http://dx.doi.org/10.7236/JIWIT.2012.12.5.185
JIWIT 2012-5-23

제곱합동 기반 소인수분해법

The Integer Factorization Method Based on Congruence of Squares

이상운*, 최명복**

Sang-Un Lee, Myeong-Bok Choi

요 약 큰 반소수 $n=pq$의 소인수 p,q를 직접 찾는 것은 현실적으로 거의 불가능하여 대부분의 소인수분해 알고리즘은 $a^2 \equiv b^2 \pmod n$의 제곱합동을 찾아 $p=GCD(a-b,n), q=GCD(a+b,n)$의 소인수를 찾는 간접 방법을 적용하고 있다. 제곱합동 a,b을 찾는 다양한 방법이 제안되었지만 100자리 이상인 RSA 수에 대해서는 적용이 쉽지 않다. 본 논문에서는 $xa=\lceil \sqrt{zn} \rceil$ or $\lceil \sqrt{zn} \rceil + z, z=1,2,\cdots$로 설정하고 $(xa)^2 \equiv (yb)^2 \pmod n$을 찾는 간단한 방법을 제안한다. 제안된 알고리즘은 19 자리 수 까지는 제곱합동을 빠르게 찾는데 성공하였으나 39 자리 수에 대해서는 실패하였다.

Abstract It is almost impossible to directly find the prime factor, p,q of a large semiprime, $n=pq$. So Most of the integer factorization algorithms uses a indirect method that find the prime factor of the $p=GCD(a-b,n), q=GCD(a+b,n)$ after getting the congruence of squares of the $a^2 \equiv b^2 \pmod n$. Many methods of getting the congruence of squares have proposed, but it is not easy to get with RSA number of greater than a 100-digit number. This paper proposes a fast algorithm to get the congruence of squares. The proposed algorithm succeeded in getting the congruence of squares to a 19-digit number.

Key Words : prime number, semiprime, composite number, Sieve, Trial Division, congruence of squares

Ⅰ. 서 론

RSA 암호체계 (cryptograph)의 공개키 $n=pq$를 생성하기 위해서는 합성수 n이 2개 소수 p,q의 곱인 반소수 (semiprime)가 되어야 한다. 여기서 p,q의 소수 여부는 소수 판별법 (primality test, PT)을 적용한다. 또한, RSA 암호 해독은 반소수 n을 p,q로 소인수분해 (factorization)하여야 한다.[1-6]

2개의 소수 곱 $p \times q$으로 반소수 n을 생성하는 것은 쉬운 반면에, 역으로 n이 주어졌을 때 이를 2개의 소인수 p,q로 인수분해하는 것은 어렵다. 소인수분해 방법에는 나눗셈 시행 (Trial Division), Pollard의 rho 알고리즘, Pollad의 p-1 알고리즘, William의 p+1 알고리즘, Lenstra elliptic curve 인수분해, 페르마 (Fermat)의 인수분해 방법, 오일러 (Euler)의 인수분해 방법, Special Number Field Sieve, 2차 (Quadratic, Q), MPQ (Multiple-polynomial quadratic), NF (Number field), GNF (General number field), Dixon, CFRAC (continued

*정회원, 강릉원주대학교, 멀티미디어공학과
**종신회원, 강릉원주대학교 멀티미디어공학과
접수일자 : 2012년 6월 24일, 수정완료 : 2012년 8월 28일
게재확정일자 : 2012년 10월 12일

Received: 24 June 2012 / Revised: 28 August 2012 / Accepted: 12 October 2012
**Corresponding Author: cmb5859@gmail.com
Dept. of Multimedia Engineering, Gangnung-Wonju National University Wonju Campus, Korea

fraction factorization), SQUFOF (Shanks' square forms factorization)과 양자 컴퓨터를 활용한 Shor 알고리즘이 있다.[7]

