Journal of The Korea Society of Computer and Information
Vol. 19, No. 7, July 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.7.071

RSA의 오일러 함수 $\phi(n)$ 해독 $2^k\beta$ 알고리즘

이 상 운*

A $2^k\beta$ Algorithm for Euler function $\phi(n)$ Decryption of RSA

Sang-Un Lee *

요 약

대표적인 공개키 암호방식인 RSA에 사용되는 합성수 $n=pq$의 큰자리 소수 p,q를 소인수분해하여 구하는 것은 사실상 불가능하다. 공개키 e와 합성수 n은 알고 개인키 d를 모를 때, $\phi(n)=(p-1)(q-1)=n+1-(p+q)$을 구하여 $d=e^{-1}(\mathrm{mod}\,\phi(n))$의 역함수로 개인키 d를 해독할 수 있다. 따라서 $\phi(n)$을 알기 위해 n으로부터 p,q를 구하는 수학적 난제인 소인수분해법을 적용하고 있다. 소인수분해법에는 $n/p=q$의 나눗셈 시행법보다는 $a^2\equiv b^2(\mathrm{mod}\,n), a=(p+q)/2, b=(q-p)/2$의 제곱합동법이 일반적으로 적용되고 있다. 그러나 다양한 제곱합동법이 존재함에도 불구하고 아직까지도 많은 RSA 수들이 해독되지 않고 있다. 본 논문은 $\phi(n)$을 직접 구하는 알고리즘을 제안하였다. 제안된 알고리즘은 $2^j\equiv\beta_j(\mathrm{mod}\,n)$, $2^{\gamma-1}<n<2^\gamma$, $j=\gamma-1,\gamma,\gamma+1$에 대해 $2^k\beta_j\equiv 2^i(\mathrm{mod}\,n)$, $0\le i\le\gamma-1, k=1,2,\cdots$또는 $2^k\beta_j=2\beta_j$로 $\phi(n)$을 구하였다. 제안된 알고리즘은 $n-10\lfloor\sqrt{n}\rfloor<\phi(n)\le n-2\lfloor\sqrt{n}\rfloor$의 임의의 위치에 존재하는 $\phi(n)$도 약 2배 차이의 수행횟수로 찾을 수 있었다.

▸ Keywords : 오일러 totient 함수, 소인수분해, 모듈러 지수연산법, 이산대수법, 충돌

Abstract

There is to be virtually impossible to solve the very large digits of prime number p and q from composite number $n=pq$ using integer factorization in typical public-key cryptosystems, RSA. When the public key e and the composite number n are known but the private key d remains unknown in an asymmetric-key RSA, message decryption is carried out by first obtaining $\phi(n)=(p-1)(q-1)=n+1-(p+q)$and then using a reverse function of $d=e^{-1}(\mathrm{mod}\,\phi(n))$.Integer factorization from n to p,q is most widely used to produce $\phi(n)$, which has been regarded as mathematically hard. Among various integer factorization methods, the most popularly used is the congruence of squares of $a^2\equiv b^2(\mathrm{mod}\,n), a=(p+q)/2, b=(q-p)/2$which is more commonly used then

•제1저자 : 이상운
•투 고 일 : 2014. 05. 21, 심사일 : 2014. 06. 26, 게재확정일 : 2014. 07. 09.
* 강릉원주대학교 멀티미디어공학과 (Dept. of Multimedia Eng., Gangneung-Wonju National University)

$n/p=q$ trial division. Despite the availability of a number of congruence of scares methods, however, many of the RSA numbers remain unfactorable. This paper thus proposes an algorithm that directly and immediately obtains $\phi(n)$. The proposed algorithm computes $2^k\beta_j \equiv 2^i \pmod n$, $0 \le i \le \gamma-1, k=1,2,\cdots$ or $2^k\beta_j = 2\beta_j$ for $2^j \equiv \beta_j \pmod n$, $2^{\gamma-1} < n < 2^\gamma$, $j=\gamma-1,\gamma,\gamma+1$ to obtain the solution. It has been found to be capable of finding an arbitrarily located $\phi(n)$ in a range of $n-10\lfloor\sqrt{n}\rfloor < \phi(n) \le n-2\lfloor\sqrt{n}\rfloor$ much more efficiently than conventional algorithms.

