

A Study on Parallel RSA Factorization

Yi-Shiung Yeh, Ting-Yu Huang, Han-Yu Lin and Yu-Hao Chang

Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

Email: {ysyeh, tingyu}@csie.nctu.edu.tw, {hanyu.cs94g, uhchang.csie93g}@nctu.edu.tw

Abstract—The RSA cryptosystem is one of the widely used public key systems. The security of it is based on the intractability of factoring a large composite integer into two component primes, which is referred to as the RSA assumption. So far, the Quadratic Sieve (QS) is the fastest and general-purpose method for factoring composite numbers having less than about 110 digits. In this paper, we present our study on a variant of the QS, i.e., the Multiple Polynomial Quadratic Sieve (MPQS) for simulating the parallel RSA factorization. The parameters of our enhanced methods (such as the size of the factor base and the length of the sieving interval) are benefit to reduce the overall running time and the computation complexity is actually lower. The experimental result shows that it only takes 6.6 days for factoring larger numbers of 100 digits using the enhanced MPQS by 32 workstations.

Index Terms—RSA, factorization, cryptosystem, Multiple Polynomial Quadratic Sieve

I. INTRODUCTION

The RSA cryptosystem [13] is one of the most popular and widely used systems in practice. Instead of relying on the discrete logarithm problem [2, 7], the security of the RSA cryptosystem is based on the intractability of factoring a large composite integer, say, 2048 bits, into two large component primes, which is referred to as the RSA assumption. That is, given a RSA modulus $n = pq$ where p and q are two large primes of the same size, say, 1024 bits, the probability for any probabilistic polynomial-time (PPT) algorithm $P(\cdot)$ to successfully factor n into p and q is at most $1/P(\cdot)$ which is negligible. It can be seen that the straightforward trial division method which divides n by each prime less than or equal to \sqrt{n} until p or q is found is time-consuming and would be infeasible when n is large enough, though, it is guaranteed to find p and q .

So far, several efficient factorization algorithms have been proposed such as Pollard's probabilistic methods [9], (ρ and $p - 1$ methods,) the Continued Fraction Method, the Elliptic Curve Method, the Number Field Sieve (NFS) [6], the Quadratic Sieve (QS) [10] and its variant, the Multiple Polynomial Quadratic Sieve (MPQS) [11, 12, 14]. Generally speaking, the QS is still faster than the NFS for numbers with no more than 110 digits and thus may be the better alternative for factoring large integers between 50 and 110 digits. In this paper, we focus on the MPQS and design an experimental

environment to simulate the parallel RSA factorization. With optimized parameters in the sieving procedure, the computation complexity of our method is lower than that of the original one, which contributes to the reduction of the overall running time.

The rest of this paper is organized as follows. Section 2 recalls some security notions of the RSA cryptosystem. Section 3 introduces some related factoring algorithms including the Dixon's Random Squares algorithm, the QS and its variant, the MPQS. We simulate the parallel RSA factorization using an enhanced MPQS in Sections 4. Finally, a conclusion with the significance of our experimental results is given in Section 5.

II. SECURITY NOTIONS OF THE RSA CRYPTOSYSTEM

This section reviews some security notions with respect to the RSA cryptosystem. That is, the RSA problem and the RSA assumption.

(i) *RSA Problem*:

Let $n = pq$ where p and q are two large primes, e an integer satisfying $\gcd(e, (p - 1)(q - 1)) = 1$ and d an integer such that $ed = 1 \pmod{(p - 1)(q - 1)}$. Given $c \equiv m^e \pmod{n}$ as the input, output the unique integer $m \in \mathbb{Z}_n$ satisfying $m \equiv c^d \pmod{n}$.

(ii) *RSA Assumption*:

Let \mathbf{G} be the RSA key generator which takes the security parameter 1^k as its input and outputs (n, e, d) . Given a RSA instance (n, e, c) , the advantage for any probabilistic polynomial-time (PPT) adversary \mathcal{A} , every positive polynomial $P(\cdot)$ and all sufficiently large k to solve the RSA problem is at most $1/P(k)$, i.e.,

$$\Pr[\mathcal{A}(n, e, c) = m, c \leftarrow m^e \pmod{n}, m \leftarrow \mathbb{Z}_n, (n, e, d) \leftarrow \mathbf{G}] \leq 1/P(k).$$

III. REVIEW OF RELATED FACTORING ALGORITHMS

To facilitate the readers with better understanding in the article, we first introduce some factoring algorithms including the Dixon's Random Squares algorithm, the QS and the MPQS which is one of the most useful variants of the QS and widely employed in practice.