나눗셈 시행법은 $n \equiv 0 (\mathrm{mod}\, p), 2 \le p < \sqrt{n}$ 의 나눗셈으로 p를 찾는 직접 방식이다. n이 큰 수인 경우 나눗셈 시행법은 비현실적으로 과다한 시간이 소요된다. 따라서 일반적으로 체 (Sieve) 방법이 사용되고 있다. 체 방법은 $a^2 - b^2 \equiv 0 (\mathrm{mod}\, n)$의 제곱 합동 (congruence of squares)인 a,b를 결정하고 $p = GCD(a-b,n)$, $q = GCD(a+b,n)$을 구하는 간접 방식이다. 그러나 a,b를 결정하는 과정도 쉽지 않다. 왜냐하면, RSA Lab.에서 1991년에 RSA Factoring Challenge를 결성하여 총 $687,563의 상금을 걸었지만 지금까지 RSA-100부터 RSA-200의 일부 데이터만 소인수분해에 성공하여 $82,563의 상금을 지급하였고, $605,000의 상금이 철회된 상태로 2007년에 해체되었다.[8] 이후, 2009년에 T. Kleinjung et al.이 $50,000 상금이 걸린 RSA-768을 소인수분해에 성공하였으나 상금은 수령하지 못하였다. 결국, RSA 수의 소인수분해 문제는 현재 진행형이며, 다항시간 알고리즘을 찾지 못할 수도 있다.

본 논문은 $(xa)^2 - (yb)^2 \equiv 0 (\mathrm{mod}\, n)$의 제곱합동을 간단히 찾는 방법을 제안한다.

2장에서는 소인수분해법을 고찰해 본다. 3장에서는 제곱합동을 간단히 구하는 방법을 제안한다. 4장에서는 제안된 알고리즘의 적합성을 검증해 본다.

II. 반소수의 소인수 분해법

RSA 보안체계에 적용되는 합성수 n은 홀수 반소수로 반드시 2개의 소인수 p,q만을 가진다. 주어진 수 $n = p \times q$로부터 직접 소인수 p,q를 직접 구하는 방법은 나눗셈 시행법[9]이 있다. 나눗셈 시행법은 $n/p = q$ 공식에 의거 $2 \le p < \sqrt{n}$의 범위에서 소수들만을 대상으로 $n/p = q$ 또는 $n = 0 (\mathrm{mod}\, p)$를 찾아야 하나 큰 자리수에 대해서는 현실적으로 불가능하여 홀수들을 대상으로 한다. $n = pq$의 특징은 n,p,q는 모두 홀수이다. 왜냐하면 소수(홀수)×소수(홀수)=합성수(홀수)가 되기 때문이다. 또한, n,p,q의 1의 자리는 1,3,7 또는 9이며, $6k \pm 1, k = 1,2,\cdots$이다. $n = 6k \pm 1, (n_1 = \{1,3,7,9\})$라면 $7 \le p < \sqrt{n}$, $(p = 6k \pm 1, p_1 = \{1,3,7,9\})$로 한정시켜, $m = 1,2,\cdots$, $\sqrt{n}/6$에 대해 $p = n/\sqrt{36m^2 \pm 12m + 1}$, $q = n/p$로 결정된다.

Shor 알고리즘을 제외한 대부분의 간접 소인수분해 알고리즘은 제곱합동을 찾는 페르마 알고리즘[10]에 기반을 두고 있다. 제곱합동 알고리즘들은 항상 $a^2 - b^2 = n$이 되는 (a,b) 쌍을 찾는다. 페르마 알고리즘은 항상 $a^2 - b^2 = n$의 1개 쌍을 순차적 또는 랜덤하게 찾는 방법이다. 그러나 1개의 $a^2 \equiv b^2 (\mathrm{mod}\, n)$을 찾는 것은 쉽지 않다. 페르마 알고리즘을 적용할 경우 주의할 점은 $a = \lceil 0.5n \rceil$, $b = a - 1$로 결정하면 $6k \pm 1$의 모든 합성수나 소수는 $a^2 - b^2 = n,\ a - b = 1, a + b = n$이 되므로 p,q를 얻지 못한다. 따라서 $\lceil \sqrt{n} \rceil \le a \le \lfloor 0.5n \rfloor$으로 설정하여야 한다.

$a_0 = \lceil \sqrt{n} \rceil, r_0 = a_0^2 - n, d = a - a_0,\ a = a_0 + d$라 하면 $(a_0 + 1)^2 - a_0^2 = 2a_0 + 1$이 성립하므로 식 (1)로 표현된다.

