent는 Pollard의 Rho 알고리즘의 수행시간을 약 24% 향상시킨 알고리즘을 제안하였다.[7] Pollard의 Rho 알고리즘은 제곱합동법이나 나눗셈시행법에 비해 월등히 빠르게 소인수를 찾을 수 있지만 특정 합성수에 대해서는 찾지 못하는 단점이 있다.

본 논문에서는 Pollard의 Rho 알고리즘의 수행횟수를 약 68%감소시키면서 모든 합성수에 대해 소인수분해를 할 수 있는 알고리즘을 제안한다.

2장에서는 Pollard의 Rho 알고리즘과 문제점을 고찰한다. 3장에서는 Pollard rho 알고리즘의 문제점을 해결하고, 수행횟수도 획기적으로 감소시킨 다중-Pollard Rho 알고리즘을 제안하고, 4장에서는 제안된 알고리즘의 성능을 검증하여 본다.

II. Pollard Rho 소인수분해 알고리즘

$n=pq$로 소인수의 개수가 2인 반소수 (semi-prime) 또는 소인수의 개수가 3 이상인 카마이클 수 (Carmichael number)인 합성수 (composite number)라 가정하자.

반소수 $n=p\times q$에서 p의 배수는 q개, q의 배수는 p개, $p\cap q=1$개로 $(p+q-1)$개가 존재한다. Pollard의 Rho 알고리즘은 $(p+q-1)$개의 소인수 배수들 중에서 어느 하나를 x^2+c로 찾는 방법이다.

Pollard의 Rho 알고리즘은 그림 1에 제시되어 있으며, $x_0=y_0=2, c=1$로 설정하고 $x=f(x), y=f(f(y))$을 계산하여 식 (1)로 소인수를 구한다.[3,4]

$$\begin{aligned} &x_0=y_0=2, c=1\\ &x_i=(x_{i-1})^2+c \pmod n, c\neq 0,-2\\ &y_i=[(y_{i-1})^2+c]^2+c \pmod n, c\neq 0,-2\\ &1<\gcd(|x_i-y_i|,n)<n \end{aligned} \quad (1)$$

Brent는 Pollard의 Rho 알고리즘의 $y=f(f(y))$를 구하지 않고, $y=x_i, (i=2^k)$를 적용하는 그림 2의 알고리즘을 제안하였으며, Pollard의 Rho 알고리즘의 수행시간을 약 24% 감소시키는 효과를 얻었다.[3,7]

$x=f(x), y=f(f(y))$
$f(x)=x^2+c \pmod n, c\neq 0,-2.$
$f(y)=y^2+c \pmod n, c\neq 0,-2.$

입력 : n, 출력 : p or q

초기치 : $x_0=2, y_0=2, c=1, d=1.$
do
  $x_i=(x_{i-1})^2+1 \pmod n$
  $y_i=[(y_{i-1})^2+1)]^2+1 \pmod n$
  $d=\gcd(|x_i-y_i|,n)$
  if $1<d<n$ then return d.
end do

그림 1. Pollard's Rho 알고리즘
Fig. 1. Pollard's Rho Algorithm

입력 : n, 출력 : p or q
$i=1, k=2$.
초기치 : $x_1=2, y=2.$
while TRUE
  do $i\leftarrow i+1$
    $x_i=(x_{i-1})^2+1 \pmod n$
    $d=\gcd(|x_i-y|,n)$
    if $1<d<n$ then return d.
    if $i=k$ then $y\leftarrow x_i$, $k\leftarrow 2k$.
  end do

그림 2. Brent's 알고리즘
Fig. 2. Brent's Algorithm

Pollard의 Rho 알고리즘을 적용할 경우 표 1과 같이 소인수를 찾지 못하는 경우가 발생한다.

l을 자리수라 할 경우, $l(n)=4$인 [1000,9999]에서 $n=pq$인 반소수 887개를 대상으로 Pollard의 Rho 알고리즘인 $(x_0,y_0)=(2,2)$를 적용할 경우 소인수분해를 하지 못하는 수를 검증한 결과는 표 2와 같이 15개로 1.69%이다. 즉, Pollard의 Rho 알고리즘은 반소수에 대해 1.69%의 오차를 갖고 있다. 이 경우 $f(x)$를 변경시켜 다시 찾아야 하는 문제점을 갖고 있다.