A. Dixon's Random Squares Algorithm

The method is proposed by Dixon [3] in 1981 and works very well on parallel processors. Let x and y be

two integers satisfying $x \equiv \pm y \pmod{n}$ and $x^2 \equiv y^2 \pmod{n}$. Then we can derive $(x + y)(x - y) = x^2 - y^2 \equiv 0 \pmod{n}$. It can be seen that $\gcd(x + y, n)$ and $\gcd(x - y, n)$ must be non-trivial factors of n with the probability of at least $1/2$. Therefore, n is successfully factored. To find such x and y fulfilling $x^2 \equiv y^2 \pmod{n}$, the algorithm begins by choosing several random integers r_i 's such that $r_i^2 > n$, and proceeds to compute $f(r_i) = r_i^2 \pmod{n}$. It is obvious that $f(r_i) \equiv r_i^2 \pmod{n}$ and $f(r_i) \neq r_i^2$ for all the r_i 's. Let the set S contain these r_i 's, and we have $\prod_{r_i \in S} f(r_i) = y^2$

for some y , and $x = \prod_{r_i \in S} r_i$. Derived from the equality

$x = \prod_{r_i \in S} r_i$, we know that

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

Note that $\prod_{r_i \in S} f(r_i) = y^2$ for some y holds if and

only if each prime factor of $\prod_{r_i \in S} f(r_i)$ is used for even

times. Such notion benefits us to find the set S . Suppose we have known the complete factorization of each $f(r_i)$, it would be easy to check whether the product of some specific $f(r_i)$'s is a square. Yet, it is clearly difficult to factor every $f(r_i)$. Thus, we just retain and use those $f(r_i)$'s which can be "easily" factored. For simplicity, we first give the definitions of a *factor base* and being *smooth* over a factor base.

Definition 1. factor base and being smooth over a factor base

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 .

The algorithm uses a factor base β composed of b smallest primes, i.e., $\beta = \{p_1, p_2, \dots, p_b\}$. Let the set $W = \{r_{a_1}, r_{a_2}, \dots, r_{a_m}\}$ such that $f(r_{a_j})$ is smooth over β

and $f(r_{a_j}) = \prod_{k=1}^b p_k^{e_{k,j}}$ where $e_{k,j} \geq 0, 1 \leq j \leq m$ and 1

$\leq k \leq b$. We attempt to find a set S satisfying $\prod_{r_i \in S} f(r_i) = y^2$ for some y from W . Observe that every

subset $U \subseteq W$ can be mapped to a vector

$\vec{z} = (z_1, z_2, \dots, z_m) \in (Z_2)^m$ as $z_j = 1$ if $r_{a_j} \in U$ and $z_j = 0$ if $r_{a_j} \notin U$, for $1 \leq j \leq m$. Note that $(Z_2)^m$ denotes the

m -dimensional vector space over the finite field Z_2 of 2 elements. Since $\prod_{r_i \in U} f(r_i)$ is a perfect square and

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

we know $\sum_{j=1}^m e_{k,j} z_j \equiv 0 \pmod{2}$, for $1 \leq k \leq b$. This

homogeneous linear system can be written in the form of matrix as

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

To solve Eq. (1), we can use the algorithms such as the Gauss-Jordan Elimination [4], the Block Lanczos algorithm [1] and the Wiedemann algorithm [8]. If a solution \vec{z} of Eq. (1) is found, the set S can thus be constructed according to it. As a matter of fact, the crucial part is how to efficiently find enough r_i 's such that $f(r_i)$ is smooth over β , rather than solving the homogeneous linear system over a finite field.

B. Quadratic Sieve

The QS is a well-known factoring algorithm proposed by Pomerance [10] in 1981. When factoring large integers between 50 and 110 digits, the QS is undoubtedly the best choice and has been widely used in practice for a long time. In fact, the QS use similar factoring ideas to the Dixon's Random Squares algorithm except for two major differences. One is that the function of $f(r_i) = r_i^2 \pmod{n}$ is replaced with $f(r_i) = r_i^2 - n$. The other is that the QS chooses all the r_i 's as successive integers in the form of $r_i = \lfloor \sqrt{n} \rfloor + i$, for $i \in N$. Such slight modifications can lead to dramatically faster running time. More details are described as follows.

