Journal of The Korea Institute of Information Security & Cryptology
VOL.26, NO.3, Jun. 2016
ISSN 1598-3986(Print)
ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.3.631

다항식 선택을 위한 효율적인 최적화 기법*

김 수 리,1† 권 희 택,1 이 용 성,1 장 남 수,2 윤 기 순,3 김 창 한,4‡ 박 영 호,2 홍 석 희1
1고려대학교 정보보호연구원, 2세종사이버대학교, 3엔에스에이치씨, 4세명대학교

Efficient Optimization Method for Polynomial Selection*

Suhri Kim,1† Heetaek Kwon,1 Yongseong Lee,1 Nam Su Chang,2 Kisoon Yoon,3
Chang Han Kim,4‡ Young-Ho Park,2 Seokhie Hong1
1Center for Information Security Technologies(CIST), Korea University,
2Sejong Cyber University, 3NSHC, 4Semyung University

요 약

현재까지 알려진 가장 효율적인 인수분해 방법은 General Number Field Sieve (GNFS)를 이용하는 방법이다. CADO-NFS는 GNFS를 기반으로 구현된 공개된 소프트웨어로 RSA-704의 인수분해에 사용된 도구이다. CADO-NFS에서 다항식 선택은 크게 다항식을 생성하는 과정과 이를 최적화하는 과정으로 나누어져 있다. 그러나 CADO-NFS에서 다항식의 최적화 과정은 전체 다항식 선택 소요 시간 중 약 90%를 차지할 정도로 큰 부하를 주고 있다. 본 논문에서는 사전 연산 테이블을 이용하여 다항식 최적화 과정의 부하를 줄이는 방안을 제안한다. 제안하는 방법은 기존 CADO-NFS의 다항식과 같은 다항식을 선택하지만, 다항식 선택에 걸리는 시간은 약 40% 감소한다.

ABSTRACT

Currently, General Number Field Sieve(GNFS) is known as the most efficient way for factoring large numbers. CADO-NFS is an open software based on GNFS, that was used to factor RSA-704. Polynomial selection in CADO-NFS can be divided into two stages - polynomial selection, and optimization of selected polynomial. However, optimization of selected polynomial in CADO-NFS is an immense procedure which takes 90% of time in total polynomial selection. In this paper, we introduce modification of optimization stage in CADO-NFS. We implemented precomputation table and modified optimization algorithm to reduce redundant calculation for faster optimization. As a result, we select same polynomial as CADO-NFS, with approximately 40% decrease in time.

Keywords: GNFS, Polynomial Selection, Root optimization

I. Introduction

RSA cryptosystem is one of the most widely used public key cryptosystem for providing privacy and ensuring authenticity of digital data. The security of RSA cryptosystem is based on hardness of factoring large numbers. In RSA cryptosystem, public modulus N is chosen as product of two distinct primes p, q of same size[1]. Let e, d be two integers satisfying $ed = 1 \bmod \phi(N)$ where $\phi(N) = (p-1)(q-1)$. The pair (N, e) is called public key and d is called private

Received(05. 02. 2016), Modified(06. 01. 2016), Accepted(06. 02. 2016)

* 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대 정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구입니다(No. NRF-2014M3C4A7030649)

† 주저자, suhrikim@gmail.com
‡ 교신저자, chkim@semyung.ac.kr(Corresponding author)

key. Note that by factoring $N=pq$ to obtain p, q, it is easy to find private key. Hence integer factorization is one of main topics for research.

General Number Field Sieve (GNFS) is currently best known algorithm for factoring large numbers over 110 digits[9]. Factoring based on GNFS is recently performed by Kleinjung et al. in 2009 for RSA-768[6], and also by Bai et al. in 2012 for RSA-704[5].

Implementation of GNFS includes GGNFS, Msieve, and CADO-NFS. GGNFS is best optimized for factoring up to 160 digit integers and Msieve is best optimized for factoring up to 130 digit integers. Meanwhile CADO-NFS was used to factor RSA-704 (212 digit) so that it is reasonable to consider CADO-NFS for factoring integers over 300 digits[9].

CADO-NFS selects polynomial using Kleinjung's second algorithm and optimizes selected polynomial in two perspective for better performance in sieving stage[2]. Since optimization of selected polynomial takes roughly 90% of total polynomial selection stage, modification is needed for faster polynomial selection. We conclude that redundant calculation of values delays optimization time even more. Hence we implemented precomputation table and modified root optimization procedure for faster selection. In this way, we can generate polynomial with same Murphy E value as CADO-NFS in shorter time.

This paper is organized as follows : In Section 2, we cover the concept of GNFS and describe properties that good polynomial should have. In Section 3, we briefly describe optimization of polynomials and focus on its implementation in CADO-NFS. In section 4, we present our modification of root optimization process. Following experimental result for our modification is presented in Section 5. Finally, we conclude our result in Section 6.