$$a^2 - b^2 = n, (a-b)(a+b) = n, (a-b)a + (a-b)b = n$$
$$a^2 - a_0^2 + r_o = b^2,\ (a + a_0)d + r_0 = b^2 \text{ or}$$
$$d^2 + 2a_0 d + r_0 = b^2, (d + 2a_0)d + r_0 = b^2. \quad (1)$$

d를 페르마의 순차적 방법을 적용하면 수행횟수는 d이다. 예를 들면, $n = 18,206,927$의 경우 $a_0 = 4267$, $r_0 = 362$로 $a = 5676, b = 3743, d = 1409$이다. 즉, $(5676 + 4267) \times 1409 + 362 = 3743^2$ 또는 $1409^2 + 2 \times 4267 \times 1409 + 362 = 3743^2$이 되어 $d = 1409$회를 수행해야 한다. $d^2 + 2a_0 d + r_0 = b^2$는 $d^2 + 2a_0 d + (r_0 - b^2) = 0$의 이차방정식이 된다. 근의 공식에 의거 $d = -a_0 \pm \sqrt{b^2 - r_0 + a_0^2} = -a_0 + \sqrt{b^2 + n} = a - a_0$이 되나 b를 모르기 때문에 d를 구할 수 없다.

Dixon[11]과 2차 체[12], GNFS[13] 등의 방법은 페르마 알고리즘의 1개 제곱합동 쌍을 찾는 어려움을 개선한 형태이다. 이들 알고리즘은 데이터 수집 단계 (data collection phase)와 데이터 처리 단계 (data processing phase)를 수행한다. Dixon 알고리즘은 데이터 수집 단계에서 $a_1^2 \equiv r_1 (\mathrm{mod}\, n),\ a_2^2 \equiv r_2 (\mathrm{mod}\, n)$의 2개 쌍을 찾는다. 여기서 r_1과 r_2는 $r_1 \ne b^2, r_2 \ne b^2$이지만 동일한 작은 소인수들의 집합 B로 구성되어 있다. 이를 $B-smooth$

또는 소인수 기준(factor base, FB)이라 한다. $(r_1 \times r_2) = b^2$이 되려면 $r_1 \times r_2$의 소인수들의 지수항이 짝수가 되어야 한다. 즉, $a_1 \times a_2 = a, r_1 \times r_2 = b$를 얻어 $a^2 - b^2 = (a+b)(a-b)$ 공식에 기반하여 유클리드 최대 공약수 (Great Common Divisor, GCD) 법칙을 적용하여 $p = GCD(a-b, n)$, $q = GCD(a+b, n)$을 결정한다. 실제로 a 값을 랜덤하게 선택하여 $B-smooth$를 찾는 것은 비현실적으로 많은 시간과 메모리가 요구되어 대안으로 n보다 작은 몇 개만을 찾는다. Dixon 알고리즘은 a_1과 a_2를 랜덤하게 찾으며, FB를 결정하는 어려움이 존재한다. 예로, $n = 84923$의 경우, $B = 7, FB = \{2,3,5,7\}$로 설정하고, $\lceil \sqrt{n} \rceil = 292 \le a$인 임의의 수 $513^2 \pmod n = 8400 = 2^4 3^1 5^2 7^1$과 $537^2 \pmod n = 33600 = 2^6 3^1 5^2 7^1$을 구한다. $(513 \times 537)^2 \pmod n = 2^{10} 3^2 5^4 7^2 = (2^5 3^1 5^2 7^1)^2$이다. $(513 \times 537) \pmod n = 20712$, $2^5 3^1 5^2 7^1 = 16800$로 $20712^2 \pmod n = 16800^2$이 되어 a,b가 결정된다. Dixon 알고리즘 적용시 유의할 점은 $a_1 = \lceil \sqrt{n} \rceil$으로 설정할 경우, $a_2 = ka_1 - l, (k = 2,3,\cdots, l = 0,1 \text{ or } 2)$은 $(a_1 \times a_2)^2 \equiv b^2 \pmod n, b = \sqrt{(r_1 \times r_2)}$을 만족시키지만 p,q를 얻지 못해 제외시켜야 한다.

단순히 많은 난수를 제곱한 $a^2 \pmod n$은 수많은 다른 소인수들을 유발시킨다. 따라서 소인수들 FB가 $FB \le B$인 $B-$smooth들 만을 갖도록 하여야 하나 이 방법도 매우 어렵다. 2차 체 알고리즘[12]은 Dixon 알고리즘의 데이터 수집 단계를 개선한 것이다. 이 방법은 B를 먼저 결정하고 체 방법으로 $B-$smooth로 구성된 값들만을 가지는 $a^2 \pmod n$을 찾는다. 결국, 다수의 $a^2 \pmod n$을 찾아 2개 쌍의 곱이 제곱이 되는 것을 찾는다.