Setting up the factor base:

To set up the factor base $\beta = \{p_1, p_2, \dots, p_b\}$, the QS uses any p_k satisfying that there exists at least one r_i such

that $f(r_i)$ is divisible by p_k . Consequently, for each $p_k \in \beta$, we have

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

where $\left(\frac{n}{p_k}\right)$ stands for the Legendre symbol. To efficiently compute the Legendre symbol, we can use the well-known Square-and-Multiply algorithm [16] and thus decide which odd prime p_k should be put into β .

Sieving procedure:

To determine whether $f(r_i)$ is smooth over β for all the r_i 's, we first choose a prime of β and decide which $f(r_i)$'s are divisible by it. For instance, it is easy to see that a specific $f(r_i) = r_i^2 - n$ is divisible by two if and only if r_i is odd. For a fixed odd prime $p_k \in \beta$, we have to find all the r_i 's with $p_k \mid (r_i^2 - n)$. Then we know

$$\begin{aligned} & p_k \mid (r_i^2 - n) \\ \Leftrightarrow & r_i^2 \equiv n \pmod{p_k} \\ \Leftrightarrow & r_i \text{ is a solution to the congruence } r^2 \equiv n \pmod{p_k}. \end{aligned}$$

Since n is a quadratic residue modulo p_k and p_k is an odd prime, the congruence $r^2 \equiv n \pmod{p_k}$ has exactly two solutions in Z_{p_k} , say $s_{o,1}$ and $s_{o,2}$. In addition, these two solutions are negatives of each other modulo p_k , namely $s_{o,2} = p_k - s_{o,1}$. Let $s_o \in \{s_{o,1}, s_{o,2}\}$. It is obvious that $r_i = s_o + tp_k$, for $t \in Z$. To compute $s_{o,1}$ and $s_{o,2}$, the Shanks-Tonelli algorithm [15] is applicable and thus all the r_i 's satisfying that $r_i = s_o + tp_k$, for $t \in Z$ can be found.

In practice, we just pick an interval called sieving interval and consider the r_i 's in this interval. Let the sieving interval be $[\lfloor \sqrt{n} \rfloor + 1, \lfloor \sqrt{n} \rfloor + \delta]$ and $r_i = \lfloor \sqrt{n} \rfloor + i$, for $1 \leq i \leq \delta$, $\delta \geq b$. Allocate an one-dimension array M of size δ and store all the $f(r_i)$'s in the array. Since all the r_i 's are successive, every r_i can be mapped to the index of the array element which saves the corresponding $f(r_i)$ such as $M[r_i - \lfloor \sqrt{n} \rfloor] = f(r_i)$. Then we proceed to select those r_i 's fulfilling $\lfloor \sqrt{n} \rfloor + 1 \leq r_i = s_o + tp_k \leq \lfloor \sqrt{n} \rfloor + \delta$, for $t \in Z$ and the corresponding $M[r_i - \lfloor \sqrt{n} \rfloor]$'s are repeatedly divided by p_k until their quotients are not divisible by p_k . This procedure is performed for every odd prime $p_k \in \beta$. Similarly, for every odd r_i , $f(r_i)$ is repeatedly divided by two until it is not divisible. (Even we can easily divide $f(r_i)$ by 2^c if there are c 0's in the tail of the binary representation of $f(r_i)$.) In the end, all the $M[l]$'s are scanned for which $M[l] = 1$, for $1 \leq l \leq \delta$.

$f(\lfloor \sqrt{n} \rfloor + l)$ is smooth over β if and only if $M[l] = 1$.

Therefore, we can find out all the r_i 's such that $f(r_i)$ is smooth over β within the sieving interval.

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 $p_k \in \beta$. Moreover, the divisions that divide an integer by its prime factor are much faster than the others. It can be seen that useless divisions are discarded, so as to dramatically speed up the running time.

C. Multiple Polynomial Quadratic Sieve

The MPQS was proposed by Montgomery [14], which is one of the variants of the QS. It uses several polynomial functions $g_h(r_i)$'s, for $h \in N$ to ensure the proper size of r_i . Once the values of one polynomial get "too" large, we discard it and choose a new one. Such procedure makes each $g_h(r_i)$ smaller as well as the sieving interval and the factor base. More details are described as follows.