II. Background

In this Section, we briefly describe modern factoring algorithm, GNFS. Next, we introduce classical method for polynomial selection in GNFS. Lastly, two measurement that quantifies polynomial's qualities are presented.

2.1 General Number Field Sieve

Most of modern factoring algorithms are based on 'Difference of squares.' For $n=pq$ where p, q are two distinct primes, 'Difference of squares' focuses on finding two random integers x, y such that $x \neq y \bmod n$ and $x^2 \equiv y^2 \bmod n$. Then by computing $\gcd(x-y, n)$ and $\gcd(x+y, n)$, we can obtain non-trivial factor of n with high probability. This idea is extended to Quadratic Sieve(QS) and later developed into GNFS.

Instead of searching for random integers as in 'Difference of squares,' GNFS searches irreducible monic polynomial $f(x)$ of degree greater than 1 and monic linear polynomial $g(x)$ such that $f(x)$ and $g(x)$ have common root modulo n[7,8].

Let α and β be roots of $f(x), g(x)$ not in Z_n, respectively, and consider rings $Z[\alpha]$, $Z[\beta]$. Goal in GNFS is to find (a,b) pairs such that $a-b\alpha$ and $a-b\beta$ are smooth over chosen basis of primes. We say that an element is smooth if all of its factors are member of our chosen basis of primes.

We collect (a,b) pairs where $\prod(a-b\,\alpha)=X^2$ for $X\in Z[\alpha]$ and $\prod(a-b\beta)=Y^2$ for $Y\in Z[\beta]$. Consider homomorphisms from ring $Z[\alpha]$ and $Z[\beta]$ to Z_n that maps α and β to m. Then there exist $x, y \in Z_n$ such that X, Y are mapped to x, y respectively. Hence $x^2 \equiv y^2 \bmod n$ is again obtained and non-trivial factors of n can be found with high probability.

Generally, GNFS is divided into four stages - polynomial selection, sieving, linear algebra, square roots - but we focus on polynomial selection stage of GNFS. Namely, polynomial selection where we select $f(x)$ and $g(x)$. This is because that sieving takes over 90%, 80% of total time for factoring 512 bit, 768 bit numbers, respectively, and choice of polynomial dramatically affect time to complete sieving. In next Section, we briefly describe method for polynomial selection.

2.2 Classical Polynomial Selection

Classic way to generate polynomial is using base-m method[11]. The base-m method expresses number n to be factored as $n = m^d + c_{d-1}m^{d-1} + \dots + c_0$, where $|c_i| \le \frac{m}{2}$ for each i, and generates two monic polynomials $f(x)$ of degree d and $g(x) = x - m$ of degree 1 with $m \bmod n$ as common root between $f(x)$ and $g(x)$. To reduce size of coefficient of $f(x)$, this method is modified to select non-monic polynomial $f(x) = \sum_{i=0}^{d} c_i x^i$ and $g(x) = m_2 x - m_1$ such that $n = \sum_{i=0}^{d} c_{d-i} m_1^{d-i} m_2^i$[3]. We choose m to be close to $(n/c_d)^{1/d}$[4]. If c_{d-2} is not small enough, try another c_d. Otherwise we optimize the generated polynomial pair.

2.3 Quantifying Quality of Polynomials

In Section 2.1, we highlighted the importance of selecting good polynomial. Since selection of polynomial greatly affect the number of relations to be found, we want to select good polynomial in order to expect good performance in sieving. Hence quantifying quality of found polynomials is needed. In this subsection, we present three measurements of quality, namely, lognorm, α-value, and Murphy E. Lognorm and α-value relates to size and root property respectively, and Murphy E score is combination of size and root properties.

Recall that aim of sieving stage is to collect many relations (a, b) such that $a - b\alpha$ is smooth over chosen basis of primes where α is root of $f(x)$. Generally, chosen basis consist of small primes and hence small value of norm of $f(x)$ is more likely to be factored by such basis. Thus, we calculate 'lognorm' of polynomial. Lognorm is logarithmic average of polynomial values across sieving region, and it is computed as below[4].

$$\frac{1}{2}\log\left(s^{-d}\int_0^{2\pi}\int_0^1 F^2(s\cos\theta, \sin\theta)\, r^{2d+1}\, dr\, d\theta\right).$$

In above equation, s refers to skewness of sieving region, calculated by ratio of a, b [4]. Hence small lognorm means size of polynomial is small so that it is more likely to be smooth over our chosen basis of primes. Thus we are searching for polynomial with smaller lognorm as possible. Since $g(x)$ is linear, we may assume size of $g(x)$ does not vary much across sieving region than $f(x)$. Therefore, in practice we only focus on lognorm of $f(x)$.