▸ Keywords : Euler's totient function, integer factorization, modular exponentiation, discrete logarithm, collide

I. 서 론

대표적인 비대칭키 RSA의 n은 유사하거나 동일한 길이의 2개 소수 p와 q를 선택하여 곱한 값 $n=pq$로 쉽게 계산되는 합성수 (composite number)이다. 일단 n을 얻고 난 후에는 p,q를 없애 공개키 n을 만든 사람조차도 p,q를 알 수 없다. 비대칭키를 사용하는 이유는 n을 p,q로 소인수분해 (integer factorization)하기 어렵다는 수학적 난제에 기반하고 있다[1-3].

RSA의 암호를 해독하기 위해 $q=n/p$의 나눗셈 시행법 (trial division)이나 $\alpha^2 \equiv \beta^2 \pmod n, \alpha=(p+q)/2, \beta=(q-p)/2$의 제곱 합동법 (congruence of squares)으로 p,q를 구하여 다시 $\phi(n)$을 구하는 간접법을 적용하고 있다[4,5].

십진수 길이가 100자리 이상인 n만 알고 있는 상태에서 나눗셈 시행법으로 소인수분해하기 위해 최신 컴퓨터를 활용해도 80년 이상이 소요되어 현실적으로 거의 불가능한 것으로 알려져 있다.

제곱합동법을 적용하는 방법에는 2차 체 (quadratic Sieve), MPQS (multiple-polynomial quadratic sieve), NFS (Number field sieve), GNFS (General number field), Dixon, CFRAC (continued fraction factorization), SQUFOF (Shanks' square forms factorization) 등이 있다[6]. 지금까지 소인수 분해된 RSA 수들을 고찰해보면 대부분 GNFS를 적용하였으나 아직까지 해독되지 않은 RSA 수들이 많이 존재하고 있다[7].

RSA 암호체계에서 송신자는 메시지 m에 대해 $e<n, \gcd(e,\phi(n))=1$을 선택하여 암호 코드 $c=m^e \pmod n$로 변환시켜 전송하며, 수신자는 개인키 d를 이용하여 $m=c^d \pmod n$로 해독한다[1-3]. 여기서 $ed \pmod{\phi(n)}=1$이며, (e,n)은 공개된다. $\phi(n)$은 n과 서로소인 개수로 오일러 totient 함수이다[8]. RSA를 적용하는 암호에서 $\phi(n)$만 알고 있으면 식 (1)에 의해 개인키 d를 해독할 수 있다.

$$ed \pmod{\phi(n)}=1,\ d=e^{-1} \pmod{\phi(n)} \tag{1}$$

소수 p에 대해 $\phi(p)=p-1$, $\phi(p)/2=(p-1)/2$로 쉽게 구할 수 있다. 그러나 합성수 n의 $\phi(n)$은 식 (2)와 같다. 여기서 $(p+q-1)$는 p,q의 배수 개수로 n개에서 소인수 p,q의 배수 개수를 빼면 n과 서로소인 원소의 개수 $\phi(n)$이 구해진다.

$$\begin{aligned}\phi(n) &= (p-1)(q-1)\\ &= pq-(p+q-1)=n-(p+q-1)\\ (p+q) &= (n+1)-\phi(n)\end{aligned} \tag{2}$$

합성수 n에 대해서는 임의의 수 $\alpha(\alpha<n)$에 대해 식 (3)이 성립한다.

$$\alpha^{\phi(n)} = \alpha^{\phi(n)/2} = \alpha^0 \equiv 1 \tag{3}$$

$\alpha^\gamma \equiv \beta \pmod n$에서 α,β,n이 주어졌을 때 γ를 구하는 문제는 $\gamma=\log_\alpha\beta$로 풀 수 있어 이산대수 (discrete logarithm)라 한다[2]. 이산대수 알고리즘을 적용하면 $\phi(n)$을 구할 수 있다. 이산대수도 수학적 난제로 대칭키 알고리즘에 적용되고 있다. 대표적인 이산대수 알고리즘으로는 아기걸음-거인걸음 (Baby-step Giant-step)과 Pollard의 rho 알고리즘으로도 이산대수 문제를 $O(\sqrt{n})$회 수행하여야 해를 구한다[9].

본 논문은 아기걸음-거인걸음 알고리즘에서 채택한 개념을 적용하여 $\phi(n)$을 찾는 알고리즘을 제안한다. 2장에서는 아기걸음-거인걸음법을 적용하여 $\phi(n)$을 찾는 방법을 고찰한다. 3장에서는 $\phi(n)$을 찾는 알고리즘을 제안하고 4장에서는 제안된 알고리즘의 성능을 검증하여 본다.

