INFORMATICA, 2006, Vol. 17, No. 3, 445–462
© 2006 *Institute of Mathematics and Informatics, Vilnius*

Fast Parallel Exponentiation Algorithm for RSA Public-Key Cryptosystem

Chia-Long WU
Department of Aviation & Communication Electronics, Chinese Air Force Institute of Technology
Kaohsiung 82042, Taiwan
e-mail: chialongwu@seed.net.tw

Der-Chyuan LOU, Jui-Chang LAI, Te-Jen CHANG
Department of Electrical Engineering, Chung Cheng Institute of Technology
National Defense University
Tahsi, Taoyuan 33509, Taiwan
e-mail: dclou@ccit.edu.tw

Received: June 2005

Abstract. We know the necessity for information security becomes more widespread in these days, especially for hardware-based implementations such as smart cards chips for wireless applications and cryptographic accelerators. Fast modular exponentiation algorithms are often considered of practical significance in public-key cryptosystems. The RSA cryptosystem is one of the most widely used technologies for achieving information security. The main task of the encryption and decryption engine of RSA cryptosystem is to compute $M^E \bmod N$. Because the bit-length of the numbers M, E, and N would be about 512 to 1024 bits now, the computations for RSA cryptosystem are time-consuming. In this paper, an efficient technique for parallel computation of the modular exponentiation is proposed and our algorithm can reduce time complexity. We can have the speedup ratio as 1.06 or even 2.75 if the proposed technique is used. In Savas–Tenca–Koc algorithm, they design a multiplier with an insignificant increase in chip area (about 2.8%) and no increase in time delay. Our proposed technique is faster than Savas–Tenca–Koc algorithm in time complexity and improves efficiency for RSA cryptosystem.

Key words: exponentiation, parallel computing, modular arithmetic, complexity analyses, number theory, information security, cryptography, algorithm design.

1. Introduction

A public-key cryptosystem can be used to encrypt messages sent between two communicating parties so that an eavesdropper who overhears the encrypted messages will not be able to decode them in an efficient time. This cryptosystem also enables a party not to append a forged “digital signature” (Rivest *et al.*, 1978) to the end of an electronic message. It can be easily checked by anyone and forged by no one and then loses its validity if any bit of the message is altered. The cryptosystem therefore provides the authentications of both the identity of the receiver and the contents of the message, which is sent to the receiver. RSA (Lou *et al.*, 2003; Savas *et al.*, 2000) security was one of the schemes to drive industry standardization in public-key cryptography, starting in 1991 with its widely adopted Public-Key Cryptography Standards (PKCS).

A simple procedure to compute the ciphertext $C = P^E \mathrm{mod}\, M$ (P is a message, E is a public key and M is the product of two large prime numbers), which is based on the paper-and-pencil method requires $E - 1$ modular multiplications. It computes all powers of P: $P \to P^2 \to P^3 \to \ldots \to P^{E-1} \to P^E$. The computation of exponentiations using the paper-and-pencil method is very inefficient. So we propose a fast and efficient parallel method to speed up the computations of the exponentiation.

Public-key systems (such as the RSA cryptographic scheme) often involve large elements of some groups fields (such as GF(2^n) or elliptic curves) to large powers. A scalable and unified multiplier architecture shown in (Savas *et al.*, 2000) for both fields (prime GF(p) and binary extension GF(2^n) field) is proposed. The authors (E. Savas, A.F. Tenca, and C.K. Koc) had designed a multiplier with an insignificant increase in chip area (about 2.8%) and no increase in time delay. Using our proposed technique, it will be faster than (Savas *et al.*, 2000) in time complexity and take less space.

The paper is organized as follows. In Section 2 we first briefly describe RSA algorithm. Some modular arithmetic is inducted in Section 3. In Section 4 we illustrate two kinds of the binary methods. The proposed technique is described in Section 5. In Section 6 we put the complexity analyses for the proposed technique and the binary method and use some figures and tables to depict the time complexity of the two methods in detail. Finally we come to a conclusion.

2. RSA Public-Key Cryptosystem

The RSA algorithm was invented by (Rivest *et al.*, 1978). We know this algorithm becomes more widespread in these days. Here we describe the RSA algorithm as following (Lam *et al.*, 2001; Knuth, 1998; Koc, 1994):

1. Find P and Q, which are two large prime numbers (e.g., 1024 bits).
2. Choose E, which is greater than 1 and less than $[P * Q]$. These two numbers "E and $[(P - 1) * (Q - 1)]$" are relatively prime. It means the two numbers don't have any common divisor except 1 and themselves. E does not have to be a prime number, but it must be an odd number. $[(P - 1) * (Q - 1)]$ should not be a prime number because it is an even number.
3. Compute D such that $(D*E-1)$ is divisible by $[(P-1)*(Q-1)]$. Mathematicians describe this as $[D * E] \equiv 1 \bmod [(P - 1) * (Q - 1)]$ and they call D is the multiplicative inverse of E.
4. The encryption function is $C \equiv T^E \bmod (P * Q)$, where C is the ciphertext (a positive integer) and T is the plaintext (a positive integer). The message being encrypted, T, must be less than the modulus $(P * Q)$.
5. The decryption function is $T \equiv C^D \bmod (P * Q)$, where C is the ciphertext (a positive integer) and T is the plaintext (a positive integer).

The public key pair is $(P * Q, E)$. The product of $P * Q$ is the modulus. E is the public key and D is the private key, which reveals it to no one.