표 1. Pollard의 Rho 알고리즘 실패 사례
Table 1. Failure Example of Pollard's Rho Algorithm

$n=$	1027	(13×79)			
i	x_0	y_0			
0	2	2			
	x_i	y_{i1}	y_{i2}	$\lvert x_i-y_{i2}\rvert$	$\gcd(\lvert x_i-y_{i2}\rvert,n)$
1	5	5	26	21	1
2	26	677	288	262	1
3	677	785	26	651	1
4	288	677	288	0	**1027**
5	785	785	26	759	1
6	26	677	288	262	1
7	677	785	26	651	1
8	288	677	288	0	1027
9	785	785	26	759	1
10	26	677	288	262	1
11	677	785	26	651	1
12	288	677	288	0	1027
13	785	785	26	759	1
14	26	677	288	262	1
15	677	785	26	651	1
16	288	677	288	0	1027
17	785	785	26	759	1
18	26	677	288	262	1
19	677	785	26	651	1
20	288	677	288	0	1027

표 2. $l(n)=4$에서 Pollard의 Rho 알고리즘 실패사례
Table 2. Failures of Pollard's Rho Algorithm in $l(n)=4$

순번	n	p	q	소인수분해 여부				
				$c=1$	$c=2$	$c=3$	$c=-1$	$c=-3$
1	1027	13	79	×	○	-	-	-
2	1241	17	73	×	×	×	○	-
3	1469	13	113	×	×	×	○	-
4	1943	29	67	×	○	-	-	-
5	2249	13	173	×	×	×	○	-
6	3587	17	211	×	×	×	○	-
7	3683	29	127	×	○	-	-	-
8	3749	23	163	×	×	○	-	-
9	5371	41	131	×	×	×	×	×
10	5671	53	107	×	○	-	-	-
11	8149	29	281	×	○	-	-	-
12	8509	67	127	×	○	-	-	-
13	8927	79	113	×	○	-	-	-
14	9167	89	103	×	×	○	-	-
15	9259	47	197	×	×	○	-	-

Lee[8]는 $\alpha^{\gamma}\equiv\beta(\mathrm{mod}\,p)$에서 γ를 구하는 이산대수 Pollard Rho 알고리즘의 수행횟수를 감소시키고자 하였다. Pollard Rho 알고리즘의 기본형을 $\alpha^a\beta^b\equiv x(\mathrm{mod}\,p)$라 할 때 $x_i=(x_{i-1})^2,\alpha x_{i-1},\beta x_{i-1}$이며, 일반형은 임의로 설정된 $M=\alpha^m, N=\beta^n$에 대해 $x_i=(x_{i-1})^2, Mx_{i-1}, Nx_{i-1}$으로 계산된다. Lee[8]는 $m=n=\lceil\sqrt{n}\rceil$, $(a,b)=(0,0),(1,1)$로 부터 $\beta_{\gamma}=\alpha^{\gamma},\beta_{\gamma'}=\alpha^{(p-1)/2+\gamma}$, $\beta_{\gamma^{-1}}=\alpha^{(p-1)-\gamma}$을 찾는 4가지 형태의 모델을 동시에 수행하여 그들 중 가장 먼저 결과를 얻는 방법을 선택하는 다중 병렬처리 방법을 제안하였다.

본 논문에서는 초기치를 다중으로 설정하여 Pollard Rho 알고리즘을 빠르게 수행하는 방법을 제안한다. 따라서 제안된 알고리즘은 Lee[8]의 알고리즘과는 차별성이 있으며, Pollard rho 알고리즘이 소인수 분해를 실패하는 반소수에 대해서도 소인수분해를 할 수 있는 장점을 갖고 있음을 보인다.