If a polynomial $f(x)$ has many roots modulo small prime powers, then we can expect that polynomial values to behave more smoothly than random integers about the same size. We define expected p-valuation of x as $\nu_p(x)$, where value of $\nu_p(x)$ is exponent of the largest power of prime p dividing x in set of integers S, and $\nu_p(0)=\infty$. We use same notation for polynomials to let $\nu_p(f)$ refer to expected p-valuation of f in set S.

Murphy defined $\alpha(F)$ function to compare cumulated expected p-valuation of polynomial values to random integers of similar size[4]. Hence, $\alpha(F)$ can be considered as logarithmic benefit of using polynomial values compared to using random integers. We call $\alpha(F)$ as α-value of polynomial and is defined as below.

$$\alpha(F)=\sum_{p\le B}\left(\frac{1}{p-1}-\frac{n_p p}{p^2-1}\right)\log p$$
$$=\sum_{p\le B}\left(1-\frac{n_p P}{p+1}\right)\frac{\log p}{p-1}.$$

In above equation, n_p refers to number of simple or multiple root of $f(x)$ modulo p^e for $p^e\le B$. Combining $f(x)$ and $g(x)$ we can approximate number of sieving reports as equation below[4,11].

$$\frac{6}{\pi^2}\iint_\Omega \rho\left(\frac{\log|F(x,y)|+\alpha(F)}{\log B_1}\right)\rho\left(\frac{\log|G(x,y)|+\alpha(G)}{\log B_2}\right)dxdy.$$

Above measurement is called 'Murphy E' of polynomials. Since collecting as many relation as possible is goal in sieving stage, larger Murphy E implies that it is more likely to have large number of sieving reports in sieving stage. Hence we focus on selecting $f(x), g(x)$ with larger Murphy E value.

III. Polynomial Optimization

Through polynomial generation described in Section 2.2, we can obtain polynomial whose size of first three leading coefficients are small. We namely call this output $f(x)$ of polynomial generation stage as 'raw polynomial.' However to have better performance in sieving stage, optimization of raw polynomial is necessary.

In this section, two optimization stages for better lognorm and Murphy E, are described. In Section 3.1, size optimization for smaller lognorm is described. In Section 3.2, root optimization for larger Murphy E is described. Polynomial optimization takes 90% of total time of polynomial generation stage, and root optimization for better Murphy E values takes 90% of total time in polynomial optimization.

3.1 Size Optimization

The goal of size optimization is to obtain polynomial $f(x)$ with smaller lognorm. Note that for $f(x)=\sum_{i=0}^{d}c_i x^i$, size of first three leading coefficient c_d, c_{d-1}, and c_{d-2} are controlled during polynomial generation. However other coefficients are left uncontrolled. Thus size optimization focuses on controlling c_{d-3} through translation and other coefficients through rotation.

We call computing $f(x+k)$ and $g(x+k)$ as translation. To reduce size of c_{d-3} effectively, right choice of k is needed. Let $f(x+k)=\sum_{i=0}^{d}c_i(x+k)^i=\sum_{i=0}^{d}a_i x^i$. We want a_{d-3} to be small after translation. Note that $a_{d-3} = \binom{d}{3} c_d k^3 + \binom{d}{2} c_{d-1} k^2 + d c_{d-2} k + c_{d-3}$ and consider a_{d-3} as function of k. Then we may write a_{d-3} as $h(k) = a_{d-3} = \binom{d}{3} c_d k^3 + \binom{d}{2} c_{d-1} k^2 + d c_{d-2} k + c_{d-3}$. Let a be nearest integer of k where $h(k) = 0$. Then translating $f(x)$ by a, we can expect to have small a_{d-3}. Since degree of $h(k)$ is 3, there are at most 3 values of a for translation. We calculate $f(x+a)$ for each a and check its lognorm. If lognorm of $f(x+a)$ is smaller than $f(x)$, we replace $f(x)$ by $f(x+a)$. If not, $f(x)$ is left unchanged and we continue to next optimization stage.

After translation, we rotate $f(x)$ and obtain $f(x) + (wx^2 + ux + v) g(x)$ to reduce size of last three coefficients. As stated above, we use quadratic rotation in order to preserve the size properties of translated polynomial. Hence after each rotation, lognorm is calculated to check polynomial's size property. If lognorm of rotated polynomial is larger than original, $f(x)$ remain unchanged and move to root optimization stage.

As a summary, total procedure of size optimization is described in Algorithm 1[4]. Note that $L^2(f)$ indicates lognorm of $f(x)$.