We can publish our public key freely. There is unknown easy method of calculating D, P, or Q respectively if the product of $P * Q$ and E are only given. If P and Q are 1024 bits long respectively, it will not even succeed to factor the modulus into P and Q efficiently and uses the most powerful computers and software technologies in existence quickly.

3. The Mathematic Preliminaries

Modular arithmetic is not a usual versatile tool discovered by K.F. Gauss (1777–1855) in 1801. Two numbers a and b are said to be equal or congruent modulo N if and only if $N|(a - b)$, i.e., if and only if their difference is exactly divisible by N.

The set of numbers congruent to a modulo N is denoted $[a]_N$. Since there are exactly N possible remainders of division by a modulo N, there are exactly N different sets $[a]_N$. Quite often these N sets are simply identified with the corresponding remainders: $[0]_N = 0, [1]_N = 1, \ldots,$ and $[N - 1]_N = N - 1$. Remainders are often called residues. Accordingly, $[a]$'s is also known as the residue classes (Knuth, 1998; Koc, 1994; Lou and Wu, 2004; Stalling, 2003).

If the relationship of $a \equiv b \bmod N$ and $c \equiv d \bmod N$ are verified, then$(a + c) \equiv (b + d) \bmod N$. The same situation is true for the multiplication. Some situations are introduced in (Seiffert, 2004; Lastovetsky and Reddy, 2004; Karatza, 2004; McIvor *et al.*, 2004; Chang and Lai, 2005; Liu *et al.*, 2005; Chen, 2005; L and Nedjah, 2005; Hwang *et al.*, 2005). An algebraic structure puts into the set:

$$\big\{[a]_N\colon\ a = 0, 1, \ldots, N - 1\big\}.$$

By the definition, shown above, we can have the following relations:

(i) $[a]_N + [b]_N = [a + b]_N$,

(ii) $[a]_N * [b]_N = [a * b]_N$.

Subtraction is defined in an analogous manner:

(iii) $[a]_N - [b]_N = [a - b]_N$.

It can be verified that the equipped set $\{[a]_N\colon\ a = 0, 1, \ldots, N - 1\}$ becomes a ring with commutative addition and multiplication. Division can't be always defined.

4. Binary Method

The binary method is also called the square-and-multiply method (Shand and Vuillemin, 1993; Wu *et al.*, 2001; Diffie and Hellman, 1976; Blum and Paar, 2001). This method includes two different types. One is the right to left binary method and the other is the left to right binary method. We describe them respectively as follows.

4.1. *The Right to Left Binary Method*

Let us considerM^E and the exponent E can be expressed in binary form (Mao, 2004; Rosen, 2000; Lou and Chang, 1996). We assume M is a plaintext and the bit-length of E is k, i.e., $E = \sum_{j=0}^{k-1}(e_j * 2^j)$. We can compute M^E by using the following algorithm to scan the exponent E from the Least Significant Bit (LSB) toward the Most Significant Bit (MSB) and work as follows.

```
Algorithm 1 (the right to left binary method):
INPUT:
    Exponent: $E = (e_{k-1}e_{k-2}\ldots e_1e_0)_2$;
    Message: $M$;
OUTPUT:
    Ciphertext: $C = M^E$
BEGIN
    $C = 1$; $S = M$;
        FOR $i = 0$ TO $k - 1$ DO              /* scan from LSB to MSB */
            BEGIN
        IF $(e_i = 1)$ $C = C * S$;            /* multiply */
        $S = S * S$;                           /* square */
            END;
END
```

4.2. *The Left to Right Binary Method*

Different from Algorithm 1, the left to right binary method computes exponent in M^E staring from the Most Significant Bit (MSB) of the exponent and proceeds to the Least Significant Bit (LSB) of the exponent (scanning the exponent from left to right), which is described as following Algorithm 2.

```
Algorithm 2 (the left to right binary method):
INPUT:
    Exponent: $E = (e_{k-1}e_{k-2}\ldots e_1e_0)_2$;
    Message: $M$;
OUTPUT:
    Ciphertext: $C = M^E$
BEGIN
    $C = 1$; $S = M$;
        FOR $i = k - 1$ DOWNTO 0 DO            /* scan from MSB to LSB */
            BEGIN
        $S = S * S$;                           /* square */
        IF $(e_i = 1)$ $C = C * S$;            /* multiply */
            END;
END
```

As the above-mentioned two algorithms have the same computations for both multiplication and squaring operations, therefore they have the same computational complexity. But there are few differences existing between these two algorithms, to be specially, the scan patterns of these two algorithms are different and squaring operations are executed in different procedures. Both Algorithm 1 and Algorithm 2 have two same states. The first state is to execute $\{C * C, C * S\}$ as the bit "1" is scanned and the second state is to execute $\{C * C\}$ when the bit "0" is scanned.