Ⅲ. 다중 초기치 Pollard Rho 알고리즘

Pollard rho 알고리즘은 $(x_0,y_0)=(2,2)$를 적용한다. 이 경우 1.69%의 반소수에 대해서는 소인수 분해 실패 확률을 갖고 있으며, 수행횟수도 $\sqrt{p}$ 수행된다. 이러한 소인수 분해 실패시 주어진 수 n이 합성수라면 $f(x)=x^2+c(\mathrm{mod}\,n)$의 c값을 변경시키면서 다시 수행해야 한다.

본 장에서 제안하는 알고리즘은 이러한 문제점을 간단히 해결하였다. 제안된 알고리즘은 그림 3에 제시되어 있으며, $f(x)=x^2+c(\mathrm{mod}\,n)$을 변형시키지 않고 사용한다. 다만, 식 (2)와 같이 초기치를 $(x_0,y_0)=(2^k,2^k)$과 $(2^k,2^1), 2\le k\le 10$을 적용한다.

입력 : n, 출력 : p or q
초기치 : $x_{10}=2^k, y_{10}=2^k, c=1, d=1.$
$x_{20}=2^k, y_{20}=2, c=1, d=1.$
$i=1$.
do
for $2\le k\le 10$
for $1\le j\le 2$
$x_{ijk}=(x_{(i-1)jk})^2+1(\mathrm{mod}\,n)$
$y_{ijk}=[(y_{(i-1)jk})^2+1)]^2+1](\mathrm{mod}\,n)$
end
end
$d_{ijk}=\gcd(|x_{ijk}-y_{ijk}|,n)$
if $1<d_{ijk}<n$ then return $2^k, d$
else $i=i+1$, continue.
end do

그림 3. 다중 초기치 Pollard's Rho 알고리즘
Fig. 3. Multiple Initial Values Based Pollard's Rho Algorithm

$$(x_0,y_0)=(2^k,2^k) \text{ 과 } (2^k,2^1),\ 2\le k\le 10,\ c=1$$
$$x_i=(x_{i-1})^2+c(\bmod n),\ c\neq 0,-2$$
$$y_i=[(y_{i-1})^2+c]^2+c(\bmod n),\ c\neq 0,-2$$
$$1<\gcd(|x_i-y_i|,n)<n \qquad (2)$$

Ⅳ. 실험 및 결과 분석

본 장에서는 Pollard의 Rho 알고리즘과 제안된 다중-Pollard Rho 알고리즘의 성능을 비교분석하여 본다. 실험에는 $l(p)=l(q)=2,3,4$ 각 10개, $l(p)=2, l(q)=3$ 10개, $l(p)=3, l(q)=4$ 10개의 반소수 50개 데이터를 선택하였다. 표 2의 Pollard의 Rho 알고리즘으로 소인수분해를 실패한 사례에 대해 제안된 알고리즘을 수행한 결과는 표 3에 제시되어 있다.

표 3. Pollard Rho 알고리즘 실패 데이터의 결과
Table 3. The Result of failure data for Pollard's Rho Algorithms

순번	n	p	q	2^k (수행횟수)	
				$(2^k,2^k)$	$(2^k,2)$
1	1027	13	79	**4 (1)**	**8 (1)**
2	1241	17	73	**8 (1)**	8 (2)
3	1469	13	113	**4 (1)**	**8 (1)**
4	1943	29	67	8 (2)	**8 (1)**
5	2249	13	173	**4 (1)**	**8 (1)**
6	3587	17	211	4 (6)	**8 (2)**
7	3683	29	127	8 (2)	**256 (1)**
8	3749	23	163	32 (2)	**64 (1)**
9	5371	41	131	4 (7)	**128 (1)**
10	5671	53	107	8 (4)	**8 (2)**
11	8149	29	281	8 (2)	**256 (1)**
12	8509	67	127	**16 (2)**	4 (3)
13	8927	79	113	**512 (2)**	32 (3)
14	9167	89	103	**4 (4)**	**4 (4)**
15	9259	47	197	1024 (3)	**512 (1)**

표 3에서 Pollard Rho 알고리즘으로 소인수 분해를 실패한 사례 모두에 대해서도 최대 4회 이내에서 소인수분해 할 수 있는 능력을 갖고 있다.