Algorithm 1

```
Input : Polynomial pair $f(x) = \sum_{i=0}^{d} c_i x^i$,
        $g(x) = m_2 x - m_1$
        Integers $U, V, B$
Output : polynomial pair $f(x), g(x)$ of smaller lognorm.

1. $k = w = u = v = 1$;
2. while minimum is found or loop limit is reached do
  2.1 $f'(x) = f(x \pm k)$, $g'(x) = g(x) \pm k m_2$
  2,2 if either $L^2(f') < L^2(f)$ then
    2.2.1 $f = f', g = g', k = 2k$
  2.3 else
    2.3.1 $k = \lceil k/2 \rceil$
  2.4 $f'(x) = f(x) \pm w x^2 g(x)$
  2.5 if either $L^2(f') < L^2(f)$ then
    2.5.1 $f = f', w = 2w$
  2.6 else
    2.6.1 $w = \lceil w/2 \rceil$
3. return $f(x), g(x)$
```

3.2 Root Optimization

In order to have good root property in terms of α-values, root optimization is performed after adjusting polynomial size through size optimization. Good root property roughly requires polynomial to have many roots modulo small prime and prime powers. Total outline of root optimization procedure is as follows. We rotate $f(x)$ and check root property of $f(x) + h(x)g(x)$. That is, we record roots modulo small prime and prime powers of $f(x) + h(x)g(x)$. After rotation has finished for all possible $h(x)$ we replace $f(x)$ to one of $f(x) + h(x)g(x)$ having highest Murphy E values among all others. We consider quadratic rotation in order to preserve size property obtained by size optimization. Thus $h(x)$ can be written as $h(x) = wx^2 + ux + v$, and we search for (w, u, v) such that rotated polynomial $f(x) + (wx^2 + ux + v)g(x)$ has good root properties. Since linear rotation is faster than quadratic rotation, we first search for w, sets $f(x) = f(x) + wx^2 g(x)$ and find (u, v) such that $f_{u,v} = f(x) + (ux + v)g(x)$ has good root property by linear rotation. Intuitive approach for root optimization is to check α-value of $f_{u,v}$ for all possible (u,v). However, since bound of u,v are huge, this is time consuming process. Therefore, Murphy uses sieve-like procedure to find polynomial with good root properties[11]. Idea is that when r is root of $f_{u,v}(x) \equiv 0 \bmod p^e$, $f_{u+ip^e,\, v+jp^e}(x) \equiv f_{u,v}(x) \bmod p^e$ so that r is also root of $f_{u+ip^e,\, v+jp^e}(x) \equiv 0 \bmod p^e$ for $i,j \in Z$.

Let B be bound for small primes and U, V be bound for u and v respectively. Algorithm 2 describes Murphy's root sieve which is used as base method for CADO-NFS's root optimization[4].

```
Algorithm 2
Input : Polynomial pair $f(x), g(x)$,
        Integers $U, V, B$
Output : array of approximated $\alpha$-values
of dimension $U \times V$

1. For $p \le B$ do
  1.1 For $e$ where $p^e \le B$ do
    1.1.1 For $x \in [0, p^e - 1]$ do
      1.1.1.1 For $u \in [0, p^e - 1]$ do
        1.1.1.1.1
      compute $v$ in $f_{u,v}(x) \equiv 0 \bmod p^e$
        1.1.1.1.2
      update $\nu_p(f_{u+ip^e, v+jp^e})$ by sieving
```

3.3 Application in CADO-NFS

In Section 3.2, Algorithm 2 works for small bounds U, V. However, for polynomial of degree greater than 6, permissible U, V bounds are large. Hence CADO-NFS uses modification of Algorithm 2 for faster root sieve[4]. The core idea is that if r is simple root of $f(x) \equiv 0 \bmod p$, then p-valuation can be easily estimated so that there is no need to count the lifted roots. We determine simplicity of roots by using Hensel's lemma. We call root r is simple root modulo p if $f'(r) \neq 0 \bmod p$ and multiple root modulo p if $f'(r) \equiv 0 \bmod p$.

Recall that if r_e is a simple root of rotated polynomial $f_{u,v}(x) \bmod p^e$ for $e \ge 1$ then r_e is also simple root for $f_{u+ip^e, v+jp^e} \bmod p^e$. Since the contribution of the root r_e to $\nu_p(F_{u,v})$ is $p/(p^2-1)$, we can update the score for all rotated polynomials of the form $f_{u+ip, v+jp}(x)$ in sieve.