GNFS[13]는 100자리보다 큰 정수를 소인수분해하는 가장 효율적인 알고리즘으로 알려져 있으며, Rational Sieve[14]의 확장 형태이다. Rational Sieve는 먼저, 임의로 B를 결정하여 소인수 FB를 얻는다. 다음으로 z와 $z+n$이 B-smooth인 양의 정수 z를 찾아 각각에 대해 곱셈 관계 $\pmod n$을 계산한다. 이들 각각에 대해 다른 방법으로 결합시켜 지수항이 짝수가 되는 값을 결정하고 공통 인수를 삭제하여 a,b를 얻는다.

Ⅲ. 제곱합동 소인수분해 알고리즘

p,q와 a,b의 관계는 식 (2)와 같다. 예를 들면, $n = 973$, $\sqrt{n} = 31.19$의 경우, $p = 7, q = 139, a = 73$, $b = 66$이다.

$$p = a - b, q = a + b, a = \frac{p+q}{2}, b = \frac{q-p}{2} \qquad (2)$$

$n = pq$는 $[2,n]$에서 $|p| = q-1$, $|q| = p-1$개가 p 또는 q 간격으로 배치되어 있다. 반면에, $a^2 \equiv b^2 \pmod n$, $a^2 - b^2 = kn$은 표 1과 그림 1에서 알 수 있듯이 $\sqrt{n} < xa$는 $a, p = a-b$와 $q = a+b$의 일정한 간격으로 배치되어 있다. 또한, xa에 대해 $n \pm xa, 2n \pm xa, \cdots$도 성립한다. yb에 대해 xa와 동일한 개수가 존재한다. xa의 기준이 되는 a는 대부분 $\sqrt{n} < a < \sqrt{2n}$에 위치하고 있지만 $d = a - \sqrt{n}$의 거리를 알지 못해 수많은 xa 또는 yb가 존재함에도 불구하고 찾기가 쉽지 않다. 일반적으로 z는 홀수와 4의 배수이다

표 1. $a^2 \equiv b^2 \pmod n$**의** xa**와** yb **값**
Table 1. Values, xa and yb of $a^2 \equiv b^2 \pmod n$

$(xa)^2 - (yb)^2 = zn$	
$z = x^2$	$z = x^2 - y^2$
$(1a)^2 - (1b)^2 = 1n$	-
$(2a)^2 - (2b)^2 = 4n$	$xa = 2a \pm 1b,\ yb = 2b \pm 1a,\ 3n$ $yb = xa - (x \mp y)(a-b)$
$(3a)^2 - (3b)^2 = 9n$	$xa = 3a \pm 1b,\ yb = 3b \pm 1a,\ 8n$ $xa = 3a \pm 2b,\ yb = 3b \pm 2a,\ 5n$
$(4a)^2 - (4b)^2 = 16n$	$xa = 4a \pm 1b,\ yb = 4b \pm 1a,\ 15n$ $xa = 4a \pm 2b,\ yb = 4b \pm 2a,\ 12n$ $xa = 4a \pm 3b,\ yb = 4b \pm 3a,\ 7n$
$(5a)^2 - (5b)^2 = 25n$	$xa = 5a \pm 1b,\ yb = 5b \pm 1a,\ 24n$ $xa = 5a \pm 2b,\ yb = 5b \pm 2a,\ 21n$ $xa = 5a \pm 3b,\ yb = 5b \pm 3a,\ 16n$ $xa = 5a \pm 4b,\ yb = 5b \pm 4a,\ 9n$
...	...
$xa = \sqrt{z}\,a \pm kb, (0 \le k \le \sqrt{z}-1)$ $yb = \sqrt{z}\,b \pm ka, (0 \le k \le \sqrt{z}-1)$	

다수의 xa중 어느 하나를 찾기 위해서는 $(xa)^2 - zn = (yb)^2$이 성립하므로 $GCD(xa - yb, , n) = p$와 $GCD(xa + yb, n) = q$를 결정할 수 있다. 그러나 xa를 한 번에 결정하는 방법은 찾지 못하였다. 본 장에서는 대안으로 xa를 간단히 선택하는 방법을 그림 2와 같이 제안한다.

