國立交通大學

資訊科學與工程研究所

碩 士 論 文

$n = p \times q$ 的因數分解之研究

Study on Factorization of $n = p \times q$

研 究 生：張煜晧

指導教授：葉義雄 教授

中 華 民 國 九 十 五 年 九 月

$n = p \times q$ 的因數分解之研究
Study on Factorization of $n = p \times q$

研 究 生：張煜皓　　Student：Yu-Hao Chang

指導教授：葉義雄　　Advisor：Yi-Shiung Yeh

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

September 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年九月

$n = p \times q$ 的因數分解之研究

學生：張煜皓 指導教授：葉義雄 博士

國立交通大學資訊工程學系碩士班

摘 要

RSA 密碼系統(RSA Cryptosystem)是使用最為廣泛的公鑰密碼系統之一，其安全性乃建立在大整數難以分解為其質因數乘積的事實之上，此一事實並被稱為 RSA 假定(RSA assumption)。一般相信沒有確定型的圖靈機(deterministic Turing machine，簡稱 DTM)可在多項式時間內破解 RSA 假定，多項式時間的演算法若被發現，RSA 密碼系統將變得不再安全。因為如此，許多科學家致力於研究有效率的分解演算法。目前所知，分解小於 110 位數的大數時，「二次篩選法(quadratic sieve factoring algorithm，簡稱 QS)」是最快的通用演算法。受限於時間與硬體資源，我們主要著眼於 QS 的一種變型，稱之為「複數多項式二次篩選法(multiple polynomial quadratic sieve，簡稱 MPQS)」。為了確認 RSA 假定的強度，我們提出一個方法來加速 MPQS 的篩選程序，其實驗結果將有助於分析 RSA 抵抗現行分解技術的強度，同時可被納入實作 RSA 密碼系統時的考量。

關鍵字：RSA 密碼系統、因數分解、二次篩選、複數多項式二次篩選法

Study on Factorization of $n = p \times q$

student：Yu-Hao Chang

Advisor：Dr. Yi-Shiung Yeh

Institute of Computer Science and Information Engineering
National Chiao Tung University

ABSTRACT

The RSA Cryptosystem is one of the most used public-key cryptosystems. The security it rests on the fact that it is computationally infeasible to factor a large integer into its component primes. This fact is referred to as the RSA assumption. It is believed that there is no deterministic Turing machines (DTM) that can break the RSA assumption in polynomial time. If a polynomial-time algorithm is found, the RSA Cryptosystem would be insecure. Owing to this, many scientists have devoted themselves to researching efficient factoring algorithms. So far, the quadratic sieve factoring algorithm (abbreviated to QS) is the fastest known general-purpose method for factoring numbers having less than about 110 digits. Restricted by time and computer hardware, we focus on one of the variants of the QS, called the multiple polynomial quadratic sieve (MPQS). To ensure the strength of the RSA assumption, we propose a scheme to enhance the sieving procedure of the MPQS. The experimental results are contributive to the analyses of the strength of the RSA assumption against the modern factoring technology and should be taken into consideration on future cryptographic implementations based on the RSA cryptosystem.

Keywords：RSA Cryptosystem, factoring integers, quadratic sieve, multiple polynomial quadratic sieve

誌　　謝

能夠完成這篇論文，首先必須感謝我的指導教授 葉義雄老師。這兩年來，承蒙老師悉心指導，使我在研究和待人處世方面獲益匪淺；此外，老師一直容忍我各種任性的要求，如今學業有成，我衷心感謝他。再者，要感謝 定宇學長，在我論文寫作的期間，不時給我鼓勵，並與我討論研究方向；這篇論文得以順利付梓，定宇學長居功厥偉。還有特別感謝中山大學的 官大智教授，願意提供他珍貴的程式，供我們研究。也感謝 韓禹學長，幫忙整理我的論文，不吝指導我寫作的技巧。感謝實驗室的同窗好友，宗哥、鴻祥、阿甘、昇哥，研究所的兩年有幸與你們同行，讓我習得了不少特殊技能，這將是我一生寶貴的資產。碩二的學弟妹，Qting、Gobby、伯昕，雖然我大你們一屆，卻總是受你們照顧，在此謝謝你們。還有新進的學弟妹，佳君、小強、家明，與你們相處的時日雖短，但非常開心，祝你們接下來的兩年一帆風順。也謝謝 以德學長、鎮宇學長、靜紋學姊，一直以來對我照顧有加。最後，要謝謝我的 爺爺、奶奶、爸爸、媽媽，多虧你們苦心栽培，今日我才能拿到這個學位，謝謝你們。僅以此文獻給所有支持和關心我的人，謝謝!

張煜晧

中華民國九十五年九月

Contents

List of Tables

Chapter 1 Introduction

The *RSA Cryptosystem* [1] is one of the most important public-key cryptosystems, and the security of it rests on the fact that it is computationally infeasible to factor a large integer into its component primes. If an efficient algorithm is found that can factor any large integer in polynomial time, the RSA Cryptosystem would be insecure.

In this chapter, we will describe some important number-theoretic results, the RSA Cryptosystem, the details of setting up it, etc.

1.1 Elementary Number Theory

In the beginning of this section, we first introduce some basic definitions from elementary group theory.

Definition 1: [1]

For a finite multiplicative group G, define the *order* of an element $g \in G$ to be the smallest positive integer m such that $g^m = 1$. If there are n elements of G, then we say that G is a multiplicative group of order n. □

We then proceeds to mention a very important theorem, called the *Lagrange's theorem* [1].

Theorem 1

Suppose G is a multiplicative group of order n, and $g \in G$. Then the order of g divides n. □

From Theorem 1, it is clear that $g^n = (g^m)^{n/m} = 1$ for any element $g \in G$.

For any positive integer n, let $\mathbb{Z}_n^*$ denote the set of residues modulo n that are relatively prime to n. It can be easily verified that $\mathbb{Z}_n^*$ is a (finite) multiplicative group. The *Euler phi-function* $\phi(n)$ [1] is defined to be the number of positive integers not exceeding n and relatively prime to n. That is, $\left|\mathbb{Z}_n^*\right| = \phi(n)$. Given the prime-power factorization of n, a well-known theorem provides a formula to evaluate the value of $\phi(n)$ [2]:

Theorem 2

Let $n = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$ be the prime-power factorization of the positive integer n. Then

$$\phi(n) = n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_k}\right). \qquad (1)$$

□

By using the results above, it is easy to see that

$$g^{\phi(n)} \equiv 1 \pmod{n} \qquad (2)$$

for any element $g \in \mathbb{Z}_n^*$. This fact is fairly important and essentially relevant to the RSA Cryptosystem.

1.2 The RSA Cryptosystem

The RSA Cryptosystem is one of the most important public-key cryptosystems, which is invented by *Ronald Rivest*, *Adi Shamir*, and *Leonard Adleman* in 1977. In this section, we will describe how it works. Let $n = p \times q$, where p and q are two large primes. By Theorem 2, it is clear that $\phi(n) = (p - 1)(q - 1)$. An integer d is chosen such that $\gcd(d, \phi(n)) = 1$. We next compute

$$e = d^{-1} \bmod \phi(n). \qquad (3)$$

(Since $\gcd(d, \phi(n)) = 1$, the inverse of d modulo $\phi(n)$ must exist.) Then, the private key is pair (d, n), and the public key is pair (e, n). To encrypt a message M (where M is a nonnegative integer less than n), the cipher C is computed as

$$C = M^e \bmod n. \qquad (4)$$

To decrypt the cipher C, we compute

$$M' = C^d \bmod n. \qquad (5)$$

We now verify that $M' = M$. Since $e = d^{-1} \bmod \phi(n)$, we have that

$$ed \equiv 1 \pmod{\phi(n)}.$$

$$\Rightarrow \quad ed = k\phi(n) + 1, \text{ for some } k \in N. \qquad (6)$$

We first consider the case that $M \in \mathbb{Z}_n^*$. Using the result from Section 1.1, it follows that

$$\begin{aligned} M' &= C^d \bmod n \\ &= (M^e)^d \bmod n \\ &= M^{ed} \bmod n \\ &= M^{k\phi(n)+1} \bmod n \\ &= \left(M^{\phi(n)}\right)^k M \bmod n \\ &= (1)^k M \bmod n \\ &= M. \end{aligned} \qquad (7)$$

For $M \notin \mathbb{Z}_n^*$, if $M = 0$, it is clear that $M' = M$. If $M \neq 0$, without loss of generality, suppose that $M = kp$ for some $k \in N$. Since $M < n$, it must be the case that $\gcd(k, q) = 1$, namely $\gcd(M, q) = 1$. Then it follows from the *Fermat's Little Theorem* [2] that

$$M^{(q-1)} \equiv 1 \pmod{q}.$$

$$\Rightarrow \quad M^{(q-1)} = k'q + 1, \text{ for some } k' \in N. \qquad (8)$$

Thus we have

$$\begin{aligned} M' &= C^d \bmod n \\ &= M^{k\phi(n)+1} \bmod n \\ &= M^{k\phi(n)} M \bmod n \\ &= (M^{(q-1)})^{k(p-1)} M \bmod n \\ &= (k'q+1)^{k(p-1)} M \bmod n \\ &= M \sum_{i=0}^{k(p-1)} C_i^{k(p-1)} (k'q)^i \bmod n \\ &= M + (kp)q \sum_{i=1}^{k(p-1)} C_i^{k(p-1)} (k')^i q^{i-1} \bmod n \\ &= M + kn \sum_{i=1}^{k(p-1)} C_i^{k(p-1)} (k')^i q^{i-1} \bmod n \\ &= M, \end{aligned} \qquad (9)$$

as desired.