If r_e is a multiple root of $f(x) \bmod p^e$ and $f(r_e) \equiv 0 \bmod p^{e+1}$ then r_e can lifted to p number of roots modulo p^{e+1} such that $r_{e+1} = r_e + lp^e$ for all integer $l \in [0, p)$. Additionally, the lifted roots r_{e+1} are also multiple since $f'(r_{e+1}) \equiv 0 \bmod p$. Whereas if $f(r_e) \neq 0 \bmod p^{e+1}$, r_e cannot be lifted to a root modulo p^{e+1}. Note that in order for r to be multiple root modulo p for some rotated polynomial $f_{u,v}(x) \bmod p$, it should satisfy equation $f(r) + (ur+v)g(r) \equiv 0 \bmod p$ and $f'(r) + ug(r) + (ur+v)g'(r) \equiv 0 \bmod p$. Since $(ur+v) \equiv -f(r)/g(r) \bmod p$, we can obtain following equation.

$$ug^2(r) \equiv f(r)g'(r) - f'(r)g(r) \bmod p. \qquad (1)$$

Hence if r is root of $f(x)$ modulo p and if

$$ug^2(r) - f(r)g'(r) + f'(r)g(r) \qquad (2)$$

is divisible by p, then r is multiple root of $f(x)$ modulo p. If r is multiple root of $f_{u,v}(x) \bmod p$, lifting is needed for counting the lifted roots.

CADO-NFS divides root optimization into two stages. In the first stage, it searches for pair $(u_0, v_0) \bmod (p_1^{e_1} \cdots p_m^{e_m})$ such that $f_{u_0,v_0}(x)$ has many roots modulo prime powers for first m smallest primes. In second stage, it applies root sieve to $f_{u_0,v_0(x)}$ for larger prime powers up to some bound B.

Naive way for selecting (u_0,v_0) for given $f(x)$ in first stage is to generate matrix of pairs of (u,v) with size $\left(\prod_{i=1}^{m} p_i^{e_i}\right)^2$ and chose one (u,v) that best generates polynomial with good root property. This approach is possible only if size of matrix is small. In CADO-NFS, m individual polynomials $f_{u_i,v_i,p_i}(x)$ $(1 \le i \le m)$ is founded for each prime p_i that has many roots modulo small $p_i^{e_i}$. Then by Chinese Remainder Theorem(CRT) we calculate $(u_0, v_0) \bmod (p_1^{e_1} \cdots p_m^{e_m})$ to obtain combined polynomial $f_{u_0,v_0}(x)$. Note that $f_{u_0,v_0}(x) \bmod p_i^{e_i}$ has same number of roots as $f_{u_i,v_i,p_i}(x) \bmod p_i^{e_i}$ for $1 \le i \le m$. Since we selected (u_i,v_i) such that $f_{u_i,v_i}(x) \bmod p_i^{e_i}$ to have many roots, combined polynomial $f_{u_0,v_0}(x)$ is also likely to have many roots modulo small prime powers of $p_1^{e_1}, \ldots, p_m^{e_m}$.

Second stage of root sieve is analogous to first stage except for the fact that it uses larger primes. Let $M = \prod_{i=1}^{m} p_i^{e_i}$ and (u_0, v_0) be fixed as in first stage. Recall that we used sub-lattice defined by (u,v) in first stage. In second stage, we do root sieve for larger prime powers on sub-lattice defined by $(u_0 + \gamma M, v_0 + \beta M)$.

As a summary, modification of Algorithm 2 for faster root sieve is described in Algorithm 3[4].

```
Algorithm 3
Input : Polynomial pair $f(x), g(x)$,
        Integers $U, V, B$
Output : an array of murphy E of dimension $U \times V$

1. For $p \le B$ do
  1.1. For $x \in [0, p-1]$ do
    1.1.1 compute $\tilde{u}$ such that
      $\tilde{u} g^2(x) \equiv f(x)g'(x) - f'(x)g(x) \bmod p$
  1.2. For $u \in [0, p-1]$ do
    1.2.1. compute $v$ such that
        $f(x) + uxg(x) + vg(x) \equiv 0 \bmod p$
    1.2.2. if $u \ne \tilde{u}$
      1.2.2.1 then update $\nu_p(f_{u+ip, v+jp})$
            in sieving
    1.2.3 else
      1.2.3. lift to count multiple roots
          of $f_{\tilde{u},\tilde{v}}(x) \bmod p^e$ such that
          $(\tilde{u}, \tilde{v}) \equiv (u, v) \bmod p$,   $\tilde{u}, \tilde{v} \le p^e$,
          $p^e \le B$ and then sieve;
```

IV. Proposed Method

In this Section, we proposed our modification for faster root sieve. We implemented precomputation table to reduce the amount of redundant calculation. We also modified searching procedure at beginning of second stage of root sieve and reduced number of sub-lattice for faster root sieve.

4.1 Table Implementation

Since root optimization in CADO-NFS has redundant calculation of values, one way to optimize CADO-NFS's root optimization procedure is to reduce the number of such calculation. Note that for each prime p, and for each $0 \le u < p$ and for each $0 \le r < p$, equation (2) is calculated in order to determine whether root is simple or not. Note that when $p=2$, equation (2) is calculated for $(u,r)=(1,1)$. But when $p=3$, equation (2) is again calculated for $(u,r)=(1,1)$. Hence $f(r)$ for $0 \le r < p$ is calculated repeatedly for some value.