50개 데이터에 Pollard의 Rho 알고리즘과 제안된 알고리즘을 적용한 결과는 표 4에 제시되어 있다. 표에서 $n=1943$의 경우 $c=1$인 Pollard Rho 알고리즘으로 소인수 분해에 실패하여 $c=2$를 적용한 수행횟수를 표기하였다. 표 4에서 제안된 알고리즘을 수행한 결과 Pollard Rho 알고리즘의 수행횟수를 67.94% 감소시켰음을 알 수 있다.

Pollard의 Rho 알고리즘은 초기치에 따라 알고리즘 수행횟수에 커다란 변화를 보여 다른 2가지 방식의 알고리즘도 추가로 실험을 수행하였다. 첫 번째는 x_0를 n의 *MSB*에서 *LSB*로 1자리, 2자리, 순으로 선택하는 $x_{LR}=$ *MSB*→*LSB*로 선택하면서 Pollard의 Rho 알고리즘을 수행하는 방식이다. 두 번째는 $x_{LR}=MSB\rightarrow LSB$와 $x_{RL}=LSB\rightarrow MSB$ 초기치로 Pollard의 Rho 알고리즘을 수행하면서 $y_i=2^i$, (i=수행횟수)로 변형된 Brent 알고리즘을 수행하는 방식이다. 추가로 수행된 2가지 알고리즘의 수행 결과는 표 5에 제시되어 있다.

첫 번째 방식의 알고리즘은 Pollard의 Rho 알고리즘에 비해 32.97%를 수행하여 제안된 $(2^k,2^k)$ 초기치 방식보다 약간 성능이 떨어진다. 또한, 이 방법은 n의 자리수가 커짐에 따라 초기치 개수가 따라서 증가하는 단점이 있다. 두 번째 방법은 Pollard의 Rho 알고리즘 수행횟수에 비해 20.04%만 수행하여 가장 좋은 성능을 보였다. 그러나 이 방법은 첫 번째 방법과 동일하게 초기치 개수가 증가함과 더불어 최대공약수 계산 횟수도 많음을 알 수 있다. 따라서 제안된 $(2^k,2^k)$ 초기치 방법이 Pollard의 Rho 알고리즘의 단점을 해소할 수 있는 최선의 방법임을 알 수 있다.

Ⅴ. 결론

본 논문은 소인수분해에 탁월한 성능을 갖춘 Pollard의 Rho 알고리즘이 특정 합성수에 대해서는 소인수분해에 실패하는 문제점을 해결함과 더불어 수행횟수도 획기적으로 감소시킨 알고리즘을 제안하였다.

제안된 알고리즘은 초기치를 $(x_0,y_0)=(2,2)$ 대신 $(2^k,2^k)$와 $(2^k,2),\ 2\le k\le 10$를 적용하여 Pollard의 Rho 알고리즘을 수행하는 방식이다.

제안된 알고리즘은 $l(n)=4$의 반소수들 중에서 Pollard Rho 알고리즘으로 소인수분해에 실패한 15개 데이터들에 대해서도 4회 이내에 소인수분해를 할 수 있는 능력을 갖고 있을 뿐 아니라, 전반적으로 Pollard Rho 알고리즘의 수행횟수를 67.94% 감소시켰다.