Selection of polynomials:

The MPQS uses the polynomial functions in the form of $g_h(r_i) = a_h r_i^2 + 2b_h r_i + c_h$ where a_h is a perfect square, say $a_h = d_h^2$, $0 \leq b_h < a_h$ such that $b_h^2 \equiv n \pmod{a_h}$, and the coefficient c_h satisfying $b_h^2 - a_h c_h = n$. Then we have

$$\begin{aligned} & a_h \cdot g_h(r_i) \\ &= (a_h r_i)^2 + 2(a_h r_i) b_h + a_h c_h \\ &= (a_h r_i)^2 + 2(a_h r_i) b_h + (b_h^2 - n) \\ &= (a_h r_i + b_h)^2 - n \\ \Leftrightarrow & a_h \cdot g_h(r_i) \equiv (a_h r_i + b_h)^2 \pmod{n} \\ \Leftrightarrow & g_h(r_i) \equiv [\tilde{d}_h (a_h r_i + b_h)]^2 \pmod{n} \end{aligned}$$

where $\tilde{d}_h = d_h^{-1} \pmod{n}$ (assume d_h and n are relatively prime). Thus, $g_h(r_i)$ is congruent to a perfect square modulo n . As to setting up the factor base, the MPQS still uses the same procedure as that of the QS.

Selection of the coefficients:

Let the sieving interval be $[-\delta, \delta]$ of length 2δ . Now consider

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

To make $|g_h(r_i)|$ to be as small as possible on the sieving interval, we attempt 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. That is,

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

$$\begin{aligned} \Rightarrow n &\approx (a_h \delta)^2 - n \\ \Rightarrow a_h \delta &\approx \sqrt{2n} \\ \Rightarrow a_h &\approx \frac{\sqrt{2n}}{\delta} \\ \Rightarrow d_h &\approx \sqrt{\frac{\sqrt{2n}}{\delta}}, \end{aligned}$$

which helps finding a proper d_h . For convenience, we choose d_h as a prime close to $\sqrt{\frac{\sqrt{2n}}{\delta}}$ such that

$$\left(\frac{n}{d_h}\right) = 1 \text{ and } d_h \equiv 3 \pmod{4}.$$

Once d_h has been chosen, we proceed to solve $r^2 \equiv n \pmod{d_h^2}$ with the assistance of the Hensel's Lemma [5] by first solving $r^2 \equiv n \pmod{d_h}$. Let $f(r) = r^2 - n$ and s_h be a solution of $r^2 \equiv n \pmod{d_h}$.

Since d_h is chosen as a prime with $\left(\frac{n}{d_h}\right) = 1$ and $d_h \equiv 3 \pmod{4}$, we know that $n^{(d_h-1)/2} \equiv 1 \pmod{d_h}$ and $\frac{d_h+1}{4}$ is an integer. Besides,

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

That is, $(n^{(d_h+1)/4} \pmod{d_h})$ is a modular square root of $r^2 \equiv n \pmod{d_h}$. Set $s_h = n^{(d_h+1)/4} \pmod{d_h}$ and compute $t_h = -\tilde{f}'(s_h) \left(\frac{f(s_h)}{d_h}\right) \pmod{d_h}$ where $\tilde{f}'(s_h)$

is an inverse of $f'(s_h)$ modulo d_h , i.e., $\tilde{f}'(s_h) = (2s_h)^{-1} \pmod{d_h}$. Consequently, we can derive $s_h' = s_h + t_h d_h \pmod{d_h^2}$ which is the solution of $r^2 \equiv n \pmod{d_h^2}$ and b_h is set to be one of the modular square roots.

Sieving procedure

Same as the QS, we have to solve the congruence $g_h(r) \equiv 0 \pmod{p_k}$ for each odd prime p_k in the factor base β . Since the MPQS uses several polynomial functions $g_h(r_i)$'s, we need to do this work repeatedly for each polynomial. Fortunately, the congruence $a_h r^2 + 2b_h r + c_h \equiv 0 \pmod{p_k}$ can be easily solved by using the standard formula for solving a quadratic polynomial. That is,

$$\begin{aligned} r &= (2a_h)^{-1} [-2b_h \pm ((2b_h)^2 - 4a_h c_h)^{1/2}] \pmod{p_k} \\ &= 2^{-1} a_h^{-1} [-2b_h \pm 2(b_h^2 - a_h c_h)^{1/2}] \pmod{p_k} \\ &= a_h^{-1} [-b_h \pm n^{1/2}] \pmod{p_k}. \end{aligned}$$

Since $\gcd(a_h, p_k) = 1$, we can always find $(a_h^{-1} \pmod{p_k})$. Further, by using the Shanks-Tonelli algorithm [15], the square roots of n modulo p_k ($n^{1/2} \pmod{p_k}$) can be efficiently computed. Therefore, the sieving procedure of the MPQS just works the same as the QS, except that the MPQS uses multiple polynomials instead of a single one.