Take k-length exponent for example, for the average case, we assume the occurrence probabilities for both bits "1" and bit "0" are the same. Then, the expectation numbers for bits "1" and "0" are both $k/2$. If we set one multiplication is M and one squaring is S, the numbers of executing the exponent E are needed

$$\frac{k}{2} * M + k * S. \tag{1}$$

5. The Proposed Technique

In this section we will depict our proposed technique, which can perform modular exponentiation efficiently for RSA cryptosystem. First we analyze the following equations for the proposed technique as following (Gueron and Zuk, 2005; Menezes *et al.*, 1996; Samoa *et al.*, 2006; Yasuyuki and Kouichi, 2006):

$$\begin{gathered} c_1^d \bmod r \equiv c_1^{d_k d_{k-1} d_{k-2} d_{k-3} \ldots d_3 d_2 d_1} \bmod r \equiv m_1 \bmod r, \\ c_2^d \bmod r \equiv c_2^{d_k d_{k-1} d_{k-2} d_{k-3} \ldots d_3 d_2 d_1} \bmod r \equiv m_2 \bmod r, \\ \vdots \\ c_n^d \bmod r \equiv c_n^{d_k d_{k-1} d_{k-2} d_{k-3} \ldots d_3 d_2 d_1} \bmod r \equiv m_n \bmod r. \end{gathered}$$

Before we send the ciphertext C to the receiver, we divide it into the n equal parts $(c_1, c_2, \ldots, c_n)$. If the bit numbers of the nth part are not equal to the bit numbers of the other parts, it doesn't affect our result. The receiver deciphers c_i into m_i for1 $\leqslant i \leqslant k$. Next, we consider

$c_1^d \bmod r \equiv c_1^{d_k d_{k-1} d_{k-2} d_{k-3} \ldots d_3 d_2 d_1} \bmod r \equiv m_1 \bmod r$.

We assume that $d = (d_k d_{k-1} d_{k-2} d_{k-3} \ldots d_2 d_1)_2$, where $d_k = 1$ and $d_1, d_2, d_3, d_4, \ldots, d_{k-1}$ are either 1 or 0 in the exponent. It means this exponent d has k bits, where k is any integer. k can be either an odd number or an even number.

Now we describe a general case in the following section. If there are k bits in the exponent d, the proposed technique is depicted in detail as follows.

a.	k bits	$d_k d_{k-1}\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots d_2 d_1$
b.	n parts	N	$\dots$	i	$\dots$	3	2	1
	Each part has $\lceil \frac{k}{n} \rceil$ bits	$\lceil \frac{k}{n} \rceil$bits	$\dots$	$\lceil \frac{k}{n} \rceil$bits	$\dots$	$\lceil \frac{k}{n} \rceil$bits	$\lceil \frac{k}{n} \rceil$bits	$\lceil \frac{k}{n} \rceil$bits
c.	multiplication numbers	$\lceil \frac{k}{2n} \rceil M$	$\dots$	$\lceil \frac{k}{2n} \rceil M$	$\dots$	$\lceil \frac{k}{2n} \rceil M$	$\lceil \frac{k}{2n} \rceil M$	$\lceil \frac{k}{2n} \rceil M$

d. squaring numbers:

the first part:	$\lceil \frac{k}{n} \rceil S$,
the second part:	$2 * \lceil \frac{k}{n} \rceil S$,
the third part:	$3 * \lceil \frac{k}{n} \rceil S$,
$\vdots$	$\vdots$
the ith part	$i * \lceil \frac{k}{n} \rceil S$,
$\vdots$	$\vdots$
the nth part:	$n * \lceil \frac{k}{n} \rceil S$.

To depict the definition in the item b, k bits in the exponent d are divided into n parts from the Least Significant Bit (LSB) toward Most Significant Bit (MSB) and each part has $\lceil \frac{k}{n} \rceil$ bits, i.e., each part has the same bit numbers. If the nth part doesn't have the same bit numbers as the bit numbers of the other parts, it doesn't affect the result.

To depict the definition in item c, because each part has $\lceil \frac{k}{n} \rceil$ bits and the occurrence probability for bit "1" in each part is $\frac{1}{2}$, the multiplication numbers are $\lceil \frac{k}{2n} \rceil M$ in each part.

To depict the definition in item d, the differences of squaring numbers among parts are $\lceil \frac{k}{n} \rceil S$. If we use the proposed technique, the squaring numbers in each part are only needed $\lceil \frac{k}{n} \rceil S$, we will detail to discuss the computational complexity in Section 6.

We can have the following two remarks for our proposed technique.

1. We assume the bits of $d_1, d_2, d_3, \dots, d_{\lceil \frac{k}{n} \rceil - 2}, d_{\lceil \frac{k}{n} \rceil - 1}$, and $d_{\lceil \frac{k}{n} \rceil}$ are all 1.
2. We set $d_{\lceil \frac{k}{n} \rceil} d_{\lceil \frac{k}{n} \rceil - 1} \dots d_2 d_1$ in the exponent d is the first part, $d_{\lceil \frac{2k}{n} \rceil} d_{\lceil \frac{2k}{n} \rceil - 1} \dots d_{\lceil \frac{k}{n} \rceil + 2} d_{\lceil \frac{k}{n} \rceil + 1}$ in the exponent d is the second part, $\dots, d_{\lceil \frac{ik}{n} \rceil} d_{\lceil \frac{ik}{n} \rceil - 1} \dots d_{\lceil \frac{(i-1)k}{n} \rceil + 2} d_{\lceil \frac{(i-1)k}{n} \rceil + 1}$ in the exponent d is the ith part, $\dots, d_{\lceil \frac{(n-1)k}{n} \rceil} d_{\lceil \frac{(n-1)k}{n} \rceil - 1} \dots d_{\lceil \frac{((n-1)-1)k}{n} \rceil + 2} d_{\lceil \frac{((n-1)-1)k}{n} \rceil + 1}$ in the exponent d is the $(n-1)$th part, and $d_k d_{k-1} \dots d_{\lceil \frac{(n-1)k}{n} \rceil + 2} d_{\lceil \frac{(n-1)k}{n} \rceil + 1}$ in the exponent d is the nth part.