1.3 RSA and Factoring Integers

The security of the RSA Cryptosystem rests on the fact that it is computationally infeasible to factor a large integer into its component primes. Obviously, if $n = p \times q$ can be factored, it is easy to compute $\phi(n) = (p-1)(q-1)$ and then compute $d = e^{-1} \bmod \phi(n)$ exactly. Therefore, to ensure the security of the RSA Cryptosystem, it is necessary to set n large enough. Nowadays, it is believed that there is no efficient algorithm that can factor any large integer in polynomial time. If a polynomial-time algorithm is found, the RSA Cryptosystem would be insecure.

Chapter 2 Factoring Algorithms

Throughout this chapter, we suppose that $n = p \times q$ is the composite integer that we want to factor, where *p*, *q* are two large primes, and *p* and *q* are roughly the same size. To attempt to factor *n*, the straightforward method is *trial division*, which divides *n* by each prime less than or equal to $\sqrt{n}$ until *p* or *q* is found. This method is guaranteed to find *p*, *q*. However, it is computationally infeasible to factor large enough *n* by using this method. For very large *n*, we need to use more effective algorithms.

Mathematicians have been attempting to find more efficient factoring algorithms for a long time, and a lot of powerful algorithms have been proposed, such as the well-known *Pollard's rho-algorithm* and $p - 1$ *algorithm*, the *continued fraction algorithm*, the *elliptic curve factoring algorithm*, the *quadratic sieve factoring algorithm* (abbreviated to *QS*) [3] and the *number field sieve* (abbreviated to *NFS*) [4]. Because of the restriction of time and computer hardware, we will focus on the quadratic sieve algorithm.

The rest of this chapter is organized as follows. Section 2.1 introduces the *Dixon's random squares algorithm*, which consists of several essential concepts still used in the QS and NFS (specifically, the concepts of a *factor base*, being *smooth* over a factor base, and finding dependencies among vectors over $\mathbb{Z}_2$). In Section 2.2, we will give a brief overview of the QS. Finally, Section 2.3 presents the *multiple polynomial quadratic sieve* (abbreviated to *MPQS*) [3], one of the most useful variants of the QS, which is widely employed in practice.

2.1 The Dixon's Random Squares Algorithm

The basic idea many factoring algorithms use is pretty simple and is described as follows. Suppose we can find two integers x and y such that $x \not\equiv \pm y \pmod{n}$ and

$$x^2 \equiv y^2 \pmod{n}. \qquad (10)$$

Then

$$(x + y)(x - y) = x^2 - y^2 \equiv 0 \pmod{n}, \qquad (11)$$

but neither $(x + y)$ nor $(x - y)$ is divisible by n. Therefore $\gcd(x + y, n)$ and $\gcd(x - y, n)$ must be non-trivial factors of n. This means that n is successfully factored.

If integers x and y satisfying (10) are produced randomly, then there is no guarantee that $x \not\equiv \pm y \pmod{n}$, and the factorization of n may not be yielded. However, what is the probability that $x \equiv \pm y \pmod{n}$? It can be proved that $x \not\equiv \pm y \pmod{n}$ with probability $\leq 1/2$. In other words, there is at least 1/2 chance that $\gcd(x + y, n)$ and $\gcd(x - y, n)$ will be nontrivial. By producing enough x and y satisfying (10), the probability of success can be increased above any desired threshold.

The Dixon's random squares algorithm is a method used to find two integers x and y satisfying (10). It begins by choosing several random integers r_i such that $r_i^2 > n$, and then proceeds to compute the values

$$f(r_i) = r_i^2 \bmod n. \qquad (12)$$

It is clear that for all r_i,

$$f(r_i) \equiv r_i^2 \pmod{n}, \qquad (13)$$

and $f(r_i) \neq r_i^2$. Therefore the right side of the congruence (13) is already a perfect square for any r_i, and of course multiplying arbitrary ones of the r_i^2's will yield a perfect square. The idea is to then find a subset S of these r_i's such that

$$\prod_{r_i \in S} f(r_i) = y^2, \text{ for some } y. \qquad (14)$$

If this can be done, then by letting

$$x = \prod_{r_i \in S} r_i, \qquad (15)$$

a congruence of the desired type follows

$$\begin{aligned} x^2 &\equiv \left(\prod_{r_i \in S} r_i\right)^2 \pmod{n} \\ &\equiv \prod_{r_i \in S} r_i^2 \pmod{n} \\ &\equiv \prod_{r_i \in S} f(r_i) \pmod{n} \\ &\equiv y^2 \pmod{n}. \end{aligned} \qquad (16)$$

Notice that the equation (14) holds if and only if every prime factor of $\prod_{r_i \in S} f(r_i)$ is used an even number of times. This then gives us an idea to find *S*: if we have known the complete factorization of each of the $f(r_i)$'s, it is easy to check to see if the product of some specific $f(r_i)$'s is a square. However, it is clearly difficult to factor each of the $f(r_i)$'s. Therefore, instead of factoring each of the $f(r_i)$'s, we just retain those $f(r_i)$'s, which can be "easily" factored, and use them. The details of doing this will be explained below. For simplicity, we first give the definitions of a *factor base* and being *smooth* over a factor base as follows:

Definition 2:

A *factor base* β is a nonempty set of prime integers. An integer α is said to be *smooth* over the factor base β if all the prime factors of α occur in β (in other words, α factors completely over β). □

Here is an example to illustrate.

Example 1:

Suppose that β = {2, 3, 7, 13} is the factor base and $\alpha = 504 = 2^3 \times 3^2 \times 7$.

Then α is smooth over β because all the prime factors of α (namely, 2, 3, 7) occur in β. □

The method of Dixon uses a factor base $\beta = \{p_1, p_2, \ldots, p_b\}$, which is a set of the b smallest primes, for an appropriate value b (it is generally recommended that $b \approx \dfrac{2^{\sqrt{r \log_2 r}}}{\ln 2\sqrt{r \log_2 r}}$). For all r_i, we then check to see if $f(r_i)$ is smooth over β. If it is, this r_i is said to be "useful", and is reserved; otherwise we throw this r_i out, and try the next one. Suppose $W = \{ r_{\alpha_1},\ r_{\alpha_2}, \ldots,\ r_{\alpha_m} \}$ is a set of r_i's with the property that $f(r_{\alpha_j})$ is smooth over β for $1 \le j \le m$, and

$$f(r_{\alpha_j}) = \prod_{k=1}^{b} p_k^{\,e_{k,j}} \qquad (17)$$

with $e_{k,j} \ge 0$, $1 \le j \le m$, $1 \le k \le b$. We then attempt to find a set S satisfying (14) from the subsets of W. Observe that every subset U of W can be mapped to a vector $\vec{z} = (z_1, z_2, \ldots, z_m) \in (\mathbb{Z}_2)^m$ as follows (where $(\mathbb{Z}_2)^m$ denotes the m-dimensional vector space over the finite field $\mathbb{Z}_2$ of 2 elements):

$$z_j = \begin{cases} 1 & \text{if } r_{\alpha_j} \in U \\ 0 & \text{if } r_{\alpha_j} \notin U \end{cases} \qquad (18)$$

for $1 \le j \le m$. It is clear that this mapping is one-to-one and onto, and

$$\begin{aligned} \prod_{r_i \in U} f(r_i) &= \prod_{j=1}^{m} \left(f(r_{\alpha_j}) \right)^{z_j} \\ &= \prod_{j=1}^{m} \left(\prod_{k=1}^{b} p_k^{\,e_{k,j}} \right)^{z_j} \\ &= \prod_{k=1}^{b} \left(\prod_{j=1}^{m} p_k^{\,e_{k,j} z_j} \right) \end{aligned}$$

$$= \prod_{k=1}^{b} p_k^{\sum_{j=1}^{m} e_{k,j} z_j} . \qquad (19)$$

As described previously, $\prod_{r_i \in U} f(r_i)$ is a perfect square if and only if

$$\sum_{j=1}^{m} e_{k,j} z_j \equiv 0 \pmod 2 \qquad (20)$$

for $1 \le k \le b$. This homogeneous linear system can be written in matrix form as

$$\begin{bmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,m} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ e_{b,1} & e_{b,2} & \cdots & e_{b,m} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \pmod 2. \qquad (21)$$

The question then becomes one of solving the equation (21). If a solution $\vec{s}$ of the equation (21) is found, the set S can then be constructed according to $\vec{s}$. It is a standard result from linear algebra [4] that if $m > b$ then the equation (21) has at least $|\mathbb{Z}_2|^{m-b} > 2$ solutions. This means that there must be at least one non-trivial solution of the equation (21),which can be used to construct a nonempty set S satisfying (14). Since the equation (21) is solved only modulo 2, it can be simplified by replacing the $e_{k,j}$ with ($e_{k,j}$ mod 2) for $1 \le j \le m$, $1 \le k \le b$.

There are many efficient algorithms for solving a homogeneous linear system over a finite field, such as *Gauss-Jordan elimination* [5], *block Lanczos algorithm* [6], and *Wiedemann algorithm* [7]. In fact, it spends most of time determining whether $f(r_i)$ is smooth over β for all r_i, instead of solving the linear system. Therefore, the real question is how to find enough r_i with $f(r_i)$ smooth over β in an efficient way.

2.2 The Quadratic Sieve Factoring Algorithm

The quadratic sieve factoring algorithm is a well-known algorithm invented by Carl

Pomerance in 1981. It was the fastest known general-purpose factoring algorithm until the number field sieve was proposed, and has been widely used in practice for a long time. Generally speaking, the QS is faster than the number field sieve for numbers having less than about 110 digits. Up to now, the QS is still the algorithm of choice for factoring large integers between 50 and 110 digits.