IV. ENHANCED MULTIPLE POLYNOMIAL QUADRATIC SIEVE

For that the sieving procedure is obviously the most time-consuming part of the MPQS, we can make some modifications to optimize the running time. Suppose that $g(r_i) = 504 = 2^3 \times 3^2 \times 7$ and $\beta = \{2, 3, 7, 13\}$. We know that $g(r_i)$ is smooth over β because $\frac{504}{2^3 \times 3^2 \times 7} = 1$. On

the other hand, we can also conclude the same result according to the fact that 504 is divisible by $2^3, 3^2, 7$ and $\log(504) - [3\log(2) + 2\log(3) + \log(7)] = 0$. This idea is fairly simple and anyone can see that logarithmic operations spend less time than the trial division (of large numbers). In general, the factor base β should be large as

$$e^{\left(\frac{1}{2} + o(1)\right) \sqrt{\ln(n) \ln(\ln(n))}}$$

Thus, the running time of original MPQS is $e^{(1+o(1)) \sqrt{\ln(n) \ln(\ln(n))}}$, where each "step" executed is trial division. Moreover, the numbers $g(r_i)$ are all smaller than

approximately $n^{\frac{1}{2} + o(1)}$. Division of an l -bit integer by an m -bit integer can be done in lm bit-operations. From above, we can determine the amount of bit-operations to perform the original MPQS as

$$\begin{aligned} &e^{(1+o(1)) \sqrt{\ln(n) \ln(\ln(n))}} \left(\ln \left(n^{\frac{1}{2} + o(1)} \right) \right) \\ &\left(\ln \left(e^{\left(\frac{1}{2} + o(1)\right) \sqrt{\ln(n) \ln(\ln(n))}} \right) \right) \\ &= e^{(1+o(1)) \sqrt{\ln(n) \ln(\ln(n))}} \left(\frac{1}{2} + o(1) \right)^2 \ln(n) \sqrt{\ln(n) \ln(\ln(n))}. \end{aligned}$$

On the other hand, enhanced MPQS spends the same steps as original MPQS, but each step is logarithmic subtraction. Assume that $m \geq l$, subtraction of an l -bit integer from an m -bit integer can be done in m bit-operations. Therefore, enhanced MPQS performs the amount of bit-operations as

$$\begin{aligned} &e^{(1+o(1)) \sqrt{\ln(n) \ln(\ln(n))}} \ln \left(n^{\frac{1}{2} + o(1)} \right) \\ &= e^{(1+o(1)) \sqrt{\ln(n) \ln(\ln(n))}} \left(\ln(\ln(n)) + \ln\left(\frac{1}{2} + o(1)\right) \right). \end{aligned}$$

A. Basic Ideas

Recall that in the sieving procedure of the MPQS, we can find all the r_i 's with $g(r_i)$ divisible by each odd prime p_k in the factor base β . However, we do not know the exponent of p_k in the prime power factorization of $g(r_i)$. A flat-out way would be performing the trial division, which is considered to be inefficient for that the size of $|g(r_i)|$ are almost as large as \sqrt{n} . If the exponent of p_k

(in the factorization of $g(r_i)$) can be derived without any trial division, it will lead to a speed-up.

Assume that we can find the solutions of the congruence $g(r) = ar^2 + 2br + c \equiv 0 \pmod{p_k^t}$ for any positive integer t , and $S_{o,t} = \{r_i \mid g(r_i) \equiv 0 \pmod{p_k^t}\}$. Note that if $g(r_i)$ is divisible by p_k^{t+1} , it is also divisible by p_k^t and we know that $S_{o,t+1} \subseteq S_{o,t}$, for $t \in \mathbb{N}$. Let

$$\begin{aligned} D_{o,t} &= S_{o,t} - S_{o,t+1} \\ &= \{r_i \mid g(r_i) = wp_k^t, \gcd(w, p_k) = 1\}. \end{aligned}$$

If $D_{o,t}$ can be found for a particular t , we can find all the $g(r_i)$'s which are divisible exactly by p_k^t but not divisible by p_k^{t+1} . Namely, we can find all the $g(r_i)$'s whose prime power factorization p_k^t appears.