II. 이산대수 적용 $\phi(n)$ 탐색법

아기걸음-거인걸음 알고리즘은 그림 1과 같이 n을 $m=\lceil\sqrt{n}\rceil$의 거인걸음 보폭으로 분할하고, 아기걸음 단계에서는 거인걸음 보폭 $[\alpha^0,\alpha^{m-1}]$에 대해 아기걸음으로 보폭 1인 m걸음의 모듈러 지수연산을 수행한다. 다음으로 거인걸음 시작점인 $\alpha^m(\mathrm{mod}\,n)$의 역함수 $\alpha^{-m}(\mathrm{mod}\,n)$을 유클리드 알고리즘으로 구한다. 마지막으로 거인걸음 단계에서는 보폭 m으로 j번째 걸음에서 아기걸음의 i번째 값과 일치하면, $\gamma=jm+i$로 결정한다[10,11]. 이는 식 (4)로 증명된다.

$$\alpha^{\gamma}\times(\alpha^{-m})^j=\alpha^i,\ \alpha^{\gamma-jm}=\alpha^i,\ \alpha^{\gamma}=\alpha^{mj+i},\ \gamma=mj+i \qquad (4)$$

이 방법은 미지의 γ부터 시작하여 보폭 m의 $-j$ 걸음을 걸어가 0 $(\alpha^0=1)$ 근방의 i를 찾아 $\gamma=jm+i$를 계산하는 방식이며, $\gamma/mj\neq 0$가 되지 못하는 경우가 대부분으로 $-j$번째 걸음이 $[0,m-1]$의 i번째와 일치하면 i만큼 더해주는 방식이다.

```
입력 : α,β,n , 출력 : γ
α^γ ≡ β(mod n)
m = ⌈√n⌉  /* 거인 보폭 결정 */

/* Baby-step: 수행복잡도 O(√n)
for        i = 0 to m − 1
           β_i = α^i(mod n) 계산, (i, c_i) 저장.
end
/* 거인걸음 시작점 결정 */
β_m = α^m(mod n)에 대해 β^{-1} = (β_m)^{-1}(mod n) 계산

/* Giant-step: 수행복잡도 O(√n) */
j = 0, β_j = β.
if         β_j = β_i then γ = i, 알고리즘 종료.
for        j = 1 to m
           β_j = β_{j-1} × β^{-1}(mod n) 계산
                  /* β_j = β × (β^{-1})^j(mod n) */
           if      β_j = β_i then γ = mj + i
           else if β_j ≠ β_i then j = j + 1, continue.
end
```

그림 1. 아기걸음-거인걸음 알고리즘
Fig. 1. Baby-step Giant-step Algorithm

아기걸음-거인걸음 알고리즘의 문제점은 아기걸음단계에서 구한 $m=\lceil\sqrt{n}\rceil$개의 모듈로 지수연산값을 계산하여 저장하고 있어야만 하며, 이로 인해 각 j걸음에서 탐색하는데 과다한 시간과 메모리를 필요로 한다는 점이다. 만약, RSA-100과 같이 n이 십진수 100자리이면 $m=\lceil\sqrt{n}\rceil$은 십진수 50자리로 이 데이터를 모듈로 지수연산 계산, 저장 및 탐색에는 현실적으로 불가능하다고 할 수 있다.

합성수 n에 대해 $\phi(n)=(n+1)-(p+q)$에서 $\phi(n)$은 $n-10\lfloor\sqrt{n}\rfloor<\phi(n)\le n-2\lfloor\sqrt{n}\rfloor$에 존재한다. 따라서 $\phi(n)$을 구하려면 상한값 $i_u=n-2\lfloor\sqrt{n}\rfloor$에서 시작하여 거인걸음 보폭 $m=\lceil\sqrt{n}\rceil$을 성인걸음 보폭 $\gamma, 2^{\gamma-1}<n<2^{\gamma}$로 줄여 저장 메모리 용량을 획기적으로 줄이면서 찾는 아기걸음-성인걸음법을 적용하여 해를 구할 수도 있다. 그러나 이 방법은 $10\lfloor\sqrt{n}\rfloor/l$의 걸음수로 증가하여 $\phi(n)$이 i_u에서 멀리 떨어진 하한값 근방에 있으면 느리게 찾는 단점이 있다.

$\alpha^{\gamma}\equiv\beta(\mathrm{mod}\,n)$에서 γ를 구하는 방법으로는 이외에도 Lee[12,13]이 있다. 그러나 $\phi(n)$을 찾는 알고리즘은 제안되지 않고 있다.