In reality the QS extends the ideas of the Dixon's random squares algorithm. At its kernel, the QS is essentially the same as the Dixon's method. There are two major differences between them. The first one is that instead of using the function $f(r_i) = r_i^2$ mod n, the function

$$f(r_i) = r_i^2 - n \qquad (22)$$

is used. It is easy to see that for all r_i the congruence

$$f(r_i) \equiv r_i^2 \pmod{n} \qquad (13)$$

still holds even though the function $f(r_i)$ has been replaced. Hence the new $f(r_i)$ can play the same role the old $f(r_i)$ plays. The second difference is in how to obtain integers r_i. In the Dixon's method, we simply choose the r_i's at random. In contrast, the QS uses successive integers as r_i's, such as $r_i = \lfloor \sqrt{n} \rfloor + i$, $i = 1, 2, \ldots$. It looks like that the QS is not much different from the Dixon's method. But through these slight modifications, some special tricks can be used and the running time becomes dramatically faster. In this section, we describe the details of doing this.

2.2.1 Setting Up the Factor Base

As with the Dixon's method, the QS also begins by fixing a factor base $\beta = \{p_1, p_2, \ldots, p_b\}$. Then we search for integers r_i with $f(r_i)$ is smooth over β. However, notice that not any prime can be put into β. For any $p_k \in \beta$, it must be satisfied that there exists at least one r_i such that $f(r_i)$ is divisible by p_k; otherwise there is no $f(r_i)$

divisible by this p_k, and putting it into β doesn't make sense at all. Therefore, for any $p_k \in \beta$,

$$
\begin{aligned}
& p_k \mid f(r_i), \text{ for some } r_i. \\
\Leftrightarrow\ & p_k \mid (r_i^2 - n), \text{ for some } r_i. \\
\Leftrightarrow\ & r_i^2 \equiv n \pmod{p_k}, \text{ for some } r_i. \\
\Leftrightarrow\ & n \text{ is a quadratic residue modulo } p_k. \\
\Leftrightarrow\ & \left(\frac{n}{p_k}\right) = 1. \qquad (23)
\end{aligned}
$$

Where $\left(\frac{n}{p_k}\right)$ denotes the *Legendre symbol*, which can be evaluated by using the following theorem [8].

Theorem 3

Suppose p_o is an odd prime. Then

$$\left(\frac{n}{p_o}\right) = n^{(p_o-1)/2} \bmod p_o \qquad (24)$$

□

The modular exponentiation of (24) can be computed efficiently by using the well-known *Square-and-Multiply algorithm* [8]. Thus we can decide which odd prime p_o should be put into β by easily determining whether ($n^{(p_o-1)/2}$ mod p_o) = 1. On the other hand, we should choose the primes of β as small as possible, because the $f(r_i)$'s are intuitively thought more likely smooth over β when the primes of β are smaller. At this point, we can set up our factor base as follows. First, we set β to be an empty set. Then we should put the prime 2 into β since $f(r_i) = r_i^2 - n$ is even as r_i is odd. We then proceed to start at $p_o = 3$ and check to see if ($n^{(p_o-1)/2}$ mod p_o) = 1. If it does, then p_o is added to β, otherwise it is discarded. In either case, the next prime is assigned to p_o, and the process continues until $|\beta| = b$, for an appropriate value b.

2.2.2 The Sieving Procedure

Once β has been set up completely, we begin to determine whether $f(r_i)$ is smooth over β for all r_i. As described previously, this procedure is the most time-consuming part of this kind of algorithms. Let's consider how to determine which $f(r_i)$ is smooth over β. Obviously the straightforward method is trial division, which divides $f(r_i)$ by every prime of β. However, this method is incredibly inefficient. In general, a specific $f(r_i)$ is not divisible by most primes of β. Therefore, a lot of time is wasted attempting to divide a specific $f(r_i)$ by those primes which don't actually divide it. In Dixon's method, it seems that we have no alternative but to do trial division.

In fact, the key breakthroughs occur when we change the viewpoint of the operations. Instead of focusing on one fixed $f(r_i)$ at a time and trying to divide it by all the primes of β, we fix a prime of β and determine which $f(r_i)$ are divisible by it. It is easy to see which $f(r_i) = r_i^2 - n$ is divisible by 2 by determining if r_i is odd (because $r_i^2 - n$ is divisible by 2 if and only if r_i is odd). On the other hand, for a fixed odd prime $p_o \in \beta$, we need to find all the r_i's with

$$p_o \mid (r_i^2 - n).$$

$$\Leftrightarrow \quad r_i^2 \equiv n \pmod{p_o}.$$

$$\Leftrightarrow \quad r_i \text{ is a solution to the congruence } r^2 \equiv n \pmod{p_o}. \qquad (25)$$

We already know that n is a quadratic residue modulo p_o and p_o is an odd prime, so the congruence $r^2 \equiv n \pmod{p_o}$ has exactly two solutions in $\mathbb{Z}_{p_o}$, say $s_{o,1}$ and $s_{o,2}$.

(Moreover, these two solutions are negatives of each other modulo p_o, namely $s_{o,2}$ =

$p_o - s_{o,1}$.) Let $s_o \in \{s_{o,1}, s_{o,2}\}$. Then it is clear that

$$r_i \text{ is a solution to the congruence } r^2 \equiv n \pmod{p_o}.$$

$$\Leftrightarrow \quad r_i = s_o + tp_o,\ t \in \mathbb{Z}. \qquad (26)$$

Hence it remains to consider how to compute $s_{o,1}$ and $s_{o,2}$ in a reasonable manner. Fortunately, there is an efficient method called the *Shanks-Tonelli algorithm* [1], which can be used to compute these modular square roots efficiently. Since $s_{o,1}$ and $s_{o,2}$ only depend on n and p_o, when we set up β, we also compute (and store) them for each p_o in β.

Although all the r_i's satisfying (26) can be found, it is obviously impossible to use all of them. In practice, we pick an interval and just consider the r_i's in this interval. Such an interval is called the *sieving interval*. To simplify matters, suppose the sieving interval is $\left[\left\lfloor\sqrt{n}\right\rfloor+1, \left\lfloor\sqrt{n}\right\rfloor+\delta\right]$, and $r_i = \left\lfloor\sqrt{n}\right\rfloor + i$, for $1 \le i \le \delta$. The bound δ is selected such that it is expected more than b $f(r_i)$'s which correspond to the r_i's within this range will be smooth over β. Then an array of computer memory is allocated, and for $i = 1, 2, \ldots, \delta$, $f(r_i) = r_i^2 - n$ is calculated and stored in the array. Since the r_i's are successive instead of being random, every r_i can be mapped to the index of the array element which saves the corresponding $f(r_i)$. Suppose the array elements are $M[1], M[2], \ldots, M[\delta]$. We can store the $f(r_i)$'s in such a way: for each r_i, $M[i]$ is assigned to $f(r_i)$, namely $M[r_i - \left\lfloor\sqrt{n}\right\rfloor] = f(r_i)$. Therefore, given an r_i, we can easily determine which $M[l] = f(r_i)$, $1 \le l \le \delta$.

In the next step of the algorithm, the congruence $r^2 \equiv n \pmod{p_o}$ is solved for each odd prime $p_o \in \beta$. All the r_i's satisfying

$$\left\lfloor\sqrt{n}\right\rfloor+1 \le r_i = s_o + tp_o \le \left\lfloor\sqrt{n}\right\rfloor+\delta,\ t \in \mathbb{Z} \qquad (27)$$

are then picked out, and the corresponding $M[r_i - \left\lfloor\sqrt{n}\right\rfloor]$'s are divided by p_o

repeatedly until their quotients are not divisible by p_o any more. This procedure is performed for every odd prime $p_o \in \beta$. Similarly, for every odd r_i, $f(r_i)$ is divided by 2 repeatedly until it is not divisible by 2 any more. (Even we can easily divide $f(r_i)$ by 2^c by doing bitwise right shifts if $f(r_i)$ is divisible by 2^c.) In the end all the $M[l]$'s are scanned for which $M[l] = 1$, $1 \le l \le \delta$. $M[l] = 1$ if and only if $f(\lfloor \sqrt{n} \rfloor + l)$ is smooth over β. Consequently, we can find out all the r_i's within the sieving interval with $f(r_i)$'s smooth over β.

By using this technique, every division executed is "meaningful". That is to say, $f(r_i)$ is divided by p_k if and only if $f(r_i)$ is divisible by p_k for every prime $p_k \in \beta$. Any blind division trying to divide an $f(r_i)$ by the p_k which doesn't evenly divide it. Moreover, the divisions that divide an integer by its prime factor are much faster than the other divisions. Therefore, through omitting the useless divisions, the running time is dramatically speeded up. The approach described in this subsection is called the *sieving procedure*, which yields the so-called quadratic sieve algorithm.

2.2.3 Improvements on the QS

Although the algorithm has been dramatically improved, the sieving procedure is still the most time-consuming part of the algorithm. There are several methods of accelerating the speed of sieving. One way is simply to set the size of each $f(r_i)$ as small as possible. In order to do this, observe that replacing the sieving interval $\left[\lfloor \sqrt{n} \rfloor + 1,\ \lfloor \sqrt{n} \rfloor + \delta\right]$ by $\left[\lfloor \sqrt{n} \rfloor - \frac{\delta}{2},\ \lfloor \sqrt{n} \rfloor + \frac{\delta}{2}\right]$ can effectively decrease the sizes of half the $f(r_i)$'s. Although the $f(r_i)$'s corresponding to the r_i's within $\left[\lfloor \sqrt{n} \rfloor - \frac{\delta}{2},\ \lfloor \sqrt{n} \rfloor\right]$ are negative, we can still factor them (by especially regarding (–1) as a factor). However, condition (14) must be still satisfied for some S.