표 4. 알고리즘 수행 결과
Table 4. The Result of Algorithms

No	p	q	n	Pollard Rho 알고리즘 $k=1$	다중-Pollard Rho 알고리즘 $2 \le k \le 10$		최소값	비율 (%)
				$(2^k,2^k)$	$(2^k,2^k)$	$(2^k,2)$		
				수행횟수	2^k(수행횟수)	2^k(수행횟수)		
1	13	23	299	6	4 (1)	8 (1)	1	16.67
2	29	67	1943	x (6)	16 (2)	256 (1)	1	**16.67**
3	37	31	1147	2	64 (1)	32 (1)	1	50.00
4	41	53	2173	6	8 (4)	128 (1)	1	16.67
5	59	47	2773	6	16 (3)	64 (1)	1	16.67
6	61	79	4819	4	1024 (1)	32 (3)	1	25.00
7	73	67	4891	6	8 (1)	4 (3)	1	16.67
8	83	29	2407	5	64 (2)	256 (1)	1	20.00
9	89	17	1513	6	4 (3)	8 (2)	2	33.33
10	97	19	1843	4	8 (1)	128 (1)	1	25.00
11	113	997	112661	4	8 (3)	512 (4)	3	75.00
12	241	839	202199	13	16 (1)	64 (7)	1	7.69
13	331	727	240637	28	32 (1)	16 (7)	1	3.57
14	461	659	303799	22	8 (7)	1024 (5)	5	22.73
15	521	569	296449	26	4 (19)	256 (10)	10	38.46
16	631	463	292153	26	1024 (10)	16 (7)	7	26.92
17	751	317	238067	7	1 (7)	1024 (6)	6	85.71
18	881	269	236989	15	1 (15)	512 (2)	2	13.33
19	911	193	175823	17	4 (4)	4 (3)	3	17.65
20	991	103	102073	14	16 (4)	4 (4)	4	28.57
21	1621	4999	8103379	16	1 (18)	64 (22)	18	112.50
22	1871	4789	8960219	67	4 (10)	8 (3)	3	4.48
23	2161	4493	9709373	50	16 (4)	8 (15)	4	8.00
24	2521	4337	10933577	36	32 (23)	8 (10)	10	27.78
25	3061	3943	12069523	27	4 (27)	64 (9)	9	33.33
26	3581	3617	12952477	17	1 (17)	128 (8)	8	47.06
27	4021	3313	13321573	49	256 (2)	64 (4)	2	4.08
28	4481	2917	13071077	66	16 (30)	1024 (29)	29	43.94
29	4721	2459	11608939	26	1 (26)	4 (26)	26	100.00
30	4951	1777	8797927	16	1 (16)	64 (14)	14	87.50
31	23	113	2599	6	32 (2)	64 (1)	1	16.67
32	67	241	16147	8	16 (1)	4 (3)	1	12.50
33	31	331	10261	2	32 (1)	64 (2)	1	50.00
34	53	461	24433	6	8 (4)	8 (2)	2	33.33
35	47	521	24487	8	1024 (3)	512 (1)	1	12.50
36	79	631	49849	4	512 (2)	4 (4)	2	50.00
37	67	751	50317	8	16 (2)	4 (3)	2	25.00
38	29	881	25549	8	8 (2)	256 (1)	1	12.50
39	17	911	15487	6	4 (4)	8 (2)	2	33.33
40	19	991	18829	3	8 (1)	128 (1)	1	33.33
41	997	1621	1616137	22	128 (14)	32 (10)	10	45.45
42	839	1871	1569769	18	16 (9)	8 (3)	3	16.67
43	727	2161	1571047	28	64 (3)	512 (14)	3	10.71
44	659	2521	1661339	22	16 (8)	1024 (8)	8	36.36
45	569	3061	1741709	29	1 (29)	64 (9)	9	31.03
46	463	3581	1658003	17	1024 (10)	128 (8)	8	47.06
47	317	4021	1274657	7	1 (7)	64 (4)	4	57.14
48	269	4481	1205389	15	1 (15)	512 (2)	2	13.33
49	193	4721	911153	24	256 (4)	4 (3)	3	12.50
50	103	4951	509953	14	1 (14)	4 (4)	4	28.57
평균								**32.06 %**