The modular exponentiation of computation procedures are described in detailed as following:

Procedure 1:

We input k bits in parallel.

Procedure 2:

We calculate the values of $(1)_2, (11)_2, (111)_2, \dots$, and $(d_{\lceil \frac{k}{n} \rceil} d_{\lceil \frac{k}{n} \rceil - 1} \dots d_2 d_1)_2$ respectively. The values are 1, 3, 7, 15, $\dots$, and $(2^0 + 2^1 + 2^2 + \dots + 2^{\lceil \frac{k}{n} \rceil})$ in sequence and restore them individually.

Procedure 3:

We calculate the exponentiation evaluation of the first part from d_1 to $d_{\lceil \frac{k}{n} \rceil}$ in the exponent d. The exponentiation values in sequence from d_1 to $d_{\lceil \frac{k}{n} \rceil}$ are presented $2^1, 2^2, 2^3, \ldots$, and $2^{\lceil \frac{k}{n} \rceil}$.

Procedure 4:

The MSB in the exponentiation evaluation of the first part is multiplied by the value stored in the first part. It means the product of $2^{\lceil \frac{k}{n} \rceil} * (2^0 + 2^1 + 2^2 + \ldots + 2^{\lceil \frac{k}{n} \rceil})$ is $(11111\ldots 00000\ldots)_2$ and there are $\lceil \frac{k}{n} \rceil$ consecutive 1 and $\lceil \frac{k}{n} \rceil$ consecutive 0 in this number respectively.

Procedure 5:

We calculate the exponentiation evaluation of the second part from $d_{\lceil \frac{k}{n} \rceil + 1}$ to $d_{\lceil \frac{2k}{n} \rceil}$ in the exponent d. Because the exponentiation evaluation of the first part has already executed in the procedure 3 in advance, we can directly calculate the exponentiation evaluation of the second part here. The exponentiation values in sequence from $d_{\lceil \frac{k}{n} \rceil + 1}$ to $d_{\lceil \frac{2k}{n} \rceil}$ are presented $2^{\lceil \frac{k}{n} \rceil + 1}, 2^{\lceil \frac{k}{n} \rceil + 2}, \ldots$, and $2^{\lceil \frac{2k}{n} \rceil}$.

Procedure 6:

The MSB in the exponentiation evaluation in the second part is multiplied by the value stored in the first part. It means the product of $2^{\lceil \frac{2k}{n} \rceil} * (2^0 + 2^1 + 2^2 + \ldots + 2^{\lceil \frac{k}{n} \rceil})$ is $(11111\ldots 0000000000\ldots)_2$ and there are $\lceil \frac{k}{n} \rceil$ consecutive 1 and $\lceil \frac{2k}{n} \rceil$ consecutive 0 in this number respectively.

Procedure 7:

We calculate the exponentiation evaluation of the third part from $d_{\lceil \frac{2k}{n} \rceil + 1}$ to $d_{\lceil \frac{3k}{n} \rceil}$ in the exponent d. The exponentiation values in sequence from $d_{\lceil \frac{2k}{n} \rceil + 1}$ to $d_{\lceil \frac{3k}{n} \rceil}$ are presented $2^{\lceil \frac{2k}{n} \rceil + 1}, 2^{\lceil \frac{2k}{n} \rceil + 2}, \ldots$, and $2^{\lceil \frac{3k}{n} \rceil}$.

Procedure 8:

The MSB in the exponentiation evaluation of the third part is multiplied by the value stored in the first part. It means the product of $2^{\lceil \frac{3k}{n} \rceil} * (2^0 + 2^1 + 2^2 + \ldots + 2^{\lceil \frac{k}{n} \rceil})$ is $(11111\ldots 0000000000\ldots)_2$ and there are $\lceil \frac{k}{n} \rceil$ consecutive 1 and $\lceil \frac{3k}{n} \rceil$ consecutive 0 in this number respectively.

Procedure 9:

We calculate the exponentiation evaluation of the fourth part from $d_{\lceil \frac{3k}{n} \rceil + 1}$ to $d_{\lceil \frac{4k}{n} \rceil}$ in the exponent d. The exponentiation values in sequence from $d_{\lceil \frac{3k}{n} \rceil + 1}$ to $d_{\lceil \frac{4k}{n} \rceil}$ are presented $2^{\lceil \frac{3k}{n} \rceil + 1}, 2^{\lceil \frac{3k}{n} \rceil + 2}, \ldots$, and $2^{\lceil \frac{4k}{n} \rceil}$.

$\vdots$

Procedure $(2n-1)$:

We calculate the exponentiation evaluation of $(n-1)$th part from $d_{\lceil \frac{[(n-1)-1]k}{n} \rceil + 1}$ to $d_{\lceil \frac{(n-1)k}{n} \rceil}$ in the exponent d. The exponentiation values in sequence from $d_{\lceil \frac{[(n-1)-1]k}{n} \rceil + 1}$ to $d_{\lceil \frac{(n-1)k}{n} \rceil}$ are presented $2^{\lceil \frac{[(n-1)-1]k}{n} \rceil + 1}, 2^{\lceil \frac{[(n-1)-1]k}{n} \rceil + 2}, \ldots$, and $2^{\lceil \frac{(n-1)k}{n} \rceil}$.