In other words, except that every prime factor of $\prod_{r_i \in S} f(r_i)$ is used an even number of times, $\prod_{r_i \in S} f(r_i)$ is necessarily positive, i.e., (–1) of $\prod_{r_i \in S} f(r_i)$ is also used an even number of times. Therefore, the question can be easily solved by adding (–1) to our factor base, and the approach of finding *S* just works like the Dixon's method.

Besides the method described above, another technique usually used is to predict which $f(r_i)$ is smooth over β by using logarithmic operations. Observe that

$$f(r_i) = \prod_{k=1}^{b} p_k^{e_{k,i}}$$
$$\Rightarrow \log(f(r_i)) = \sum_{k=1}^{b} e_{k,i} \log(p_k)$$
$$\Rightarrow \log(f(r_i)) - \sum_{k=1}^{b} e_{k,i} \log(p_k) = 0 \qquad (28)$$

with $e_{k,i} \geq 0$. Thus we can probably predict whether $f(r_i)$ is smooth over β as follows. First, we compute $\log(f(r_i))$ for each r_i in the sieving interval. For every $p_k \in \beta$, we then proceed to subtract $\log(p_k)$ from $\log(f(r_i))$ for those $f(r_i)$'s are divisible by p_k. This can be done efficiently because all the r_i's satisfying (25) can be easily found. If the $\log(f(r_i))$ is reduced to 0 by this procedure, the corresponding $f(r_i)$ is necessarily smooth over β. However, this event only happens when $e_{k,i} = 0, 1$ for $1 \leq k \leq b$. If $e_{k,i} > 1$, this procedure can not yield the accurate predictions. But if we specify a reasonable threshold and only preserve the $f(r_i)$'s whose $\log(f(r_i))$'s are reduced below this threshold, we can eliminate a lot of $f(r_i)$'s which are not smooth over β. We only try to factor the remained $f(r_i)$'s. On the other hand, some $f(r_i)$'s smooth over β may also be eliminated. Therefore, the size of the threshold is a trade-off between eliminating too many "useful" $f(r_i)$'s and reserving too many "useless" $f(r_i)$'s.

2.3 The Multiple Polynomial Quadratic Sieve

The multiple polynomial quadratic sieve was suggested by Peter Montgomery and is one of the variants of the QS. As the name implies, it uses several polynomial functions instead of just one $f(r_i) = r_i^2 - n$ in the QS. A big problem in the QS is that as r_i gets large, $f(r_i) = r_i^2 - n$ also becomes large. Of course, the larger $f(r_i)$ is, the less likely it is that $f(r_i)$ is smooth over β. For fighting the drift to infinity of $f(r_i)$, the MPQS uses several polynomial functions $g_1(r_i)$, $g_2(r_i)$, …. Once the values of one polynomial get "too" large, we discard it and use a new one. This procedure not only makes the values of $g_h(r_i)$ smaller, but also makes the sieving interval and the factor base much smaller. Of course, all this is done to increase the speed of finding the $g_h(r_i)$'s smooth over β. In the MPQS, the polynomials must be chosen according to certain conditions. In the subsection below, we then proceed to describe the details of doing this.

2.3.1 Polynomials Selection

Observe that if we use polynomial functions of the form $g_h(r_i) = (r_i + b_h)^2 - n$, the values of different $g_h(r_i)$'s actually overlap. Hence selecting polynomials in such way doesn't make sense. The MPQS uses the polynomial functions of the form

$$g_h(r_i) = a_h r_i^2 + 2b_h r_i + c_h, \qquad (29)$$

where the coefficients a_h, b_h, c_h are chosen according to the guidelines below.

1. a_h is a perfect square, say $a_h = d_h^2$.
2. Choose $0 \leq b_h < a_h$ such that $b_h^2 \equiv n \pmod{a_h}$.
3. Choose c_h such that $b_h^2 - a_h c_h = n$. (Such a c_h must exist because of our choice of b_h.)

If these can be done, then

$$a_h \times g_h(r_i)$$

$$= (a_h r_i)^2 + 2(a_h r_i) b_h + a_h c_h$$

$$= (a_h r_i)^2 + 2(a_h r_i) b_h + (b_h^2 - n)$$

$$= (a_h r_i + b_h)^2 - n. \qquad (30)$$

Thus

$$a_h \times g_h(r_i) \equiv (a_h r_i + b_h)^2 \pmod{n}. \qquad (31)$$

Moreover

$$g_h(r_i) \equiv [\tilde{d}_h (a_h r_i + b_h)]^2 \pmod{n}, \qquad (32)$$

where $\tilde{d}_h = d_h^{-1} \bmod n$ (assume d_h and n are relatively prime). As with the QS, $g_h(r_i)$ is congruent to a perfect square modulo n, and this is what we want.

On the other hand, what about the factor base? Suppose the factor base $\beta = \{p_1, p_2, \ldots, p_b\}$. For any $p_k \in \beta$, the condition must be still satisfied that there exists at least one r_i such that $g_h(r_i)$ is divisible by p_k. That is, for any prime $p_k \in \beta$,

$$p_k \mid g_h(r_i), \text{ for some } g_h \text{ and } r_i. \qquad (33)$$

For $p_k = 2$, the condition (33) can always hold by restricting the values of a_h and c_h. Consider that for any odd prime $p_o \in \beta$, if $\gcd(a_h, p_o) = 1$, then

$$p_o \mid g_h(r_i), \text{ for some } r_i.$$

$$\Leftrightarrow \ p_o \mid a_h \times g_h(r_i), \text{ for some } r_i.$$

$$\Leftrightarrow \ p_o \mid [(a_h r_i + b_h)^2 - n], \text{ for some } r_i.$$

$$\Leftrightarrow \ (a_h r_i + b_h)^2 \equiv n \pmod{p_o}, \text{ for some } r_i.$$

$$\Leftrightarrow \ n \text{ is a quadratic residue modulo } p_o.$$

$$\Leftrightarrow \ \left(\frac{n}{p_o}\right) = 1.$$

$$\Leftrightarrow \ n^{(p_o - 1)/2} \bmod p_o = 1. \qquad (34)$$

If $\gcd(a_h, p_o) \neq 1$ (namely $\gcd(a_h, p_o) = p_o$), there may not exist g_h and r_i such that $g_h(r_i)$ is divisible by p_o. However, this can be avoided by choosing a_h such that for every odd prime $p_o \in \beta$, a_h is not divisible by p_o. Besides this method, if we choose a_h to be a power of a prime, there is at most one odd prime in β such that $g_h(r_i)$ is never divisible by it. Therefore, the procedure used to set up the factor base in the QS can be also used in the MPQS.

2.3.2 The Details of Choosing the Coefficients

The MPQS chooses $g_h(r_i)$'s to custom fit not only the number n, but also the length of the sieving interval. Suppose we use the sieving interval $[-\delta, \ \delta]$ of length 2δ before we change $g_h(r_i)$. Consider

$$\begin{aligned} g_h(r_i) &= a_h r_i^2 + 2b_h r_i + c_h \\ &= a_h\left(r_i^2 + 2r_i\left(\frac{b_h}{a_h}\right) + \left(\frac{b_h}{a_h}\right)^2\right) - \frac{b_h^{\ 2} - a_h c_h}{a_h} \\ &= a_h\left(r_i + \frac{b_h}{a_h}\right)^2 - \frac{n}{a_h}. \end{aligned} \qquad (35)$$

We would like to make the values of $|g_h(r_i)|$ to be as small as possible on the sieving interval. One way to do this is to have the minimum and maximum values of $g_h(r_i)$ over $[-\delta, \ \delta]$ be roughly the same in absolute values, but be opposite in sign. It is clear that the minimum value of $g_h(r_i)$ is $g_h(-\frac{b_h}{a_h}) = -\frac{n}{a_h}$. Since we choose $0 \ \le \ b_h <$ a_h, i.e., $-1 < \ -\frac{b_h}{a_h} \ \le \ 0,$ the minimum value of $g_h(r_i)$ over $[-\delta, \ \delta]$ is $g_h(-\frac{b_h}{a_h}) = -\frac{n}{a_h}$. Moreover, the maximum value of $g_h(r_i)$ over $[-\delta, \ \delta]$ appears at $r_i = \delta$, and it is

$$g_h(\delta) = a_h\left(\delta + \frac{b_h}{a_h}\right)^2 - \frac{n}{a_h}$$

$$\approx a_h \delta^2 - \frac{n}{a_h}$$

$$= \frac{(a_h\delta)^2 - n}{a_h}. \qquad (36)$$

As described above, we expect

$$\frac{n}{a_h} = \left| g_h(-\frac{b_h}{a_h}) \right| \approx |g_h(\delta)| = \frac{(a_h\delta)^2 - n}{a_h}.$$

$$\Rightarrow n \approx (a_h\delta)^2 - n.$$

$$\Rightarrow a_h\delta \approx \sqrt{2n}.$$

$$\Rightarrow a_h \approx \frac{\sqrt{2n}}{\delta}.$$

$$\Rightarrow d_h \approx \sqrt{\frac{\sqrt{2n}}{\delta}}. \qquad (37)$$

This then helps us to select a suitable d_h.