표 5. 다른 방식의 알고리즘 수행 결과

Table 5. The Result of Another Algorithms

				(2,2)	$x_{LR}=MSB\rightarrow LSB$ (x_{LR}, 수행횟수)					$x_{LR}=MSB\rightarrow LSB$, $x_{RL}=LSB\rightarrow MSB$, $y_i=2^i$, i=수행횟수 (x, 수행횟수, i)			
No	p	q	n	수행횟수	$(x_{LR},2)$	$(2,x_{LR})$	(x_{LR},x_{LR})	최소값	비율	$(x_{LR},2^i)$	$(x_{RL},2^i)$	최소값	비율
1	13	23	299	6	299,2	299,4	29,1	1	16.67	2,1,9	9,1,2	1	16.67
2	29	67	1943	x(6)	19,3	19,7	19,4	3	**50.00**	194,1,8	3,1,16	1	**16.67**
3	37	31	1147	2	11,2	11,1	11,1	1	50.00	114,1,3	47,1,6	1	50.00
4	41	53	2173	6	217,1	217,5	217,4	1	16.67	21,1,5	3,1,8	1	16.67
5	59	47	2773	6	277,1	27,7	277,3	1	16.67	2,1,6	3,1,7	1	16.67
6	61	79	4819	4	4,4	4,3	48,1	1	25.00	28,1,10	819,1,7	1	25.00
7	73	67	4891	6	4,3	4,2	489,6	2	33.33	489,2,1	1,1,10	2	33.33
8	83	29	2407	5	24,1	240,2	240,2	1	20.00	240,1,12	407,1,1	1	20.00
9	89	17	1513	6	1513,4	15,6	15,3	3	50.00	1,1,9	513,2,4	1	16.67
10	97	19	1843	4	1,2	18,1	1,3	1	25.00	18,1,1	43,1,6	1	25.00
11	113	997	112661	4	112661,2	112,1	1,4	1	25.00	112,1,1	661,1,6	1	25.00
12	241	839	202199	13	20219,6	20219,7	20,8	6	46.15	20219,4,9	199,2,10	2	15.38
13	331	727	240637	28	240,9	24,9	2406,7	7	25.00	2,8,15	637,4,8	4	14.29
14	461	659	303799	22	3,5	303,10	303,10	5	22.73	303,4,12	3799,4,0	4	18.18
15	521	569	296449	26	29,8	2964,16	29644,19	8	30.77	29,9,10	6449,1,8	1	3.85
16	631	463	292153	26	29215,11	29215,6	29,11	6	23.08	29,9,10	53,1,5	1	3.85
17	751	317	238067	7	238067,12	238067,9	238067,7	7	100.00	238067,8,5	67,3,5	3	42.86
18	881	269	236989	15	23698,2	236,4	23,15	2	13.33	2,40,0	9,3,0	3	20.00
19	911	193	175823	17	17,4	17582,2	17,2	2	11.76	1758,3,13	23,2,13	2	11.76
20	991	103	102073	14	102073,12	10,1	1020,10	1	7.14	102,1,1	2073,3,15	1	7.14
21	1621	4999	8103379	16	8103379,18	8103379,16	810,12	12	75.00	81,4,2	103379,13,13	4	25.00
22	1871	4789	8960219	67	8,3	89,59	89,18	3	4.48	89602,4,6	X	4	5.97
23	2161	4493	9709373	50	9,17	97093,7	9709,10	7	14.00	97093,3,2	373,16,14	3	6.00
24	2521	4337	10933577	36	10,16	1093,27	10,23	16	44.44	10933,3,5	33577,14,14	3	8.33
25	3061	3943	12069523	27	12069,9	12069523,29	120,27	9	33.33	12069,1,4	523,10,13	1	3.70
26	3581	3617	12952477	17	1295247,10	1295247,13	129,14	10	58.82	1,9,7	952477,3,9	3	17.65
27	4021	3313	13321573	49	13,15	13321,46	13321,7	7	14.29	1332175,18,9	X	18	36.73
28	4481	2917	13071077	66	13,35	130710,11	130,18	11	16.67	13,31,2	71077,7,12	7	10.61
29	4721	2459	11608939	26	116089,16	116089,10	1,26	10	38.46	116089,9,0	9,26,4	9	34.62
30	4951	1777	8797927	16	8797927,18	8797927,18	8797927,20	18	**112.50**	879792,7,2	97927,13,5	7	43.75
31	23	113	2599	6	2599,2	25,4	259,2	2	33.33	2599,1,11	9,1,7	1	16.67
32	67	241	16147	8	161,2	16,8	16,1	1	12.50	16,1,4	147,1,14	1	12.50
33	31	331	10261	2	1,3	1,2	10,3	2	100.00	10,1,3	1,1,6	1	50.00
34	53	461	24433	6	2443,1	244,4	2443,4	1	16.67	244,2,14	33,1,13	1	16.67
35	47	521	24487	8	24,4	2448,3	24,4	3	37.50	2448,1,6	487,1,3	1	12.50
36	79	631	49849	4	4,4	49,2	498,1	1	25.00	49,1,5	49,1,5	1	25.00
37	67	751	50317	8	50317,10	50317,8	503,2	2	25.00	503,1,13	17,2,4	1	12.50
38	29	881	25549	8	255,5	25,4	255,2	2	25.00	255,1,3	9,1,8	1	12.50
39	17	911	15487	6	154,5	154,1	1,6	1	16.67	154,1,2	7,1,4	1	16.67
40	19	991	18829	3	1,4	18,1	188,3	1	33.33	18,1,2	9,1,14	1	33.33
41	997	1621	1616137	22	161613,5	161,3	161613,10	3	13.64	161,2,0	37,7,2	2	9.09
42	839	1871	1569769	18	1569769,16	1569769,11	1,18	11	61.11	156976,4,9	769,3,3	3	16.67
43	727	2161	1571047	28	1571,22	15710,10	15710,3	3	10.71	15710,4,4	1047,4,15	4	14.29
44	659	2521	1661339	22	166133,12	166133,10	16,8	8	36.36	1,1,6	339,1,8	1	4.55
45	569	3061	1741709	29	174,5	174,24	1,29	5	17.24	174,11,16	41709,3,16	3	10.34
46	463	3581	1658003	17	16580,13	165800,6	1658,8	6	35.29	1658003,10,7	3,3,4	3	17.65
47	317	4021	1274657	7	1274657,12	1274657,9	1,7	7	100.00	1274657,8,5	274657,5,13	5	71.43
48	269	4481	1205389	15	120538,2	12053,7	1,15	2	13.33	1205,10,11	9,3,0	3	20.00
49	193	4721	911153	24	91,3	91115,2	9,8	2	8.33	9111,7,15	3,4,15	4	16.67
50	103	4951	509953	14	50,7	50995,1	5,14	1	7.14	50,3,4	53,3,4	3	21.43
평균									**32.97**				**20.04**