B. Square Roots of n Modulo p_k^t

First consider how to solve $g(r) = r^2 - n \equiv 0 \pmod{p_k^t}$ for any positive integer t . Since $\left(\frac{n}{p_k}\right) = 1$ for every odd prime $p_k \in \beta$, there are two square roots of n modulo p_k^t for any positive integer t . In addition, they are also negatives of each other modulo p_k^t . When $t = 1$, the square roots of n modulo p_k can be efficiently computed by the Shanks-Tonelli algorithm. When $t \geq 2$, the Hensel's Lemma is applicable. Let u_{t-1} be a solution of $g(r) \equiv 0 \pmod{p_k^{t-1}}$ and we can derive

$$\begin{aligned} u_{t-1} &\equiv 0 \pmod{p_k}, \\ g'(u_{t-1}) &= 2u_{t-1} \not\equiv 0 \pmod{p_k}. \end{aligned}$$

Thus, $u_t = (u_{t-1} + w_{t-1} p_k^{t-1})$ is a solution of the congruence $g(r) \equiv 0 \pmod{p_k^t}$, given by $w_{t-1} \equiv -\tilde{g}'(u_{t-1}) \left(\frac{g(u_{t-1})}{p_k^{t-1}}\right) \pmod{p_k}$ where $\tilde{g}'(u_{t-1})$ is an

inverse of $g'(u_{t-1})$ modulo p_k . Now look into the solution $u_{t+1} = (u_t + w_t p_k^t)$ of the congruence $g(r) \equiv 0 \pmod{p_k^{t+1}}$, and we have

$$\begin{aligned} g'(u_t) &= 2u_t \\ &= 2(u_{t-1} + w_{t-1} p_k^{t-1}) \\ &\equiv 2u_{t-1} \pmod{p_k} \\ &\equiv g'(u_{t-1}) \pmod{p_k}. \end{aligned}$$

Therefore, $\tilde{g}'(u_t) = \tilde{g}'(u_{t-1})$ and we can further extend this result to conclude that $\tilde{g}'(u_t) = \tilde{g}'(u_1)$, for any $t \geq 1$ (where u_1 is a solution of the congruence $g(r) \equiv 0 \pmod{p_k}$). Let $q_{t-1} = \frac{g(u_{t-1})}{p_k^{t-1}}$, and we can compute q_t as follows:

$$\begin{aligned} q_t &= \frac{g(u_t)}{p_k^t} \\ &= \frac{u_t^2 - n}{p_k^t} \end{aligned}$$

$$\begin{aligned} &= \frac{(u_{t-1} + w_{t-1} p_k^{t-1})^2 - n}{p_k^t} \\ &= \frac{2u_{t-1}w_{t-1}p_k^{t-1} + (w_{t-1}p_k^{t-1})^2 + ((u_{t-1})^2 - n)}{p_k^t} \\ &= (w_{t-1})^2 p_k^{t-2} + \frac{2u_{t-1}w_{t-1}p_k^{t-1} + g(u_{t-1})}{p_k^t} \\ &= (w_{t-1})^2 p_k^{t-2} + \frac{2u_{t-1}w_{t-1} + q_{t-1}}{p_k}. \end{aligned}$$

In fact, $q_t \equiv \frac{2u_{t-1}w_{t-1} + q_{t-1}}{p_k} \pmod{p_k}$ when $t \geq 3$.

C. Enhanced Sieving Procedure

Consider the congruence $g(r) = ar^2 + 2br + c \equiv 0 \pmod{p_k^t}$ for any positive integer t . Suppose that $\gcd(a, p_k) = 1$ and we can solve the congruence as

$$\begin{aligned} r &= (2a)^{-1}[-2b \pm ((2b)^2 - 4ac)^{1/2}] \pmod{p_k^t} \\ &= 2^{-1} a^{-1}[-2b \pm 2(b^2 - ac)^{1/2}] \pmod{p_k^t} \\ &= a^{-1}[-b \pm n^{1/2}] \pmod{p_k^t} \\ &= a^{-1}[-b \pm n_k^{(t)}] \pmod{p_k^t} \end{aligned}$$

where $n_k^{(t)}$ denotes the square root of n modulo p_k^t . Let the two solutions modulo p_k^t be $s_{o,1}^{(t)} = a^{-1}[-b + n_k^{(t)}]$ and $s_{o,2}^{(t)} = a^{-1}[-b - n_k^{(t)}]$. By the Hensel's Lemma, we can efficiently compute $n_k^{(t)}$. As to the derivation of $S_{o,t} = \{r \mid g(r) \text{ is divisible by } p_k^t\}$, it is obvious that

$$\begin{aligned} S_{o,t} &= \{r \mid g(r) \equiv 0 \pmod{p_k^t}\} \\ &= \{s_{o,1}^{(t)} + wp_k^t \mid s_{o,1}^{(t)}, s_{o,2}^{(t)}\}, w \in \mathbb{Z}. \end{aligned}$$