Thus instead of calculating equation (2) for each u and r, we precompute

$$\{ug(r)^2 - f(r)g'(r) + f'(r)'g(r)\} \bmod p \qquad (3)$$

and record this value along with u, r, p. In this way, we can look up for value in table when needed instead of calculating every time. Note that since prime p is smaller than 199 in practice, we can make size of each entry of precomputation table to be 8 bits.

When generating precomputation table, order of computation for calculating $ug(r)^2$ is modified. In CADO-NFS, it fixes value u and calculates $ug(r)^2$ for $0 \le r < p$. For example, when $u=2$, it calculates $g(0)^2 \times 2$, $g(1)^2 \times 2$, and so on. Instead, we fix value r and calculate $ug(r)^2$. For example, when $r=0$, we calculate $g(0)^2 \times 2$, $g(0)^2 \times 3$, and so on. By modifying the order of calculation, multiplication is not needed – when calculating $g(0)^2 \times u$ we add $g(0)^2$ instead. Hence if we precalculate $g(r)^2$ for $0 \le r < 199$, we can generate $ug(r)^2$ only by addition. Since addition is faster than multiplication, we can expect shorter computation time.

Comparison of computation between CADO-NFS and table implementation is shown as Table 1.

Number of computation is examined based on RSA-768 number with leading coefficient of $f(x)$ as 265482057982680[6]. Total number of calculation refers to the

Table 1. Comparison of calculation of $ug(r)^2 - f(r)g'(r) + f'(r)'g(r)$

	CADO-NFS	Table
Total number of calculation	110,299,200	11,940
Total number of division	110,299,200	11,940

number of multiplication needed for calculating equation (2) during root optimization period. Since $f(r)g'(r)$ and $f'(r)g(r)$ are precalculated, we only need to consider 2 multiplications when computing equation (2). Specifically, we consider 2 multiplications needed for $ug(r)^2$ – one for computing $g(r) \times g(r)$ and one for computing $u \times g(r)^2$. Total number of division refers to the number of division for checking existence of $g(r)^{-1} \bmod p$ and computation of equation (3) that checks whether computed root is simple root or multiple root.

In summary, total 2 multiplications and 2 divisions occurs for one round of root sieve in root optimization. Since there exist average 10,000 rounds of root sieve per one pair (p_1, p_2) of primes, this table implementation can lead to significant decrease in number of computation.

4.2 Modification of Root Sieve Process

In first stage of root sieve in CADO-NFS, it searches for top 3 w values and records (w,u,v) along with α-values in array such that

$$f(x) + (wx^2 + ux + v)g(x)$$

has good root property for first 10 smallest primes. In beginning of second stage of root sieve, it extracts (w,u,v) pairs recorded in array and rotate $f(x)$ to generate $f(x)+(wx^2+ux+v)g(x)$. Next it tests root property of rotated polynomial for first 45 primes and reorder (w,u,v) by α-value. After this step, it focuses on searching good (u,v) for larger prime that has good root property.

However, at beginning of second stage, testing root properties with larger primes does not change any (w,u,v) of recorded pairs. In other words, elements in input array is same as output array except for recorded order. Fig. 1 and Fig. 2 depicts the recorded (u,v) pairs in end of first stage and after first step of second stage, respectively.

In Fig. 1 and Fig. 2, x-axis indicates value of u and y-axis indicates value of v. As shown in Fig. 1 and Fig. 2, no extra (u,v) was recorded to array nor extra (u,v) was deleted in array. This result is natural since expected p-valuation

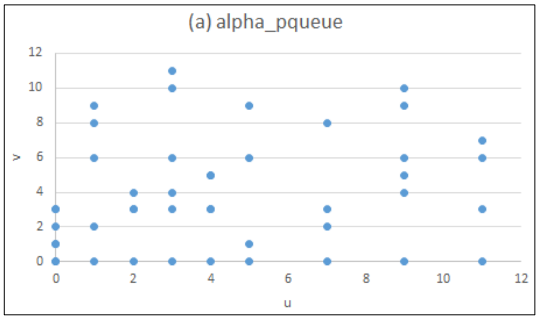

Fig. 1. Recorded (u,v) pair after first stage

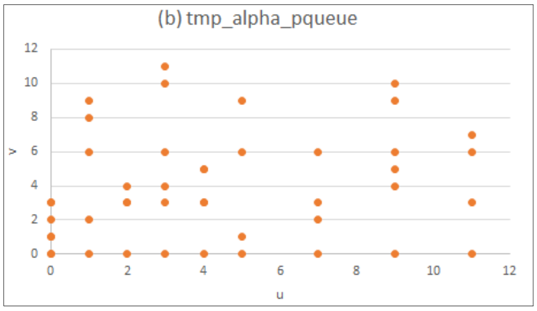

Fig. 2. Recorded (u,v) pair after first step of second stage

decreases as prime increases. Hence effect on α-value of polynomial due to inclusion of more primes can be negligible. Therefore, it is possible to delete beginning of second stage and immediately move to step where deep search on (u,v) for larger prime is performed.