Recall that in subsection 2.3.1, the coefficients a_h, b_h, c_h must be chosen according to three guidelines. The condition 1 can be easily satisfied. If the condition 2 has been satisfied, the condition 3 can be also satisfied by choosing $c_h = \frac{b_h^2 - n}{a_h}$. Therefore, the real question is how to choose b_h according to the condition 2. To do this, n must be a quadratic residue modulo a_h. This is true if and only if n is a quadratic residue modulo d for every prime factor d of a_h [8], i.e., for every prime d with $d \mid a_h$,

$$\left(\frac{n}{d}\right) = 1. \qquad (38)$$

Hence, we would like to choose a_h with its factorization known (namely, choose d_h with its factorization known, because $a_h = d_h^2$). For convenience, we choose d_h as a prime close to $\sqrt{\frac{\sqrt{2n}}{\delta}}$ such that $\left(\frac{n}{d_h}\right) = 1$.

Once d_h has been chosen, we then proceed to solve the congruence

$$r^2 \equiv n \pmod{d_h^2}, \tag{39}$$

and set b_h to be one of the modular square roots. If the congruence

$$r^2 \equiv n \pmod{d_h} \tag{40}$$

can be solved, we can also compute the solutions of the congruence (39) by the following theorem [9].

Theorem 4 (*Hensel's Lemma*)

Suppose that $f(x)$ is a polynomial with integer coefficients and that k is an integer with $k \geq 2$. Suppose further that r is a solution of the congruence $f(x) \equiv 0 \pmod{p^{k-1}}$. Then,

(i) if $f'(r) \not\equiv 0 \pmod{p}$, then there is a unique integer t, $0 \leq t < p$, such that $f(r + tp^{k-1}) \equiv 0 \pmod{p^k}$, given by

$$t \equiv -\tilde{f}'(r)\left(\frac{f(r)}{p^{k-1}}\right) \pmod{p},$$

where $\tilde{f}'(r)$ is an inverse of $f'(r)$ modulo p;

(ii) if $f'(r) \equiv 0 \pmod{p}$ and $f(r) \equiv 0 \pmod{p^k}$, then $f(r + tp^{k-1}) \equiv 0 \pmod{p^k}$ for all integers t;

(iii) if $f'(r) \equiv 0 \pmod{p}$ and $f(r) \not\equiv 0 \pmod{p^k}$, then $f(x) \equiv 0 \pmod{p^k}$ has no solutions with $x \equiv r \pmod{p^{k-1}}$. □

Suppose $f(r) = r^2 - n$, s_h is a solution of the congruence (40) (namely, the congruence $f(x) \equiv 0 \pmod{d_h}$). By the Theorem 4, we can easily calculate one solution of congruence (39) as follows. First, compute

$$t_h = -\tilde{f}'(s_h)\left(\frac{f(s_h)}{d_h}\right) \bmod d_h, \tag{41}$$

where $\tilde{f}'(s_h)$ is an inverse of $f'(s_h)$ modulo d_h, i.e.,

$$\tilde{f}'(s_h) = (2s_h)^{-1} \bmod d_h. \qquad (42)$$

Then

$$s_h' = s_h + t_h\, d_h \bmod d_h^2 \qquad (43)$$

is a solution of the congruence (39) (namely, the congruence $f(x) \equiv 0 \pmod{d_h^2}$).

As described previously, the Shanks-Tonelli algorithm can be used to compute the modular square roots of the congruence (40). However, if we choose d_h by using the tricks below, this work can be done more efficiently. Suppose we choose d_h as a prime with $\left(\frac{n}{d_h}\right) = 1$ and

$$d_h \equiv 3 \pmod 4. \qquad (44)$$

If this can be done, then $n^{(d_h-1)/2} \equiv 1 \pmod{d_h}$ and $\frac{d_h+1}{4}$ is an integer. Thus,

$$\begin{aligned}\left(n^{(d_h+1)/4}\right)^2 &\equiv n^{(d_h+1)/2} \pmod{d_h} \\ &\equiv n\, n^{(d_h-1)/2} \pmod{d_h} \\ &\equiv n \pmod{d_h}. \end{aligned} \qquad (45)$$

That is, ($n^{(d_h+1)/4} \bmod d_h$) is a modular square root of the congruence (40). Therefore, we can set s_h to be ($n^{(d_h+1)/4} \bmod d_h$) and use it to compute s_h'.

2.3.3 Sieving

Just as the QS, we need to solve the congruence $g_h(r) \equiv 0 \pmod{p_o}$ for each odd prime p_o in the factor base β. Nothing but whenever we use a new polynomial as $g_h(r_i)$, we need to do this work again for the new polynomial. Fortunately, the congruence

$$a_h\, r^2 + 2b_h\, r + c_h \equiv 0 \pmod{p_o} \qquad (46)$$

can be easily solved by using the standard formula for solving a quadratic polynomial. (Recall that there is at most one p_o with $\gcd(a_h, p_o) \neq 1$, and we would not solve $g_h(r) \equiv 0 \pmod{p_o}$ for this p_o.)

$$\begin{aligned} r &= (2a_h)^{-1}[-2b_h \pm ((2b_h)^2 - 4a_h\, c_h)^{1/2}] \bmod p_o \\ &= 2^{-1}a_h^{-1}[-2b_h \pm 2(b_h^{2} - a_h\, c_h)^{1/2}] \bmod p_o \\ &= a_h^{-1}[-b_h \pm n^{1/2}] \bmod p_o. \end{aligned} \qquad (47)$$

Since $\gcd(a_h, p_o) = 1$, ($a_h^{-1} \bmod p_o$) exists and we can always find it. Moreover, the square roots of n modulo p_o ($n^{1/2} \bmod p_o$) can be computed by using the Shanks-Tonelli algorithm. Therefore, the sieving procedure of the MPQS just works the same way as the QS, besides using multiple polynomials instead of a single one.

Chapter 3 The Modified Multiple Polynomial Quadratic Sieve

As described above, sieving procedure is the most time-consuming part of the MPQS. Specifically, it spends most of time doing trial division in order to determine which $g(r_i)$ is smooth over β. Trial division must be applied because we don't know how many times p_j divides a given $g(r_i)$ (if $g(r_i)$ is divisible by p_j), for each $p_j \in \beta$. However, if we can explicitly compute the number of times (p_j divides a given $g(r_i)$) without doing any trial division, is it possible to improve the MPQS? Notice that if this can be done, we can determine whether a given $g(r_i)$ is smooth over β by doing logarithmic operations. We illustrate this technique with a small example.

Example 2:

Suppose that $g(r_i) = 504 = 2^3 \times 3^2 \times 7$ and $\beta = \{2, 3, 7, 13\}$. Then $g(r_i)$ is smooth over β because $\frac{504}{2^3 \times 3^2 \times 7} = 1$. On the other hand, we can conclude the same result according to the reason that 504 is divisible by 2^3, 3^2, 7 and

$$\log(504) - [3\times\log(2) + 2\times\log(3) + \log(7)] = 0. \qquad (48)$$

□

This idea is fairly simple, and we will particularly mention it latter in this chapter. Generally speaking, trial division (of large numbers) spends more time than logarithmic operations. Therefore, it remains to consider how to compute the number of times p_j divides a given $g(r_i)$ without doing any trial division. In this Chapter, we will describe our methods of doing this.

The remaining sections of this chapter are organized as follows. Section 3.1 introduces the basic ideas of our methods. In Section 3.2, we discuss how to compute the square roots of n modulo p_o^k for each odd prime $p_o \in \beta$. The results of doing this are very important and will be used in the following steps. In Section 3.3, we describe how to solve the congruence $g(r) \equiv 0 \pmod{p_o^k}$ for the $g(r)$ in the MPQS; and how to apply these results to the sieving procedure. Finally, in order to make the MPQS more practical, Section 3.4 provides a scheme to parallelize the sieving procedure.

3.1 Motivation for the Modified Multiple Polynomial Quadratic Sieve

Recall from Subsection 2.3.3 that in the sieving procedure we first solve the congruence

$$g(r) = a\,r^2 + 2b\,r + c \equiv 0 \pmod{p_o} \qquad (49)$$

for each odd prime p_o in the factor base β. By doing this, we can find all the r_i with $g(r_i)$ divisible by p_o. At this point, we already know which $g(r_i)$ is divisible by p_o, but how do we know the exponent of p_o in the prime power factorization of $g(r_i)$? It might appear to be necessary to divide $g(r_i)$ by p_o repeatedly until its quotient is not divisible by p_o (i.e. do trial division). Recall that the maximum values of $|g(r_i)|$ are about $\delta\sqrt{\frac{n}{2}}$. Thus, it is intuitively reasonable that many of $|g(r_i)|$ are almost as large as $\sqrt{n}$, and it would spend a lot of time to divide each $g(r_i)$ by its prime factors. However, if the exponent of p_o (in the factorization of $g(r_i)$) can be derived without doing any trial division, this shift may lead to a speed-up.