III. $2^k\beta$ 이용 $\phi(n)$ 탐색법

본 장에서는 아기걸음-거인걸음 알고리즘을 변형시킨 방법으로 이산 대수를 계산하여 $\phi(n)$을 찾는 방법을 제안한다. 제안된 방법은 $\phi(n)$을 찾기 위해 $2^{\gamma-1}<n<2^{\gamma}$인 $2^{\gamma-1}\equiv\beta_{\gamma-1}(\mathrm{mod}\,n)$, $2^{\gamma}\equiv\beta_{\gamma}(\mathrm{mod}\,n)$과 $2^{\gamma+1}\equiv\beta_{\gamma+1}(\mathrm{mod}\,n)$의 $\beta_{\gamma-1}$, β_{γ}과 $\beta_{\gamma+1}$을 사용해 다음 2가지 기준을 적용한다.

[기준 1] $2^k\beta_j=\beta_i$, $\beta_i=2^i(\mathrm{mod}\,n), 0\le i\le\gamma, k=1,2,\cdots, j=[\gamma-1,\gamma+1]$를 찾는다.

$\beta_i=2^i(\mathrm{mod}\,n), 0\le i\le\gamma$를 구하는 방법은 아기걸음-거인걸음법의 아기걸음 단계를 적용해 $i=[0,\gamma]$, $\alpha=2$에 대해 $\alpha^i\equiv\beta_i(\mathrm{mod}\,n), \beta_i=(\beta_{i-1})\alpha(\mathrm{mod}\,n)$을 계산하여 (i,β_i)를 저장한다. $2^k\beta_j(\mathrm{mod}\,n)=(2^{k-1}\beta_j)^2(\mathrm{mod}\,n)$으로 쉽게 구할 수 있다.

[기준 2] $2^k\beta_j=2\beta_j$의 충돌쌍을 찾는다.

〔기준 2〕는 이산대수 분야의 사이클 검출 (cycle detection) 방법을 적용하였다. 대표적인 사이클 검출 방법에는 Floyd, Pollard, Brent, Yao 등이 있으며, 대표적인 Floyd 방법은 2개의 포인터 (거북이와 토끼)를 사용해 동일 지점에서 출발하여 거북이는 정상적인 속도로, 토끼는 거북이의 2배 속도로 가면서 충돌을 검출하는 방법이다. 따라서 〔기준 2〕는 3마리 토끼가 $\gamma-1, \gamma$과 $\gamma+1$의 보폭 속도로 달려가는 방법으로 2β값과 충돌 (일치)하면 멈추는 방법으로 변형된 형태이다.

위의 〔기준 1〕과 〔기준 2〕중 어느 하나를 만족하면 알고리즘을 종료하며, 그렇지 않으면 $k=k+1$로 하여 이 기준이 충족될 때까지 계속적으로 찾는다.

만약, $2^k\beta_j=\beta_i$이면 식 (5)로, $2^k\beta_j=2\beta_j$이면 식 (6)으로 $m\phi(n)$을 구한다.

$$m\phi(n)=(2^k-1)\times j+[j-i] \quad (5)$$

$$m\phi(n)=2^k j-2j \quad (6)$$

$m\phi(n)$에 대해 $j_1=n-10\lfloor\sqrt{n}\rfloor$과 $j_2=n-2\lfloor\sqrt{n}\rfloor$으로 설정하여 $l=[m\phi(n)/j_1, m\phi(n)/j_2]$의 범위를 구하여 0.25 간격 값으로 나눈 값의 소수점 아래 자리가 $0, 0.25, 0.5, 0.75$이면 $\phi(n)$을 구할 수 있다. 제안된 알고리즘은 그림 2에 제시되어 있다.

입력 : n, 출력 : $\phi(n)$

$\alpha^\gamma\equiv\beta(\bmod n)$, $\alpha=2, 3$
/* Baby-step: 수행복잡도 $O(\gamma)$ */
for $i=0$ to $\gamma+1$, $(\alpha^{\gamma-1}<n<\alpha^r)$
$\beta_i=\alpha^i(\bmod n)$ 계산, (i, c_i) 저장.
/* $\beta_i=(\beta_{i-1})\alpha(\bmod n)$ */
end

$j=\gamma-1, \gamma, \gamma+1$
$2\beta_j(\bmod n)=(\beta_j)^2(\bmod n)$ 계산, 저장.