Procedure $(2n)$:

The MSB in the exponentiation evaluation of the $(n-1)$th part is multiplied by the value stored in the first part. It means the product of $2^{\lceil\frac{(n-1)k}{n}\rceil} * (2^0 + 2^1 + 2^2 + \ldots + 2^{\lceil\frac{k}{n}\rceil})$ is $(11111\ldots00000\ldots)_2$ and there are $\lceil\frac{k}{n}\rceil$ consecutive 1 and $\lceil\frac{(n-1)k}{n}\rceil$ consecutive 0 in this number respectively.

Procedure $(2n+1)$:

To sum up, we can obtain the result of $\sum_{i=1}^{n-1}(2^{\lceil\frac{ik}{n}\rceil} * \sum_{j=0}^{\frac{k}{n}} 2^j)$, where k is the bit-length of the exponent, n means parts coming from Procedure 2, Procedure 4, Procedure 6, …, and Procedure $(2n)$. Then we can get the answer $(111\ldots111)_2$ and there are k consecutive 1 in this number.

6. Complexity Analyses

Now we generalize the above procedures from Procedure 1 to Procedure $(2n+1)$ and analyze the complexity of the proposed technique in detail as following:

In Procedure 1, the big O is 1 because the bits in exponent d are put in parallel.

In Procedure 2, this is the most complex condition when we assume consecutive one from d_1 to $d_{\lceil\frac{k}{n}\rceil}$ in the exponent d. In normal condition, there are $\lceil\frac{k}{n}\rceil$ bits on the first part, and the expectation numbers for bits "1" is $\frac{1}{2} * \lceil\frac{k}{n}\rceil$. So the numbers of multiplication and squaring from $d_1, d_2, d_3, \ldots, d_{\lceil\frac{k}{n}\rceil-2}, d_{\lceil\frac{k}{n}\rceil-1}$, and $d_{\lceil\frac{k}{n}\rceil}$ in the first part are needed

$$\frac{1}{2} * M * \left\lceil\frac{k}{n}\right\rceil + \left\lceil\frac{k}{n}\right\rceil * S. \tag{2}$$

In Procedure 3, because it needs $\lceil\frac{k}{n}\rceil S$ squarings to get the result for the exponentiation evaluation of the first part, the computational complexity is

$$\left\lceil\frac{k}{n}\right\rceil * S. \tag{3}$$

Before we explain Procedure 4 and Procedure 5, we assume that

$$\frac{k}{n} * S \approx M, \tag{4}$$

where S and M denote one squaring and one multiplication respectively.

If Eq. 4 is true, the following statement can be achieved. When the multiplication of "the value stored in the first part" and "the MSB in the exponentiation evaluation of the first part" is completed, the exponentiation evaluation of the second part is also completed at the same time. It means Procedure 4 and Procedure 5 can be executed at the same time. Here we can only consider the multiplication between the value stored in the first part and the MSB in the exponentiation evaluation of the first part. The computational complexity is needed one multiplication. It can be denoted by $1M$.

In Procedure $(2n+1)$, because the time of executing addition is much smaller than the time of executing multiplication, we skip the computational complexity of addition.

For above assumption, we know Procedure 6 and Procedure 7 can be achieved at the same time. Similarly, Procedure 8, Procedure 9 . . ., and Procedure $(2n-1)$, Procedure $(2n)$ can be achieved at the same time. We can only consider one multiplication every two procedures, then it is merely needed:

$$(n-2) * M. \tag{5}$$

To sum up Eqs. 2, 3, and 5, we can get:

$$M * \left\lceil \frac{k}{2n} \right\rceil + 2 * \left\lceil \frac{k}{n} \right\rceil * S + (n-2) * M, \tag{6}$$

for the proposed technique.

$$M * \left(\frac{k}{2n} + 1 \right) + 2 * \left(\frac{k}{n} + 1 \right) * S + (n-2) * M. \tag{7}$$

The “+1” inside the first item and the second item in Eq. 7 means the maximum numbers which we can obtain for the first item and the second item in Eq. 6.

Let $\frac{k}{n} * S$ in Eq. 4 is substituted for M in Eq. 7, we derive

$$\frac{k^2 * S}{2n^2} + \frac{k * S}{n} + \frac{2k * S}{n} + 2S + k * S - \frac{2k * S}{n}, \tag{8}$$

then we simplify Eq. 8 and get Eq. 9.

$$\frac{k^2 * S}{2n^2} + \frac{k * S}{n} + 2S + k * S, \tag{9}$$

where $n \geqslant 1$ and $k > 0$.

That means if the number n is greater than one, it will take less time. Assuming the proportion between number of multiplication and the number of squaring is $\frac{n}{lk}$, we can get:

$$\frac{k}{n} * S * l = M. \tag{10}$$

Let $\frac{k*S*l}{n}$ in Eq. 10 is substituted for M in Eq. 7. We derive:

$$\frac{l * k^2 * S}{2n^2} + \frac{3k * S}{n} + (1 - \frac{2}{n}) l * k * S + 2S. \tag{11}$$

If we let $\frac{k*S}{n}$ in Eq. 4 is substituted for M in Eq. 1, we derive:

$$\frac{k}{2} * M + k * S = \frac{k}{2} * \left(\frac{k * S}{n} \right) + k * S = \frac{k^2 * S}{2n} + k * S. \tag{12}$$

If we let $\frac{k*S*l}{n}$ in Eq. 10 is substituted for M in Eq. 1, we derive:

$$\frac{k}{2} * M + k * S = \frac{k}{2}\Big(\frac{k}{n} * S * l\Big) + k * S = \frac{k^2 * S * l}{2n} + k * S. \tag{13}$$

In other words, using the binary method, if l is equal to one and Eq. 4 is assumed, we can obtain Eq. 12. If l is not equal to one, we can obtain Eq. 13. Similarly, using our proposed technique, if l is equal to one and the equation $\frac{k}{n} * S \approx M$ is assumed, we can obtain Eq. 9. If l is not equal to one, we can obtain Eq. 11.