Now suppose that we can find the solutions of the congruence

$$g(r) = a\,r^2 + 2b\,r + c \equiv 0 \pmod{p_o^{\,k}} \qquad (50)$$

for any positive integer k, and

$$S_{o,k} = \{r_i \mid g(r_i) \equiv 0 \pmod{p_o^{\,k}}\}$$
$$= \{r_i \mid g(r_i) \text{ is divisible by } p_o^{\,k}\}. \qquad (51)$$

Notice that if $g(r_i)$ is divisible by $p_o^{\,k+1}$, it is also divisible by $p_o^{\,k}$. Thus it is clear that $S_{o,k+1} \subseteq S_{o,k}$, $k = 1, 2, \ldots$. Consider

$$D_{o,k} = S_{o,k} - S_{o,k+1}$$
$$= \{r_i \mid g(r_i) = t\,p_o^{\,k}, \gcd(t, p_o) = 1\}. \qquad (52)$$

For a particular k, if $D_{o,k}$ can be found (i.e. $S_{o,k}$, $S_{o,k+1}$ can be found), we can find all the $g(r_i)$ divisible exactly by $p_o^{\,k}$ but not divisible by $p_o^{\,k+1}$. In other words, we can find all the $g(r_i)$ in whose prime power factorization $p_o^{\,k}$ appears. Of course, the prerequisite is that the congruence (50) can be solved for any positive integer k. In the next section, we briefly discuss how to solve the simplest case of the congruence (50).

3.2 Square Roots of *n* Modulo $p_o^{\,k}$

Suppose

$$g(r) = r^2 - n, \qquad (53)$$

the simplest polynomial of the form

$$a\,r^2 + 2b\,r + c. \qquad (54)$$

(Of course, a, b, c must be chosen according to the guidelines in the MPQS.) In fact $g(r)$ is a special form of these polynomials, and it plays an important role in our method. We now consider how to solve the congruence

$$g(r) = r^2 - n \equiv 0 \pmod{p_o^{\,k}} \qquad (55)$$

for any positive integer k.

Recall that for every odd prime $p_o \in \beta$, $\left(\frac{n}{p_o}\right) = 1$. Therefore, for any positive integer k there are two square roots of n modulo $p_o{}^k$ according to the theorem below [8].

Theorem 5

Suppose that p is an odd prime, e is a positive integer, and gcd(a, p) = 1. Then the congruence $y^2 \equiv a$ (mod p^e) has no solutions if $\left(\frac{a}{p}\right) = -1$, and two solutions (modulo p^e) if $\left(\frac{a}{p}\right) = 1$. □

Since there are exactly two modular square roots, it is clear that they are negatives of each other modulo $p_o{}^k$, and we need to compute just one of them. When k = 1, as described previously the square roots of n modulo p_o can be computed efficiently by using the Shanks-Tonelli algorithm. When $k \geq 2$, the Hensel's Lemma is applied. Suppose u_{k-1} is a solution of the congruence $g(r) \equiv 0$ (mod $p_o{}^{k-1}$). Then

$$u_{k-1} ! \equiv 0 \text{ (mod } p_o). \tag{56}$$

To see this, consider that

$$(u_{k-1})^2 - n \equiv 0 \text{ (mod } p_o{}^{k-1})$$

$$\Rightarrow \quad (u_{k-1})^2 - n \equiv 0 \text{ (mod } p_o). \tag{57}$$

If $u_{k-1} \equiv 0$ (mod p_o), it implies that $n \equiv 0$ (mod p_o), which is a contradiction since gcd(n, p_o) = 1. Hence $u_{k-1} ! \equiv 0$ (mod p_o), and

$$g'(u_{k-1}) = 2\, u_{k-1}$$

$$! \equiv 0 \text{ (mod } p_o). \tag{58}$$

(Notice that p_o is an odd prime.) Therefore, case (i) of Hensel's Lemma always applies. That is, $u_k = (u_{k-1} + t_{k-1}\, p_o^{k-1})$ is a solution of the congruence $g(r) \equiv 0$ (mod p_o^{k}), given by

$$t_{k-1} \equiv -\tilde{g}'(u_{k-1})\left(\frac{g(u_{k-1})}{p_o^{k-1}}\right) \text{ (mod } p_o\text{)}, \qquad (59)$$

where $\tilde{g}'(u_{k-1})$ is an inverse of $g'(u_{k-1})$ modulo p_o.

We now consider the solution $u_{k+1} = (u_k + t_k\, p_o^{k})$ of the congruence $g(r) \equiv 0$ (mod p_o^{k+1}). First,

$$\begin{aligned} g'(u_k) &= 2\, u_k \\ &= 2\,(u_{k-1} + t_{k-1}\, p_o^{k-1}) \\ &\equiv 2\, u_{k-1} \text{ (mod } p_o\text{)} \\ &\equiv g'(u_{k-1}) \text{ (mod } p_o\text{)}. \end{aligned} \qquad (60)$$

Thus $\tilde{g}'(u_k) = \tilde{g}'(u_{k-1})$, and we don't need to compute $\tilde{g}'(u_k)$ repeatedly once $\tilde{g}'(u_{k-1})$ is computed. By extending this result, it is clear that $\tilde{g}'(u_k) = \tilde{g}'(u_1)$ for any $k \geq 1$ (where u_1 is a solution of the congruence $g(r) \equiv 0$ (mod p_o)). Suppose

$$q_{k-1} = \frac{g(u_{k-1})}{p_o^{k-1}}. \qquad (61)$$

Then we can compute q_k as follows:

$$\begin{aligned} q_k &= \frac{g(u_k)}{p_o^{k}} \\ &= \frac{u_k^{2} - n}{p_o^{k}} \\ &= \frac{(u_{k-1} + t_{k-1}\, p_o^{k-1})^2 - n}{p_o^{k}} \\ &= \frac{2\, u_{k-1}\, t_{k-1}\, p_o^{k-1} + (t_{k-1}\, p_o^{k-1})^2 + ((u_{k-1})^2 - n)}{p_o^{k}} \\ &= (t_{k-1})^2\, p_o^{k-2} + \frac{2\, u_{k-1}\, t_{k-1}\, p_o^{k-1} + g(u_{k-1})}{p_o^{k}} \\ &= (t_{k-1})^2\, p_o^{k-2} + \frac{2\, u_{k-1}\, t_{k-1} + q_{k-1}}{p_o}. \end{aligned} \qquad (62)$$

When $k \geq 3$,

$$q_k \equiv \frac{2\, u_{k-1}\; t_{k-1} + q_{k-1}}{p_o} \;\; (\text{mod } p_o). \qquad (63)$$

These results then provide an efficient method to evaluate q_k and (q_k mod p_o) through q_{k-1} when $k \geq 3$.

In the rest of this section, we discuss the size of u_k we computed. Of course, we wish to make each u_k as small as possible, i.e.

$$1 \leq u_k \leq p_o{}^k - 1, \qquad (64)$$

for $k \geq 1$. Assume

$$1 \leq u_{k-1} \leq p_o{}^{k-1} - 1. \qquad (65)$$

If we choose t_{k-1} with

$$0 \leq t_{k-1} \leq p_o - 1, \qquad (66)$$

then

$$0 \leq t_{k-1}\, p_o{}^{k-1} \leq (p_o - 1)\, p_o{}^{k-1}.$$

$$\Rightarrow 1 + 0 \leq u_{k-1} + t_{k-1}\, p_o{}^{k-1} \leq (p_o{}^{k-1} - 1) + (p_o - 1)\, p_o{}^{k-1}.$$

$$\Rightarrow 1 \leq u_k \leq p_o{}^k - 1. \qquad (67)$$

Therefore we take $u_1 \in \mathbb{Z}_{p_o}$ and

$$t_k = -\tilde{g}'(u_1)\; q_k \mod p_o, \qquad (68)$$

where $q_k = \dfrac{g(u_k)}{p_o{}^k}$. Then

$$1 \leq u_k \leq p_o{}^k - 1 \qquad (69)$$

follows for any $k \geq 1$.

3.3 Modified Sieving Procedure

Suppose

$$g(r) = a\,r^2 + 2b\,r + c, \tag{70}$$

where coefficients a, b, c satisfy the guidelines in the MPQS. In the beginning of this section, we first consider the solutions to the congruence

$$g(r) = a\,r^2 + 2b\,r + c \equiv 0 \pmod{p_o{}^k}, \tag{71}$$

for any positive integer k. Throughout this section, we will suppose that $\gcd(a, p_o) = 1$. (Recall that there is at most one p_o with $\gcd(a, p_o) \neq 1$.) Since $\gcd(a, p_o) = 1$, the congruence (71) can be solved by using the standard formula for solving a quadratic polynomial. That is,

$$\begin{aligned} r &= (2a)^{-1}[-2b \pm ((2b)^2 - 4a\,c)^{1/2}] \bmod p_o{}^k \\ &= 2^{-1}a^{-1}[-2b \pm 2(b^2 - a\,c)^{1/2}] \bmod p_o{}^k \\ &= a^{-1}[-b \pm n^{1/2}] \bmod p_o{}^k \\ &= a^{-1}[-b \pm n_o^{(k)}] \bmod p_o{}^k, \end{aligned} \tag{72}$$

where $n_o^{(k)}$ denotes the square root of n modulo $p_o{}^k$. Recall that n is a quadratic residue modulo p_o and the Legendre symbol $\left(\dfrac{n}{p_o}\right) = 1$. According to Theorem 5, there are exactly two square roots of n modulo $p_o{}^k$, namely $n_o^{(k)}$ and $p_o{}^k - n_o^{(k)}$. Therefore, the congruence (71) has exactly two solutions modulo $p_o{}^k$, say

$$s_{o,1}^{(k)} = a^{-1}[-b + n_o^{(k)}] \bmod p_o{}^k \tag{73}$$

and

$$s_{o,2}^{(k)} = a^{-1}[-b - n_o^{(k)}] \bmod p_o{}^k. \tag{74}$$

We already know from Section 3.2 that $n_o^{(k)}$ can be computed efficiently by using the Hensel's Lemma.