Once $S_{o,t}$ and $S_{o,t+1}$ are derived, $D_{o,t} = S_{o,t} - S_{o,t+1}$ is found. Similarly, if we want to find $D_{o,1}, D_{o,2}, \dots, D_{o,t_o-1}$, we has to derive $S_{o,1}, S_{o,2}, \dots, S_{o,t_o}$ by first computing $s_{o,1}^{(t_o)}$ and $s_{o,2}^{(t_o)}$ for that $s_{o,1}^{(t_o)} \equiv s_{o,1}^{(t)}$ ($\pmod{p_k^t}$) and $s_{o,2}^{(t_o)} \equiv s_{o,2}^{(t)}$ ($\pmod{p_k^t}$), for $t \leq t_o$. It can be seen that $n_k^{(t_o)}$ only depends on n, p_k and t_o . Thus, we can compute (and store) it for each p_k in β when setting up β , so as to eliminate the necessity to use the Hensel's Lemma for computing $s_{o,1}^{(t_o)}$ and $s_{o,2}^{(t_o)}$ of new polynomials. More details are described below.

Let the sieving interval be $[1 - \delta, \delta]$, $r_i = i - \delta$ and all the $g(r_i)$'s which correspond to the r_i 's within this range are divisible by p_k at most t_o times. As mentioned earlier, to determine whether a given $g(r_i)$ is divisible by p_k exactly t times, for any $t \leq t_o$, we can evaluate $s_{o,1}^{(t)}$ and $s_{o,2}^{(t)}$ for each odd prime $p_k \in \beta$ to find $S_{o,1}, S_{o,2}, \dots, S_{o,t_o}$. Afterward, we can decide which $g(r_i)$ is smooth over β without performing any trial division. To see this,

suppose that $g(r_i)$ is divisible by p_j exactly $e_{i,j}$ times with $e_{i,j} \geq 0$, for each $p_j \in \beta$. Then we have

$g(r_i)$ is smooth over β

$$\Leftrightarrow g(r_i) = \prod_{j=1}^b p_j^{e_{i,j}}$$

$$\Leftrightarrow \log(g(r_i)) = \sum_{j=1}^b e_{i,j} \log(p_j)$$

$$\Leftrightarrow \log(g(r_i)) - \sum_{j=1}^b e_{i,j} \log(p_j) = 0.$$

The result can be applied to the sieving procedure as follows. First, an array of size 2δ is allocated, and let the array elements be $M[1], M[2], \dots, M[2\delta]$. Assign the logarithm of $g(r_i)$ for each r_i to $M[i]$, i.e., $M[r_i + \delta] = \log(g(r_i))$. For every $r_i \in S_{o,t}$ and $1 - \delta \leq r_i \leq \delta$, $\log(p_k)$ is subtracted from $M[r_i + \delta]$ where $t \leq t_o$. This procedure is performed for every odd prime $p_k \in \beta$. Similarly, if $g(r_i)$ is divisible by 2 at most t' times, $t' \log(2)$ must be subtracted from $M[r_i + \delta]$. As a matter of fact, t' can be easily determined by scanning $g(r_i)$ from the least significant bit towards the most significant bit until the first 1 bit is found. In the end, all the $M[i]$'s are scanned for which $M[i] = 0$ and thus we know which $g(r_i)$ is smooth over β .

D. Parallel Sieving

In order to make the MPQS more practical, we use 32 workstations to parallelize the sieving procedure. 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. The installed operating system is RedHat Linux 9.0. We divide the sieving interval into several subintervals, and each workstation using different quadratic polynomial functions sieves over a different subinterval. That is, the j th workstation uses the coefficient d_h satisfying that d_h is prime and $d_h = 4(kt + j) + 3$ where t is the number of total workstations, $1 \leq j \leq t$, and $k \in Z$. The only constraint is that t can not be a multiple of 3. The execution time of factoring larger numbers in the range of 80-100 digits is shown as Table 1.

Although the estimated running time by using 32 workstations is not 32 times faster than that by using one PC, the parameters of our enhanced method (such as the size of the factor base and the length of the sieving interval) are benefit to reduce the overall running time and the computation complexity is actually lower.