Before we observe the variations among different k bits between binary method and proposed technique, we define the speedup ratio of our method as

$$\frac{\text{equation}(13)}{\text{equation}(13) - \text{equation}(11)}. \tag{14}$$

We compare the different part n and the speedup ratios between binary method and proposed technique when $l = 1$ and $k = 256, 512, 1024,$ and 2048 bits respectively in Table 1.

We discuss l when it is 10 or 0.1 respectively between different parts n and the speedup ratio are shown in Table 2 and 3 respectively. Other associated multiplication numbers are shown from Figs. 1 to 8.

We observe the curves from Fig. 1 to Fig. 8 and the speedup ratios from Table 1 to Table 3. No matter how l is an integer or a fraction, the curves we proposed are always below the line using binary method. Comparing the binary multiplication with the technique proposed in (Chang and Lai, 2005), when the bit-length of the exponent is 1024 bits, the speedup ratio of the Chang–Lai algorithm is 1.08 and the speedup ratio of our proposed algorithm is 1.10. If k is much larger, where k is the bit-length of the exponent, the speedup ratio of the proposed algorithm can achieve 2.75. We can therefore efficiently reduce the overall time complexity.

Table 1

Compare the squaring numbers and the speedup ratios between binary method and proposed technique when $l = 1$ and $k = 256, 512, 1024, 2048$ bits respectively

Parts	The speedup ratios			
	256	512	1024	2048
$n = 2$	2.06	2.03	2.02	2.01
$n = 4$	1.39	1.36	1.35	1.34
$n = 8$	1.23	1.18	1.16	1.15
$n = 16$	1.21	1.14	1.10	1.08
$n = 32$	1.30	1.17	1.10	1.07

Table 2

Compare the squaring numbers and the speedup ratios between binary method and proposed technique when $l = 10$ and $k = 256, 512, 1024, 2048$ bits respectively

Parts	The speedup ratios			
	256	512	1024	2048
$n = 2$	2.01	2.00	2.00	2.00
$n = 4$	1.36	1.35	1.34	1.34
$n = 8$	1.21	1.18	1.16	1.15
$n = 16$	1.21	1.13	1.10	1.08
$n = 32$	1.35	1.17	1.10	1.06

Table 3

Compare the squaring numbers and the speedup ratios between binary method and proposed technique when $l = 0.1$ and $k = 256, 512, 1024, 2048$ bits

Parts	The speedup ratios			
	256	512	1024	2048
$n = 2$	2.75	2.34	2.16	2.08
$n = 4$	1.62	1.48	1.41	1.37
$n = 8$	1.34	1.26	1.20	1.17
$n = 16$	1.23	1.17	1.13	1.10
$n = 32$	1.17	1.14	1.10	1.07

7. Example

Here we put an example to depict the process of proposed technique. We assume there are nineteen consecutive one in the exponent d. For simplicity, we call them $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, a_{17}, a_{18}$, and a_{19} from Least Significant Bit (LSB) to Most Significant Bit (MSB). We divide the exponent d into four parts from LSB to MSB. In other words, there are five consecutive one in the first, second, and third parts respectively and there are four consecutive one in the fourth part.

The detail procedures are shown as follows.

Procedure 1:

We put nineteen consecutive one in parallel.

Procedure 2:

We calculate the values of $(1)_2$, $(11)_2$, $(111)_2$, $(1111)_2$, and $(11111)_2$ respectively. The values are 1, 3, 7, 15, and 31 in sequence and restore them respectively.

Procedure 3:

We calculate the exponentiation evaluation of the first part from a_1 to a_5 in the exponent d. The exponentiation values in sequence from a_1 to a_5 are presented $2^1, 2^2, 2^3, 2^4$, and 2^5 respectively.

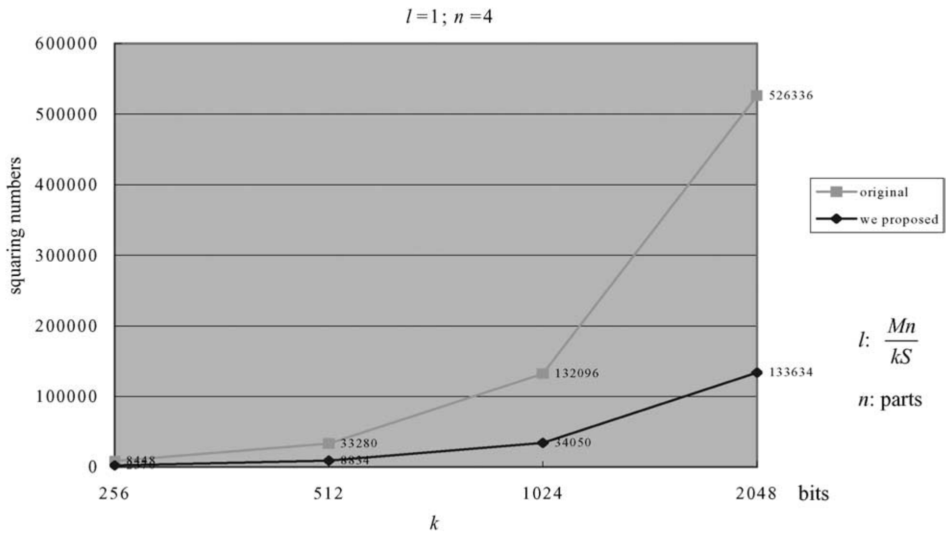

Fig. 1. The relationship between squaring numbers and k bits when $l = 1$, $n = 4$.