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. ISBN: 9780262533058

[2] D. R. Stinson, "Cryptography: Theory and Practice," 3rd ed., London, CRC Press, 2005. ISBN: 97815848850852006

[3] C. Barnes, "Integer Factorization Algorithms," Department of Physics, Oregon State University, 2004.

[4] J. M. Pollard, "A Monte Carlo Method for Factorization," Bit Numerical Mathematics (BIT), Vol. 15, No. 3, pp. 331-334, Sep. 1975. doi:10.1007/BF01933667
[5] K. Rosen, "Discrete Mathematics and It's Applications," 6th Ed., McGrew-Hill, 2011. ISBN-10: 0072899050
[6] R. Montenegro, "Lecture Notes on Pollard's Rho", http://ravimontenegro.com/92.360/PollardRho.pdf, 2011.
[7] R. P. Brent, "An Improved Monte Carlo Factorization Algorithm," Bit Numerical Mathematics (BIT), Vol. 20, No. 2, pp. 176-184, Jun. 1980. doi:10.1007/BF01933190
[8] S. U. Lee, "Multiple Parallel-Pollard's Rho Discrete Logarithm Algorithm," Journal of The Korea Society of Computer and Information (KSCI), Vol. 20 No. 8, pp. 29-33, Aug. 2015. doi:10.9708/jksci.2015.20.8.029

저자 소개

이 상 운(정회원)

- 1987년: 한국항공대학교 항공전자공학과 (학사)
- 1997년: 경상대학교 컴퓨터과학과 (석사)
- 2001년 : 경상대학교 컴퓨터과학과 (박사)
- 2003년 : 강원도립대학 컴퓨터응용과 전임강사
- 2004년 ~ 2007.2 : 국립 원주대학 여성교양과 조교수
- 2007.3 ~ 2015.3 : 강릉원주대학교 멀티미디어공학과 부교수
- 2015.4 ~ 현재 : 강릉원주대학교 멀티미디어공학과 정교수

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

- e-mail : sulee@gwnu.ac.kr