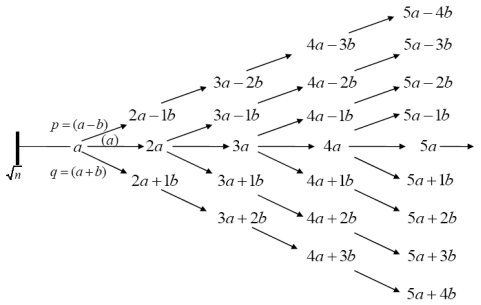

그림 1. $a^2 \equiv b^2 \pmod n$ **의** xa **값 분포**
Fig 1. Value Distribution with xa of $a^2 \equiv b^2 \pmod n$

for $z = 1, 2, \cdots$

$xa = \lceil \sqrt{zn} \rceil$ ..... (3)

$xa = \lceil \sqrt{zn} \rceil + z$ ..... (4)

if $(xa)^2 \equiv (yb)^2 \pmod n$ then $p = GCD(xa - yb, n)$, $q = GCD(xa + yb, n)$, 알고리즘 종료.

end

그림 2. 제곱합동 소인수분해 알고리즘
Fig 2. The Integer Factorization Algorithm Based on Congruence of Squares

Ⅳ. 알고리즘 적용성 평가

표 2. 소인수분해 실험 데이터
Table 2. Experiment Data for the Integer Factorization

n	$a^2 \equiv b^2 \pmod n$	$n = p \times q$
84,923	$a = 342$ $b = 179$	$p = 163$ $q = 521$
112,729	$a = 475$ $b = 336$	$p = 139$ $q = 811$
6,012,707	$a = 2,454$ $b = 97$	$p = 2,357$ $q = 2,551$
18,206,927	$a = 5,676$ $b = 3,743$	$p = 1,933$ $q = 9,419$
2,352,854,039	$a = 49,008$ $b = 6,995$	$p = 42,013$ $q = 56,003$
137,085,519,229	$a = 413,473$ $b = 184,050$	$p = 229,423$ $q = 597,523$
63,209,496,861,097	$a = 7,973,419$ $b = 604,908$	$p = 7,368,511$ $q = 8,578,327$
502,636,025,594,071	$a = 23,970,520$ $b = 8,482,323$	$p = 15,488,197$ $q = 32,452,843$
164,554,085,186,383,429	$a = 486,855,485$ $b = 269,210,286$	$p = 217,645,199$ $q = 756,065,771$
8,229,944,909,131,434,961	$a = 2,925,355,081$ $b = 572,501,040$	$p = 2,352,854,041$ $q = 3,497,856,121$
982,301,348,481,615,682,763,349,336,546,115,836,409	$a = 33,894,332,773,738,340,605$ $b = 1,904,435,117,249,313,796$	$p = 20,989,897,656,489,026,809$ $q = 46,798,767,890,987,654,401$

알고리즘 적용성을 평가하기 위해 표 2의 데이터에 적용하여 보자. 일부 데이터는 Jensen[15]의 pGNFS에서 인용되었다.

표 2의 데이터에 제안된 알고리즘을 적용한 결과는 표 3과 같다. 제안된 알고리즘은 Python Ver. 3.1 소프트웨어로 프로그램을 작성하여 결과를 도출하였다.

$n = 18,206,927$의 경우, 제안된 제곱합동 알고리즘은 $z = 5$에서 $xa = \lceil \sqrt{5n} \rceil = 9542$을 $(xa)^2 - zn = (yb)^2$에 대입하면 $9542^2 - 5 \times 18206927 = 123^2$으로 $yb = 123$을 얻어 알고리즘은 4회 (z =홀수+4의 배수) 수행되었다. 만약, Dixon의 체 알고리즘을 적용하면 $b = 3743 = 19^1 197^1$을 찾기 위해 $B = 197$, $FB = \{19, 197\}$을 선택해야 하는 어려움이 있다.

표 3. 소인수분해 알고리즘 수행횟수
Table 3. Running Time of the Integer Factorization Algorithm

n	$(xa)^2 \equiv (yb)^2 \pmod n$		
	수식	z	제곱합동
84,923	(3)	3	$xa = 505,\ yb = 16$
112,729	(3)	2	$xa = 475,\ yb = 336$
6,012,707	(4)	1	$xa = 2,454,\ yb = 97$
18,206,927	(3)	5	$xa = 9,542,\ yb = 123$
2,352,854,039	(3)	48	$xa = 336,061,\ yb = 43$
137,085,519,229	(3)	15,048	$xa = 45,418,751$ $yb = 7,003$
63,209,496,861,097	(3)	20,904	$xa = 1,149,491,767$ $yb = 4,051$
502,636,025,594,071	(3)	3,696	$xa = 1,362,990,371$ $yb = 29,035$
164,554,085,186,383,429	(3)	366,925	$xa = 245,721,402,623$ $yb = 27,048$
8,229,944,909,131,434,961	(3)	168,837	$xa = 1,178,778,693,659$ $yb = 1,180,882$
982,301,348,481,615,682,763,349,336,546,115,836,409			해를 구하지 못함

제안된 알고리즘은 19자리까지는 해를 빠르게 찾는데 성공하였으나 39자리 이상에서는 해를 얻지 못하였다. 이 데이터에 대한 해는 존재하는 것으로 판단되나 11자리까지의 z에 대해 처리하는데도 수 일이 소요되어 검증에는 실패하였다.