As a summary, total procedure of modification of root optimization algorithm is described in Algorithm 4.

```
Algorithm 4
Input : Polynomial pair $f(x), g(x)$,
        Integers $U, V, B$
        Array of dimension :
Output : an array of murphy E of
dimension $U \times V$

1. Precomputation
  1.1 For $p \le B$ do
    1.1.1For $x \in [0, p-1]$ do
      1.1.1.1 For $u \in [0, p-1]$ do
        1.1.1.1.1 Compute
  $H = ug^2(x) - f(x)g'(x) + f'(x)g(x) \bmod p$
    1.1.1.1.1 Save $H$ in array $A[x][u][p]$

2. For $p \le B$ do
  2.1. For $x \in [0, p-1]$ do
    2.1.2. For $u \in [0, p-1]$ do
      2.1.2.1. compute $v$ such that
        $f(x) + uxg(x) + vg(x) \equiv 0 \bmod p$
      2.1.2.2. if $A[x][u][p] = 0$
        2.1.2.2.1 update $\nu_p(f_{u+ip, v+jp})$
          in sieving
      2.1.2.3 else
        2.1.2.3.1 lift to count multiple roots of
              $f_{\tilde{u},\tilde{v}}(x) \bmod p^e$ such that
          $(\tilde{u},\tilde{v}) \equiv (u,v) \bmod p$, $\tilde{u},\tilde{v} \le p^e$,
        $p^e \le B$ and then sieve;
```

V. Experimental Result

Experiments were performed using gcc version 4.9.2 with Intel Core i5-4690K processor at 3.5GHz with 8GB RAM. We used CADO-NFS 2.1.1 version for size optimization.

Table 2 is the experimental result for average time took for 6 number of root optimizations. Experiment was done with RSA-768 number with 265482057982680 as leading coefficient of degree 6 polynomial $f(x)$[6]. We used $P=1000000$ so that $p_1, p_2 \in [P, 2P]$, where p_1, p_2 are prime factor of leading coefficient of $g(x)$.

In Table 2, total time for polynomial selection refers to time it takes to search polynomial for one leading coefficient of $f(x)$. This includes polynomial generation along with size and root optimization. With our selection of input parameters, CADO-NFS found 6 collision pairs (p_1, p_2) so that 6 number of root optimization occurred. Hence average time in Table 2 refers to average time for one polynomial selection. As stated in Table 1, due to our modification, we select same polynomial as in CADO-NFS with 41.5% decrease in total time for polynomial selection.

Table 3 denotes the experiment results for average time took for 24 number of root optimizations. Experiment was done with RSA-768 number with $f(x)$ of degree 6. In Table 3, ad range refers to the range of leading coefficient of $f(x)$. We used $P=10000$.

As in Table 3, our modified version selects same polynomial as in CADO-NFS. Furthermore, it optimizes polynomial in 55.94% decrease in time for root optimization. This brought 55.85% decrease in total time for polynomial selection.

Table 2. Comparison of Average Time on Root Optimization.

	CADO-NFS	Modified
Total time for polynomial selection (s)	1185.87	693.44
Average time for polynomial selecletion (s)	193.14	115.55
Best Murphy E	8.84×10^{-18}	8.84×10^{-18}

Table 3. Comparison of Average Time on Root Optimization.

	CADO-NFS	Modified
Ad range	265482057982680~ 265482057988680	
Number of optimization	24	24
Total time for polynomial selection (s)	20926.44	9238.95
Average time for root optimization(s)	870.39	383.45
Best Murphy E	4.58×10^{-23}	4.58×10^{-23}

VI. Conclusion

In this paper, we proposed table implementation of root optimization in CADO-NFS. When generating precomputation table, we modified the order of computation for faster calculation. As a result, we calculated values using addition whereas CADO-NFS used multiplication to obtain same value.

Additionally, we reduce size of sub-lattice for faster root sieve. By experiment, reducing number of sub-lattice does not effect the quality of output polynomial. In this way, we can generate polynomial with Murphy E value same as CADO-NFS but in shorter time.

Note that since Murphy E value greatly relates to number of expected relation in sieving stage, simply reducing time in polynomial selection without considering output polynomial's Murphy E value is useless. However, as in Section 5, our modified version selects same polynomial as in CADO-NFS but in shorter time. Thus we expect to search more polynomials than CADO-NFS in same amount of time and hence have higher probability of selecting better polynomial.