V. CONCLUSIONS

In this paper, we have presented our study on the MPQS for simulating the parallel RSA factorization. By optimizing the size of the factor base and the length of the sieving interval, the computation complexity of our enhanced version is lower as compared with the original MPQS, which helps reducing the overall running time.

The experimental results are contributive to the analyses of the strength of the RSA assumption against the modern computer technology and should be taken into consideration on future cryptographic implementations based on the RSA cryptosystem.

TABLE 1.
EXPERIMENTAL RESULTS FOR ESTIMATED RUNNING TIME

$\log_{10}(n) / \log_2(n)$	Original MPQS by one PC	Enhanced MPQS by 32 workstations
50 / 166	0.16 (hours)	N.A.
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)
110 / 365	N.A.	49 (days)

REFERENCES

- [1] O. Bretscher, *Linear Algebra with Applications*, 3rd Ed., Prentice Hall, 2004.
- [2] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, 1976, pp. 644-654.
- [3] J.D. Dixon, "Asymptotically fast factorization of integers," *Mathematics of Computation*, Vol. 36, No. 153, 1981, pp. 255-260.
- [4] S.H. Friedberg, A.J. Insel and L.E. Spence, *Linear Algebra*, 4th Ed., Prentice Hall, 2002.
- [5] R. Kumanduri and C. Romero, *Number Theory with Computer Applications*, 1st Ed., Prentice Hall, 1997.
- [6] A.K. Lenstra and H.W. Lenstra (eds.), *The development of the number field sieve*, Lecture Notes in Mathematics, Vol. 1554, Springer-Verlag, 1993.
- [7] A. Menezes, P. Oorschot and S. Vanstone, *Handbook of applied cryptography*, CRC Press, Inc., 1997.
- [8] P.L. Montgomery, "A block Lanczos Algorithm for finding dependencies over GF(2)," *Advances in Cryptology - EUROCRYPT'95*, Springer-Verlag, 1995, pp. 106-120.
- [9] J.M. Pollard, "Theorems on factorization and primality testing," *Proceedings of the Cambridge Philosophical Society*, Vol. 76, No. 2, 1974, pp. 521-528.
- [10] C. Pomerance, "The quadratic sieve factoring algorithm", *Advances in Cryptology - EUROCRYPT'84*, Springer-Verlag, 1985, pp. 169-182.
- [11] C. Pomerance, *Cryptology and Computational Number Theory; Factoring*, AMS, Providence, RI, 1990.
- [12] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, 2nd Ed., Boston, 1994.
- [13] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, 1978, pp. 120-126.
- [14] R. Silverman, "The multiple polynomial quadratic sieve," *Mathematics of Computation*, Vol. 48, No. 177, 1987, pp. 329-339.
- [15] D.R. Stinson, *Cryptography: Theory and Practice*, 2nd Ed., CRC Press, 2002.
- [16] D.H. Wiedemann, "Solving sparse linear equations over

finite fields”, *IEEE Transactions on Information Theory*, Vol. IT-32, No. 1, 1986, pp. 54-62.

Yi-Shiung Yeh received his M.S. and Ph.D. degrees in Department of Electrical Engineering and Computer Science from University of Wisconsin-Milwaukee, U.S.A. in 1980 and 1986, respectively. He died as a professor in the department of Computer Science, National Chiao Tung University, Taiwan in 2007. His research interests include Data security & Privacy, Information & Coding Theory, Game Theory, Reliability, and Performance.

Ting-Yu Huang received his B.S. degree in electrical engineering from National Central University, Taiwan in 1990, and his M.S. degree in computer science and information engineering from National Chiao Tung University, Taiwan in 2001. Now he is a doctoral candidate in the Department of Computer Science of National Chiao Tung University, Taiwan. His research interests include Cryptanalysis and Communication Protocols.

Han-Yu Lin received his B.A. degree in economics from the Fu-Jen University, Taiwan in 2001, and his M.S. degree in information management from the Huafan University, Taiwan in 2003. Now he is a doctoral student in the Department of Computer Science of National Chiao Tung University, Taiwan. His research interests include Cryptography and Network Security.

Yu-Hao Chang received his B.S. degree in computer science and information engineering from National Central University, Taiwan in 2004, and his M.S. degree in computer science from National Chiao Tung University, Taiwan in 2006. His research interests include System Performance and Information & Coding Theory.