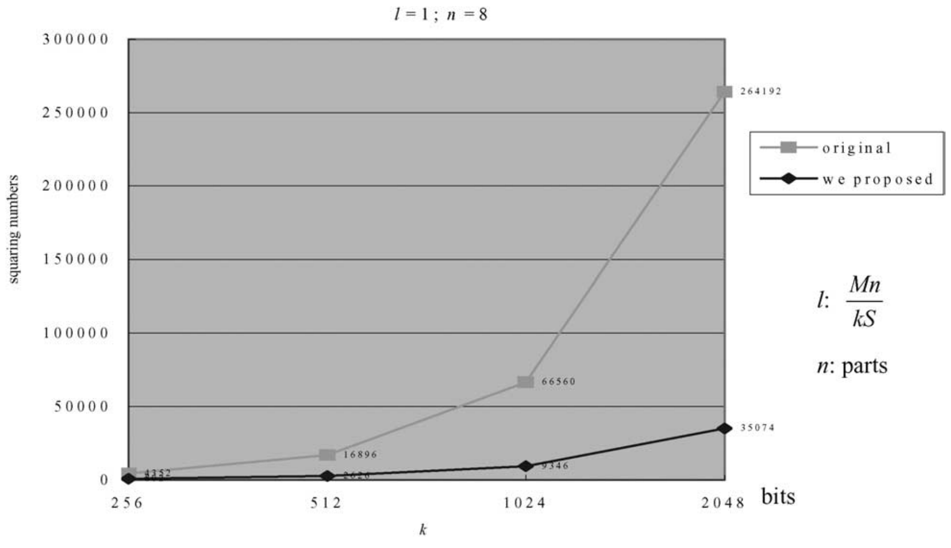

Fig. 2. The relationship between squaring numbers and k bits when $l = 1$, $n = 8$.

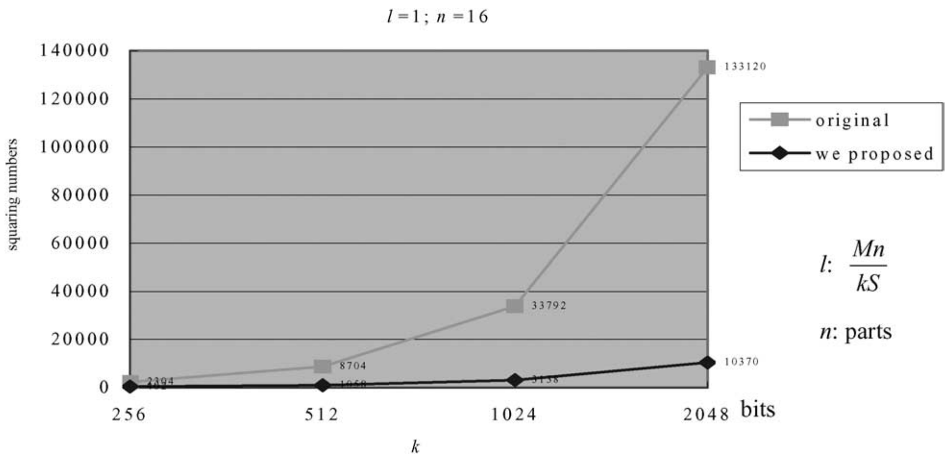

Fig. 3. The relationship between squaring numbers and k bits when $l = 1$, $n = 16$.

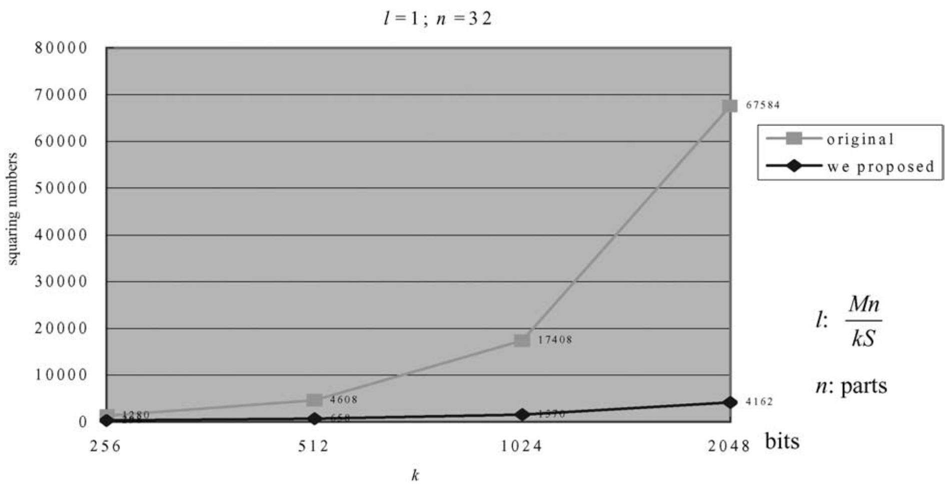

Fig. 4. The relationship between squaring numbers and k bits when $l = 1$, $n = 32$.