for $k=2$ to $\sqrt{n}$
$2^k\beta_j(\bmod n)=(2^{k-1}\beta_j)^2(\bmod n)$ 계산
if $2^k\beta_j=\beta_i$ then
$m\phi(n)=(2^k-1)\times j+(j-i)$
else if $2^k\beta_j=2\beta_j$ then
$m\phi(n)=2^k j-2j$
else $k=k+1$, continue.
end
$j_1=n-10\lfloor\sqrt{n}\rfloor$, $j_2=n-2\lfloor\sqrt{n}\rfloor$
$l=[m\phi(n)/j_1, m\phi(n)/j_2]$의 범위를 구함.
$\phi(n)=m\phi(n)/l$ 범위의 0.25 간격 값. (소수점 아래 자리가 $0, 0.25, 0.5, 0.75$)

그림 2. $2^k\beta$의 $\phi(n)$ 알고리즘
Fig. 2. $2^k\beta$'s $\phi(n)$ Algorithm

IV. 적용 결과 및 분석

$(n+1)-10\lfloor\sqrt{n}\rfloor<\phi(n)<(n+1)-2\lfloor\sqrt{n}\rfloor$에 대해 $(n+1)-10\lfloor\sqrt{n}\rfloor, (n+1)-5\lfloor\sqrt{n}\rfloor$과 $(n+1)-2\lfloor\sqrt{n}\rfloor$ 근방에 위치한 표 1의 3개 데이터에 대해 실험을 수행한다.

표 1. 실험 데이터의 $\phi(n)$ 분포
Table 1. $\phi(n)$ distribution of example data

n	$\phi(n)$	$\lfloor\sqrt{n}\rfloor$	$\phi(n)$	
			하한값	상한값
10967	9960 (하한 근방)	104	9927	10759
2743	2520 (중간 위치)	52	2223	2639
3397	3276 (상한 근방)	58	2817	3281

여기서는 $\phi(n)$이 위 범위의 어느 곳에 위치하고 있는지를 보여주고 있다. 실험 데이터에 대해 아기걸음-거인걸음법, $\phi(n)$ 상한값 기준 아기걸음-성인걸음법과 $2^k\beta$법의 수행결과는 표 2에 제시되어 있다. 각 알고리즘의 수행 횟수를 비교한 결과는 표 3에 제시되어 있다. 실험은 Excel을 이용하여 계산 하였다.표 2에서는 $\phi(n)$을 찾기 위한 각 알고리즘의 걸음수 (수행횟수)를 표현하고 있다. 여기서, j는 거인걸음 수, i는 아기걸음 수이며, 실제의 $j+i$를 찾기 위해서는 $j+1$의 거인걸음수가 필요하다.

표 2. 실험 데이터의 $\phi(n)$ 탐색 결과
Table 2. The result of $\phi(n)$ search for example data
(a) 아기걸음-거인걸음법

n	$\phi(n)$	$\lceil\sqrt{n}\rceil$	j	i
10967	9960	105	**94**	90
2743	2520	53	**47**	29
3397	3276	59	**56**	28

(b) $\phi(n)$ 상한값 기준 아기걸음-성인걸음법

n	$\phi(n)$	상한값	γ	j	i
10967	9960	10759	14	**57**	1
2743	2520	2639	12	**9**	11
3397	3276	3281	12	**0**	5

(c) $2^k\beta$법

n	$\phi(n)$	$2^k\beta_j=\beta_i$						$2^k\beta_j=2\beta_j$		
		$\gamma-1$		γ		$\gamma+1$		$\gamma-1$	γ	$\gamma+1$
		k	i	k	i	k	i	k	k	k
10967	9960	**7**	4							
2743	2520								**13**	
3397	3276			**11**	6			13	13	**7**

표 3. $\phi(n)$ 탐색 수행횟수 비교
Table 3. Comparison of the number of execution for $\phi(n)$

n	$\phi(n)$	아기걸음-거인걸음법	$\phi(n)$ 상한값 기준 아기걸음-성인걸음법	$2^k\beta$법
10967	9960	95	58	**7**
2743	2520	48	10	**13**
3397	3276	57	**1**	**7**