Ⅴ. 결론 및 향후 연구과제

본 논문은 제곱합동을 쉽게 찾는 방법을 제안하였다.

먼저, $a^2 \equiv b^2 (\mathrm{mod}\, n)$의 제곱 합동이 되는 xa와 yb는 다수가 존재함을 증명하였다. 다음으로 $xa = \lceil \sqrt{zn} \rceil$으로 설정하고 $(xa)^2 - zn = (yb)^2$을 계산하여 yb를 얻을 수 있었다. 결국, $p = GCD(xa - yb, n), q = GCD(xa + yb, n)$을 구할 수 있다. 제안된 알고리즘은 n이 19자리까지는 해를 빠르게 구하였으나 39자리에 대해서는 실패하였다. 따라서 보다 큰 자리수에 대해 제곱합동을 찾는 방법을 추후 연구할 계획이다.

참 고 문 헌

[1] Wikipedia, “Prime Number,” http://en.wikipedia.org/wiki/Prime_number, 2010.

[2] Wikipedia, “Prime Number Theorem,” http://en.wikipedia.org/wiki/Prime_number_theorem, 2010.

[3] Wikipedia, “Primality Test,” http://en.wikipedia.org/wiki/Primality_test, 2010.

[4] Wikipedia, “RSA,” http://en.wikipedia.org/wiki/Rsa, 2010.

[5] J. G. Kim, Y. S. Kim, S. O. Kim, "Factorization

[6] Algorithm," Journal of Korean Institute of Information Technology, v.8, no.2, pp.37-48, June, 1998.

[7] Wikipedia, “Integer Factorization,” http://en.wikipedia.org/ wiki/Integer_factorization, 2010.

[8] Wikipedia, “RSA Factoring Challenge,” http://en.wikipedia.org/wiki/RSA_Factoring_challenge, 2010.

[9] Wikipedia, “Trial Division,” http://en.wikipedia.org/wiki/Trial_Division, 2010.

[10] Wikipedia, “Fermat's Factorization Method," http://en.wikipedia.org/wiki/Fermat's_factorization_method, 2010.

[11] Wikipedia, “Dixon's Factorization Method," http://en.wikipedia.org/wiki/Dixon's_factorization_method, 2010.

[12] Wikipedia, “Quadratic Sieve," http://en.wikipedia.org/wiki/ Quadratic_sieve, 2010.

[13] Wikipedia, “General Number Field Sieve,” http://en. wikipedia.org/wiki/General_number_field_sieve, 2010.

[14] Wikipedia, “Rational Sieve,” http://en.wikipedia.org/wiki/ Rational_sieve, 2010.

[15] P. L. Jensen, “pGNFS,” http://pgnfs.org/, 2009.

저자 소개

이 상 운(정회원)

- 1983년 ~ 1987년 : 한국항공대학교 항공전자공학과(학사)
- 1995년 ~ 1997년 : 경상대학교 컴퓨터과학과(석사)
- 1998년 ~ 2001년 : 경상대학교 컴퓨터과학과(박사)
- 2003년 : 강원도립대학 컴퓨터응용과 전임강사
- 2004년 ~ 2007년 2월 : 국립 원주대학 여성교양과 조교수
- 2007년 3월 ~ 현재 : 강릉원주대학교 멀티미디어공학과 부교수

<주관심분야 : 소프트웨어 프로젝트 관리, 신경망, 뉴로-퍼지, 그래프 알고리즘 등>

최 명 복(종신회원)

- 1992년 : 호서대학교 전자계산학과(학사)
- 1994년 : 아주대학교 컴퓨터공학과(석사)
- 2001년 : 아주대학교 컴퓨터공학과(박사)
- 1997년 ~ 현재 : 강릉원주대학교 멀티미디어공학과 교수
- 2004년 1월 ~ 현재 : 한국인터넷방송통신학회 이사

<주관심분야 : 지능형 정보검색, 소프트웨어 공학, 알고리즘 등>