We now consider

$$S_{o,k} = \{r \mid g(r) \text{ is divisible by } p_o{}^k\}. \tag{75}$$

It is clear that

$$\begin{aligned} S_{o,k} &= \{r \mid g(r) \equiv 0 \ (\text{mod } p_o{}^k)\} \\ &= \{s_o^{(k)} + t\, p_o{}^k \mid s_o^{(k)} \in \{s_{o,1}^{(k)},\ s_{o,2}^{(k)}\},\ t \in \mathbb{Z}\}. \end{aligned} \tag{76}$$

Once $S_{o,k}$, $S_{o,k+1}$ are found,

$$D_{o,k} = S_{o,k} - S_{o,k+1} \tag{77}$$

is found. As described in Section 3.1, we can find all the $g(r)$ divisible exactly by $p_o{}^k$ but not divisible by $p_o{}^{k+1}$. If we wish to find $D_{o,1}$, $D_{o,2}$, …, D_{o,k_o-1}, we need to find $S_{o,1}$, $S_{o,2}$, …, S_{o,k_o} in advance. It might appear to be necessary to first compute $s_{o,1}^{(k)}$, $s_{o,2}^{(k)}$, for $k = 1, 2, …, k_o$. Fortunately, we just need to compute $s_{o,1}^{(k_o)}$, $s_{o,2}^{(k_o)}$. To see this, notice that

$$s_{o,1}^{(k_o)} \equiv s_{o,1}^{(k)} \ (\text{mod } p_o{}^k) \tag{78}$$

and

$$s_{o,2}^{(k_o)} \equiv s_{o,2}^{(k)} \ (\text{mod } p_o{}^k), \tag{79}$$

for $k \le k_o$. We will give a brief proof below.

For clarity, suppose that $a_o^{(k)}$ is an inverse of a modulo $p_o{}^k$. Then for $k \le k_o$,

$$\begin{aligned} & a\, a_o^{(k_o)} \equiv 1 \ (\text{mod } p_o{}^{k_o}). \\ \Rightarrow\ & a\, a_o^{(k_o)} \equiv 1 \ (\text{mod } p_o{}^k). \\ \Rightarrow\ & a_o^{(k_o)} \equiv a_o^{(k)} \ (\text{mod } p_o{}^k). \end{aligned} \tag{80}$$

Moreover,

$$\left(n_o^{(k_o)}\right)^2 \equiv n \ (\text{mod } p_o{}^{k_o}).$$

$$\Rightarrow \left(n_o^{(k_o)}\right)^2 \equiv n \pmod{p_o{}^k}. \tag{81}$$

In other words, $n_o^{(k_o)}$ is a square root of n modulo $p_o{}^k$. Without loss of generality, suppose

$$n_o^{(k_o)} \equiv n_o^{(k)} \pmod{p_o{}^k}. \tag{82}$$

From the discussion above, it follows that

$$\begin{aligned} s_{o,1}^{(k_o)} &= a_o^{(k_o)}[-b + n_o^{(k_o)}] \bmod p_o{}^{k_o} \\ &\equiv a_o^{(k_o)}[-b + n_o^{(k_o)}] \pmod{p_o{}^k} \\ &\equiv a_o^{(k)}[-b + n_o^{(k)}] \pmod{p_o{}^k} \\ &\equiv s_{o,1}^{(k)} \pmod{p_o{}^k}. \end{aligned} \tag{83}$$

Similarly, it can be proved that

$$s_{o,2}^{(k_o)} \equiv s_{o,2}^{(k)} \pmod{p_o{}^k}. \tag{84}$$

Therefore, we obtain the following result: for $k \le k_o$,

$$\begin{aligned} S_{o,k} &= \{s_o^{(k)} + t\, p_o{}^k \mid s_o^{(k)} \in \{s_{o,1}^{(k)},\ s_{o,2}^{(k)}\},\ t \in \mathbb{Z}\} \\ &= \{s_o^{(k_o)} + t\, p_o{}^k \mid s_o^{(k_o)} \in \{s_{o,1}^{(k_o)},\ s_{o,2}^{(k_o)}\},\ t \in \mathbb{Z}\}. \end{aligned} \tag{85}$$

That is to say, if $s_{o,1}^{(k_o)}$, $s_{o,2}^{(k_o)}$ have been computed, $S_{o,1}, S_{o,2}, \ldots, S_{o,k_o}$ are found (so are $D_{o,1}, D_{o,2}, \ldots, D_{o,k_o-1}$). $s_{o,1}^{(k_o)}$, $s_{o,2}^{(k_o)}$ can be computed by using the formula (72).

Since $n_o^{(k_o)}$ only depends on n, p_o and k_o, when we set up β, we also compute (and store) it for each p_o in β. Whenever we compute $s_{o,1}^{(k_o)}$, $s_{o,2}^{(k_o)}$ for new polynomials, the Hensel's Lemma would not be used.

We now describe the proposed scheme. To simplify matters, suppose that the sieving interval is $[1-\delta,\ \delta]$, $r_i = i - \delta$, and all the $g(r_i)$'s which correspond to the r_i's within this range are divisible by p_o at most k_o times. We first evaluate $s_{o,1}^{(k_o)}$, $s_{o,2}^{(k_o)}$ for each odd prime $p_o \in \beta$. By doing this, $S_{o,1}, S_{o,2}, \ldots, S_{o,k_o}$ can be found. Therefore, we already know which $g(r_i)$ is divisible by p_o exactly k times, for any $k \le k_o$. Then we can determine which $g(r_i)$ is smooth over β without doing any trial division. To see this, suppose that $g(r_i)$ is divisible by p_j exactly $e_{i,j}$ times with $e_{i,j} \ge 0$, for each $p_j \in \beta$. Then it is clear that

$$
\begin{aligned}
& g(r_i) \text{ is smooth over } \beta. \\
\Leftrightarrow\ & g(r_i) = \prod_{j=1}^{b} p_j^{e_{i,j}}. \\
\Leftrightarrow\ & \log(g(r_i)) = \sum_{j=1}^{b} e_{i,j} \log(p_j). \\
\Leftrightarrow\ & \log(g(r_i)) - \sum_{j=1}^{b} e_{i,j} \log(p_j) = 0. \qquad (86)
\end{aligned}
$$

Now we can apply this result to the sieving procedure as follows. First, an array of size 2δ is allocated, and suppose the array elements are $M[1], M[2], \ldots, M[2\delta]$. We then evaluate the logarithm of $g(r_i)$ for each r_i, and assign it to $M[i]$, namely $M[r_i + \delta] = \log(g(r_i))$. For every $r_i \in S_{o,k}$ and $1 - \delta \le r_i \le \delta$, $\log(p_o)$ is subtracted from $M[r_i + \delta]$, where $k \le k_o$. (That is, for every $r_i \in D_{o,k}$ and $1 - \delta \le r_i \le \delta$, $k \times \log(p_o)$ is subtracted from $M[r_i + \delta]$, where $k < k_o$. Moreover, since all the $g(r_i)$'s we consider are divisible by p_o at most k_o times, for every $r_i \in S_{o,k_o}$ and $1 - \delta \le r_i \le \delta$, $k_o \times \log(p_o)$ is subtracted from $M[r_i + \delta]$.) This procedure is performed for every odd prime $p_o \in \beta$. Similarly, if $g(r_i)$ is divisible by 2 at most k' times, $k' \times \log(2)$ must be subtracted from $M[r_i + \delta]$. In fact, k' can be easily determined by scanning $g(r_i)$ from the least significant bit towards more significant bits, until the first 1 bit is found. Finally, all the $M[i]$'s are scanned for which $M[i] = 0$, $1 \le i \le 2\delta$. Clearly, $M[i] = 0$ if and only if $g(r_i)$ is smooth over β. By using this technique, we can determine which $g(r_i)$ is smooth over β without doing any trial division (of large numbers).

At this point, we know that after sieving $M[i] = 0$ if and only if $g(r_i)$ is smooth over β. However, if $g(r_i)$ is not smooth over β, how large $M[i]$ is? The answer of this question is very useful. Generally, a logarithm of a positive integer is not a finite decimal. Therefore, when we do logarithmic operations, inaccuracy may appear. Hence, we need a reasonable bound to estimate which $M[i]$ actually equals 0. The following result is fairly simple, and we will give a brief proof of it.

Theorem 6

Suppose that the factor base $\beta = \{p_1, p_2, \ldots, p_b\}$, p_b is the maximum prime in β, $p_b < d = \sqrt{a}$ (in general, this condition holds), and $g(r_i)$ is divisible by p_j at most $e_{i,j}$ times, where $e_{i,j} \geq 0$, $j = 1, 2, \ldots, b$. Then, if $g(r_i)$ is not smooth over β,

$$q_i = g(r_i) \Big/ \left(\prod_{j=1}^{b} p_j^{e_{i,j}} \right) > p_b. \qquad (87)$$

Proof We will prove this by assuming that

$$q_i \leq p_b, \qquad (88)$$

and obtain a contradiction. Since $g(r_i)$ is not smooth over β, it must be the case that $q_i \neq 1$. Suppose q is a prime factor of q_i. Since $g(r_i)$ is divisible by q_i, it is also divisible by q. As described in Section 2.3 (because $q \leq p_b < d$ and d is prime, $\gcd(a, q) = \gcd(d^2, q) = 1$), n is a quadratic residue modulo q, i.e. $\left(\frac{n}{q}\right) = 1$. Recall from Section 2.2 that

$$\beta = \{p_j \mid p_j \text{ is prime}, p_j \leq p_b \text{ and } \left(\frac{n}{p_j}\right) = 1\}. \qquad (89)$$

Hence, it must be the case that $q \in \beta$. Without loss of generality, suppose $q = p_1$. Since

$$g(r_i) = q_i \times \left(\prod_{j=1}^{b} p_j^{e_{i,j}} \right) \qquad (90)$$

and $q = p_1$ is a prime factor of q_i, it follows that $g(r_i)$ is divisible by $p_1^{e_{i,1}+1}$. Thus, we obtain a contradiction, because $g(r_i)$ is divisible by p_1 at most $e_{i,1}$ times by our assumption. □

From Theorem 6, it is not difficult to see that after sieving $M[i] > \log(p_b)$ if and only if $g(r_i)$ is not smooth over β.