References

〔1〕 R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signature and Public-Key Cryptosystems," ACM, vol.21(2), pp.120-126, 1978.

〔2〕 T. Kleinjung. "Polynomial selection," In CADO workshop on integer factorization, INRIA Nancy, http://cado.gforge.inria.fr/workshop/slides/kleinjung.pdf. 2008.

〔3〕 T. Kleinjung. "On polynomial selection for the general number field sieve," Mathematics of Computation, pp. 2037–2047, 2006.

〔4〕 S. Bai "Polynomial Selection for the Number Field Sieve," Ph.D. Thesis ,The Australian National University, 2011.

〔5〕 S. Bai, E. Thom´e, P. Zimmermann. Factorisation of RSA-704 with CADO-NFS. Report, http://eprint.iacr.org/2012/369.pdf.

〔6〕 T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thom´e, J. W. Bos, P. Gaudry, A. Kruppa,P. L. Montgomery, D. A. Osvik, H. J. J. te Riele, A. Timofeev, and P. Zimmermann. "Factorization of a 768-bit RSA modulus," CRYPTO '10, vol.6223 LNCS, pp 333–350, 2010.

〔7〕 A. K. Lenstra and H. W. Lenstra, Jr., editors. "The Development of the Number Field Sieve," vol. 1554 of Lecture Notes in Mathematics. Springer, 1993.

〔8〕 Matthew E. Briggs "An Introduction to the General Number Field Sieve," Master Thesis. Virginia Polytechnic Institute and State University. April, 1998.

〔9〕 P. Gaudry, A. Kruppa, et al. "CADO-NFS," http://cado-nfs.gforge.inria.fr

〔10〕 B. A. Murphy, R. P. Brent, "On Quadratic Polynomials for the Number Field Sieve," CATS'98, pp 199-231, 1998.

〔11〕 B. A. Murphy, "Polynomial Selection for the Number Field Sieve Integer Factorization Algorithm," Ph.D. Thesis, The Australian National University, 1999.

〈저자소개〉

김 수 리 (Suhri Kim) 학생회원
2014년 2월: 고려대학교 수학과 학사
2014년 8월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 부채널 공격, 공개키 암호시스템

권 희 택 (Heetaek Kwon) 학생회원
2010년 2월: 고려대학교 수학과 학사
2010년 3월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 정보보호, 공개키 암호시스템

이 용 성 (Yongseong Lee) 학생회원
2015년 3월: 고려대학교 수학과 학사
2015년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 정보보호, 대칭키, 공개키 암호시스템

장 남 수 (Nam Su Chang) 정회원
2002년 2월: 서울 시립대학교 수학과 이학사
2004년 8월: 고려대학교 정보보호 대학원 공학석사
2010년 2월: 고려대학교 정보경영공학전문대학원 공학박사
2010년 7월~현재: 세종사이버대학교 정보보호학과 조교수
〈관심분야〉 암호칩 설계 기술, 부채널 공격, 공개키 암호 알고리즘, 공개키 암호 암호분석

윤 기 순 (Kisoon Yoon) 정회원
1998년 8월: 경희대학교 수학과 이학사
2007년 8월: 고려대학교 정보보호학과 공학석사
2013년 11월: Université de Caen 수학과 이학박사
2013년 11월~현재: 엔에스에이치씨 암호기술팀 팀장
〈관심분야〉 정수론, 암호학, 정보보호

김 창 한 (Chang Han Kim) 종신회원
1985년 2월: 고려대학교 수학과 이학사
1987년 2월: 고려대학교 수학과 이학석사
1992년 2월: 고려대학교 수학과 이학박사
1992년 8월~현재: 세명대학교 정보통신학부 교수
〈관심분야〉 정수론, 공개키암호, 암호프로토콜

박 영 호 (Young-Ho Park) 종신회원
1990년 2월: 고려대학교 수학과 이학사
1993년 2월: 고려대학교 수학과 이학석사
1997년 2월: 고려대학교 수학과 이학박사
2002년 3월~현재: 세종사이버대학교 정보보호학과 교수
〈관심분야〉 공개키 암호, 암호 프로토콜, 부채널 공격, 암호안전성평가

홍 석 희 (Seokhie Hong) 종신회원
1995년 2월: 고려대학교 수학과 학사
1997년 2월: 고려대학교 수학과 석사
2001년 8월: 고려대학교 수학과 박사
1999년 8월~2004년 2월: (주) 시큐리티 테크놀로지스 선임연구원
2003년 8월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U.Leuven, ESAT/SCD-COSIC 박사후연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키·공개키 암호 분석 및 설계, 컴퓨터 포렌식