아기걸음-거인걸음법은 $m=\lceil n \rceil$ 개의 모듈러 지수연산 값 저장과 역함수를 구하는 단점이 있으며, $\phi(n)$이 m에서 멀어질수록 수행횟수가 증가하며, 가장 많은 수행횟수를 보이고 있다. $\phi(n)$ 상한값 기준 아기걸음-성인걸음법은 역함수를 구하는 단점이 있으며, 상한값 $i_u=(n+1)-2\lfloor\sqrt{n}\rfloor$ 근방에 위치한 $\phi(n)$은 빠르게 찾는 반면에 상한값에서 역으로 멀어질수록 수행횟수가 증가하는 경향을 보인다. 이에 반해, $2^k\beta$법은 역함수를 구하지 않는 장점이 있으며, $(n+1)-10\lfloor\sqrt{n}\rfloor<\phi(n)<(n+1)-2\lfloor\sqrt{n}\rfloor$ 의 범위에 대해 양 끝단에서는 동일한 수행횟수를 보이며, 중심인 $(n+1)-5\lfloor\sqrt{n}\rfloor$으로 가면서 양 끝단의 약 2배 정도의 수행횟수를 보이고 있다. 즉, 제안된 방법은 $(n+1)-10\lfloor\sqrt{n}\rfloor<\phi(n)<(n+1)-2\lfloor\sqrt{n}\rfloor$ 의 어느 곳에 위치하던지간에 유사한 수행횟수로 찾을 수 있음을 알 수 있다.

V. 결론

$p\times q=n$에 대한 암호해독을 위해서는 p나 q를 찾는 소인수 분해법인 제곱합동법인 2차 체, MPQS, NFS, GNFS, Dixon, CFRAC, SQUFOF 등 다양한 방법들이 제안되었음에도 불구하고 RSA의 소수 쌍 들을 찾는데 대부분 실패하였다.

본 논문은 $\phi(n)=(n+1)-(p+q)$에 대해 $\phi(n)$ 을 찾는 것이 소인수 p나 q를 찾는 것에 비해 보다 빠르게 암호를 해독할 수 있는 방법임에 착안하였다.

따라서, 본 논문에서는 합성수 n을 사용하는 비대칭키 RSA의 $\phi(n)$을 $a^{\phi(n)}=a^{\phi(n)/2}\equiv 1(\mathrm{mod}\, n)$을 찾는 지수연산법 알고리즘을 제안하였다.

제안된 알고리즘은 지수연산법 중에서 일반적으로 알려진 아기걸음-거인걸음법과 이산대수법의 사이클 검출법을 혼합한 하이브리드형을 적용하였다. 세부적으로는 아기걸음-거인걸음법의 아기걸음 단계와 이산대수의 사이클 검출법을 적용하였으며, $2^{\gamma-1}<n<2^{\gamma}$의 $2^j\equiv\beta_j(\mathrm{mod}\, n), j=\gamma-1,\gamma,\gamma+1$의 β_j에 대해 $2^k\beta_j$로 $\phi(n)$을 구하였다.

제안된 방법은 $\lfloor\sqrt{n}\rfloor<\phi(n)<(n+1)-2\lfloor\sqrt{n}\rfloor$의 어느 곳에 위치하던지 간에 약 2배 차이의 수행횟수로 $\phi(n)$을 구할 수 있음을 보였다.

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

[3] B. Raiter, "How the RSA Cipher Works", http://www.tutorialized.com/tutorial/How-the-RSA-Cipher-Works/42395, 2009.

[4] M. Seysen, "A probabilistic factorization algorithm with quadratic forms of negative discriminant", Mathematics of Computation, Vol. 48, No. 178, pp. 757-780, Apr. 1987.

[5] C. P. Schnorr, "Refined analysis and improvements on some factoring algorithms", Journal of Algorithms, Vol. 3, No. 2, pp. 101-127, Jun. 1982.

[6] Wikipedia, "Integer Factorization," http://en.wikipedia.org/wiki/Integer_factorization, 2014.

[7] Wikipedia, "RSA Factoring Challenge," http://en.wikipedia.org/wiki/RSA_Factoring_challenge, 2014.

[8] K. Ford, "The Number of Solutions of $\phi(x)=m$", Annals of Mathematics, Vol. 150, No. 1, pp. 283-311, Jan. 1999.

[9] A. A. Razborov and S. Rudich, "Natural proofs", Journal of Computer and System Sciences, Vol. 55, No. 1, pp. 24-35, Aug. 1997.

[10] 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.

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

[12] S. U. Lee, "Square-and-Divide Modular

Exponentiation," Journal of Korea Society of Computer Information, Vol. 18, No. 4, pp. 123-129, Apr. 2013.

[13] S. U. Lee, "Modified Baby-Step Giant-Step Algorithm for Discrete Logarithm," Journal of Korea Society of Computer Information, Vol. 18, No. 8, pp. 87-93, Aug. 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,
그래프 알고리즘
e-mail : sulee@gwnu.ac.kr