In the rest of this section, we briefly discuss how large k_o should be for each odd prime $p_o \in \beta$. Recall that all the $g(r_i)$'s which correspond to the r_i's within the sieving interval are divisible by p_o at most k_o times, and the maximum values of $|g(r_i)|$ are about $\delta\sqrt{\frac{n}{2}}$. Hence, it seems to be reasonable to set k_o to be

$$\left\lfloor \log_{p_o}\left(\delta\sqrt{\frac{n}{2}}\right) \right\rfloor \cong \left\lfloor \log_{p_o}(\delta) \right\rfloor + \left\lfloor \frac{1}{2}\left(\log_{p_o}(n) - \log_{p_o}(2)\right) \right\rfloor. \qquad (91)$$

However, this value of k_o is not appropriate. It goes without saying that the smaller k_o is, the less time it spends to compute any quantity related to k_o. (e.g. $p_o^{k_o}$ and a^{-1} mod $p_o^{k_o}$). But if k_o is too small, a lot of $g(r_i)$'s smooth over β would be considered not smooth over β. Thus there is a trade-off. Our idea is to take

$$k_o = \left\lfloor \log_{p_o}(2\delta) \right\rfloor \qquad (92)$$

such that

$$p_o^{k_o} \le 2\delta < p_o^{k_o+1}. \qquad (93)$$

Then there is at most one r_i with $1 - \delta \le r_i \le \delta$ satisfies

$$r_i \equiv s_{o,1}^{(k_o+1)} \pmod{p_o^{k_o+1}}. \quad (94)$$

Similarly, there is at most one r_i with $1-\delta \le r_i \le \delta$ satisfies

$$r_i \equiv s_{o,2}^{(k_o+1)} \pmod{p_o^{k_o+1}}. \quad (95)$$

Therefore, there are at most two r_i's with $g(r_i)$'s divisible by $p_o^{k_o+1}$. Even though there are two $g(r_i)$'s divisible by $p_o^{k_o+1}$, they may not be smooth over β. Hence, most of the $g(r_i)$'s smooth over β would not be eliminated. In reality, k_o can be set to be smaller according to the properties of $g(r_i)$'s smooth over β. For example, if the exponents of most $g(r_i)$'s (smooth over β) are always small, we can set k_o to be much smaller.

3.4 Parallel Sieving

In order to make the MPQS more practical, we parallelize the sieving procedure. In general, the way this is done is to partition the sieving interval into several subintervals, and then each processor sieves over a different subinterval. To make the implementation simple, we propose another scheme in this section. Our idea is to make each processor use different quadratic polynomial functions. In fact, this can be easily done if each processor uses different coefficients d_h. Here we will use the schemes described in the last of Subsection 2.3.2. Recall that $d_h \equiv 3 \pmod 4$. Suppose the MPQS sieves in parallel by using t computers. Then, the jth computer uses the d_h satisfying that d_h is prime and

$$d_h = 4(k\,t + j) + 3, \quad (96)$$

where $j = 1, 2, \ldots, t$, $k \in \mathbb{Z}$. It is clear that $d_h \equiv 3 \pmod 4$. Moreover, it is very easy to prove that different computers never use the same d_h's, and this is what we want. The drawback of this method is that t can not be a multiple of 3. Since the tth computer uses the d_h satisfying

$$d_h = 4(k\,t + t) + 3$$
$$= 4t(k + 1) + 3, \quad (97)$$

if t is a multiple of 3, d_h is necessarily a multiple of 3 for any $k \in \mathbb{Z}$. However, d_h needs to be prime. Therefore, t can not be a multiple of 3 in this method. Fortunately, it is not difficult to prevent t from being a multiple of 3. Hence this method is indeed practical.

Chapter 4 Experimental Results

In our research, we use the program called the *GQS* [10], which is developed by *Professor D. J. Guan* in order to implement the MPQS. The GQS is written with the C language and based on the *GMP* [11]. (GMP is a library for arbitrary precision arithmetic, and performs very well on most computers.) We modified the GQS with parallel sieving, and successfully factored a 100-digit number n (where $n = p \times q$, p, q are prime, and p and q are roughly the same size), by distributing the computations to 32 workstations in the department of CS in the *NCTU* (*National Chiao Tung University*). However, we do not implement the main idea described in Chapter 3. In the rest of this chapter, we will present the experimental results we obtained.

4.1 Environment

Throughout our experiments, the GQS was performed on the workstations in the department of CS in the NCTU. Each workstation is equipped with AMD Athlon XP 2700+ CPU (running at 2.2 GHz on average), 2 GB memory and 512 MB disk space total. Moreover, they are running operating system RedHat Linux 9.0. The sieving procedure needs to allocate many memories to sieve, and take a lot of disk space to save the sieving results. Therefore, it is important to reserve large enough memory and disk space.

4.2 Results

The asymptotic running time of the MPQS is [1]:

$$O\left(e^{(1+o(1))\sqrt{\ln(n)\ln(\ln(n))}}\right). \qquad (98)$$

The notation $o(1)$ denotes a function of n that approaches 0 as $n \to \infty$. Formula (98) can be used to estimate the time required for factoring n using one personal computer. By using one workstation, we have successfully factored the numbers having less than about 50-70 digits. We tabulate the estimated times and the execution times for some values of n in Table 1.

Table 1

$\log_{10}(n)$ / $\log_2(n)$	Estimated running times of the MPQS in one PC	Execution times involving one workstation
50 / 166	0.01 (hours)	0.16 (hours)
60 / 199	0.16 (hours)	0.11 (hours)
70 / 233	2.00 (hours)	6.00 (hours)

Execution times involving one workstation

The results show that the execution time is not close to the estimation when n has more than about 70 digits. We have also factored larger numbers in the range 80-100 digits. These were done by using parallel sieve on 32 workstations. The execution times are given as follows:

Table 2

$\log_{10}(n)$ / $\log_2(n)$	Estimated running times of the MPQS in one PC	Execution times involving 32 workstations
80 / 266	22 (hours)	1.4 (hours)
82 / 272	36 (hours)	3.4 (hours)
85 / 282	72 (hours)	6.7 (hours)
90 / 299	210 (hours)	11.3 (hours)
100 / 332	80 (days)	6.6 (days)

Execution times involving 32 workstations

The results show that sieving by using 32 workstations is not 32 times faster than using one PC. Moreover, the larger *n* is, the less the speedup is.

Chapter 5 Conclusion

In this paper, we present the methods to enhance the sieving procedure of the MPQS. The advantage of our methods is that it doesn’t need to do a lot of trial division for large numbers. Conversely, our methods need to do a lot of addition and multiplication for smaller numbers. Therefore, this shift may improve the MPQS.

For factoring RSA moduli, the NFS is recently the most-used algorithm. A lot of RSA Challenge Numbers were successfully factored by using the NFS. The asymptotic running time of the NFS is [1]:

$$O\left(e^{(1.92+o(1))(\ln(n))^{1/3}(\ln(\ln(n)))^{2/3}}\right), \qquad (99)$$

which is faster than the MPQS for numbers having more than about 125-130 digits. In fact, the NFS is improved upon by the MPQS, and it still uses the essential concepts of the MPQS. Hence, our ideas can be also applied to the NFS. If the MPQS can be improved by using our methods, the NFS can be also improved. On the other hand, the parameters of our methods (such as the size of the factor base and the length of the sieving interval) can be optimized to reduce the running time. Thus the complexity of the algorithm may be actually smaller.

References

[1] Douglas R. Stinson, *Cryptography: Theory and Practice*, 2nd Edition, Chapman & Hall/CRC, 2002.

[2] Kenneth H. Rosen, *Elementary Number Theory and Its Applications*, 5th Edition, Pearson Addison Wesley, 2004.

[3] Carl Pomerance, “The Quadratic Sieve Factoring Algorithm”, University of Georgia, 1998.

[4] A. K. Lenstra, H. W. Lenstra, Jr., et al, “The Development of the Number Field Sieve”, *Lecture Notes in Mathematics*, vol. 1554, Springer-Verlag, 1993.

[5] Stephen H. FriedBerg, et al, *Linear Algebra*, 4th Edition, Pearson Prentice Hall, 2002.

[6] Otto Bretscher, *Linear Algebra with Applications*, 3rd Edition, Pearson Prentice Hall, United States of America, 2004.

[7] P. L. Montgomery, “A block Lanczos Algorithm for finding dependencies over GF(2)”, *Advances in Cryptology – EUROCRYPT ’95*, vol. 921, pp. 106-120, 1995.

[8] D. H. Wiedemann, “Solving Sparse Linear Equations over Finite Fields”, *IEEE Trans. Information Theory*, IT-32, pp. 54-62, 1986.

[9] Ramanujachary Kumanduri, Christina Romero, *Number Theory with Computer Applications*, 1st Edition, Prentice Hall, Upper Saddle River, New Jersey, 1997.

[10] D. J. Guan, “Experience in Factoring Large Integers Using Quadratic Sieve”, 2003.

[11] The GNU MP Bignum Library, http://www.swox.com/gmp/.