Procedure 4:

The MSB in the exponentiation evaluation of the first part is multiplied by the value stored in the first part. $2^5 * 31 = 992$, i.e., this is the result of

$$(1111100000)_2. \tag{15}$$

Procedure 5:

We calculate the exponentiation evaluation of the second part from a_6 to a_{10} in the exponent d. Because the exponentiation evaluation of the first part is executed in advance, we can directly calculate the exponentiation evaluation of the second

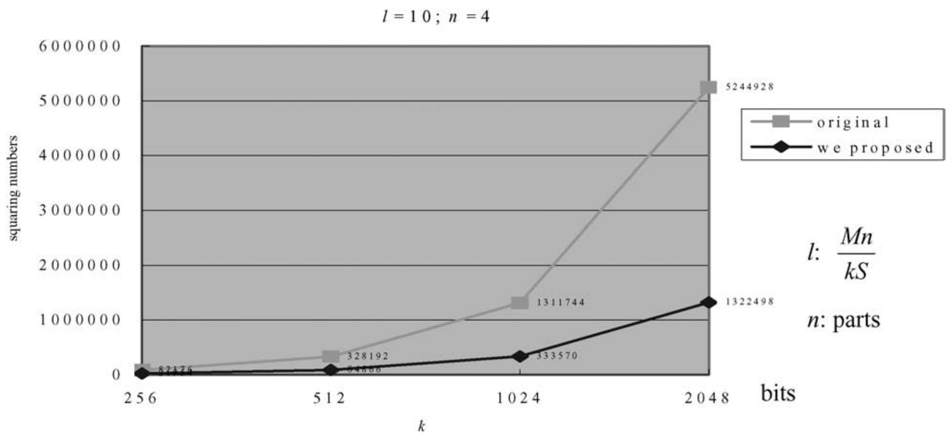

Fig. 5. The relationship between squaring numbers and k bits when $l = 10$, $n = 4$.

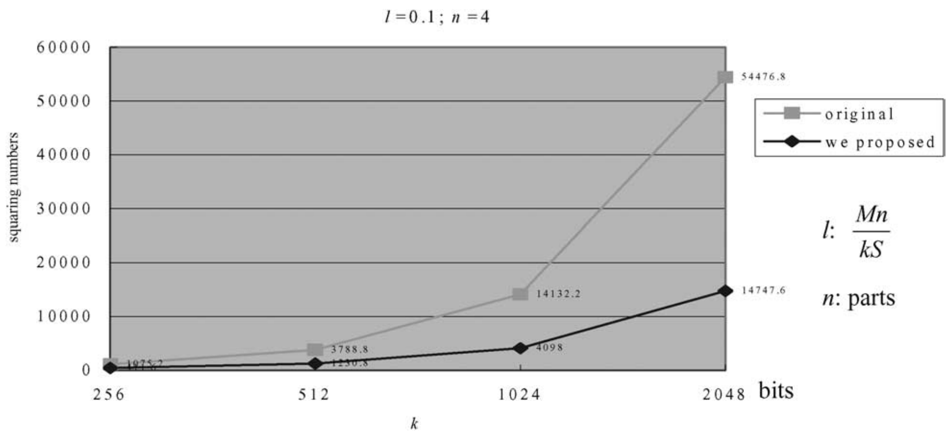

Fig. 6. The relationship between squaring numbers and k bits when $l = 0.1$, $n = 4$.

part here. The exponentiation values in sequence from a_6 to a_{10} are presented $2^6, 2^7, 2^8, 2^9$, and 2^{10} respectively.

Procedure 6:

The MSB in the exponentiation evaluation of the second part is multiplied by the value stored in the first part. $2^{10} * 31 = 31744$, i.e., this is the result of

$$(111110000000000)_2. \tag{16}$$

Procedure 7:

We calculate the exponentiation evaluation of the third part from a_{11} to a_{15} in the exponent d. The exponentiation values in sequence from a_{11} to a_{15} are presented $2^{11}, 2^{12}, 2^{13}, 2^{14}$, and 2^{15} respectively.

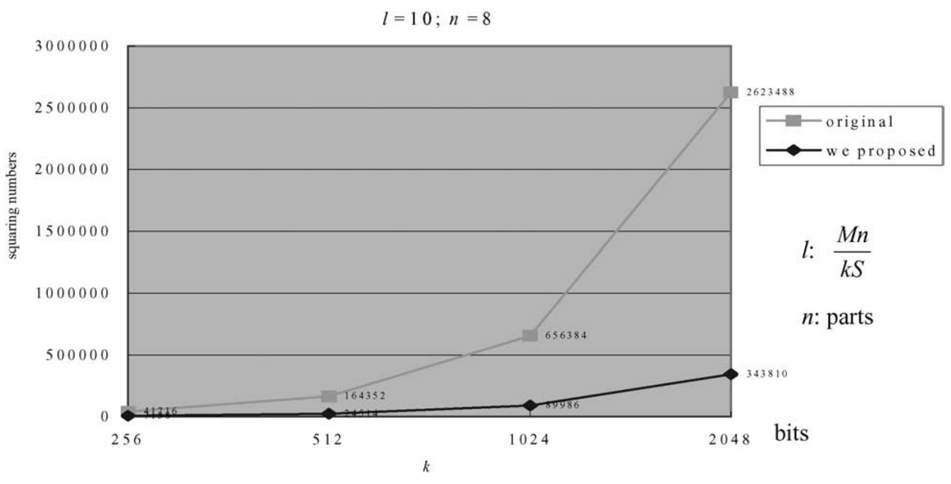

Fig. 7. The relationship between squaring numbers and k bits when $l = 10$, $n = 8$.

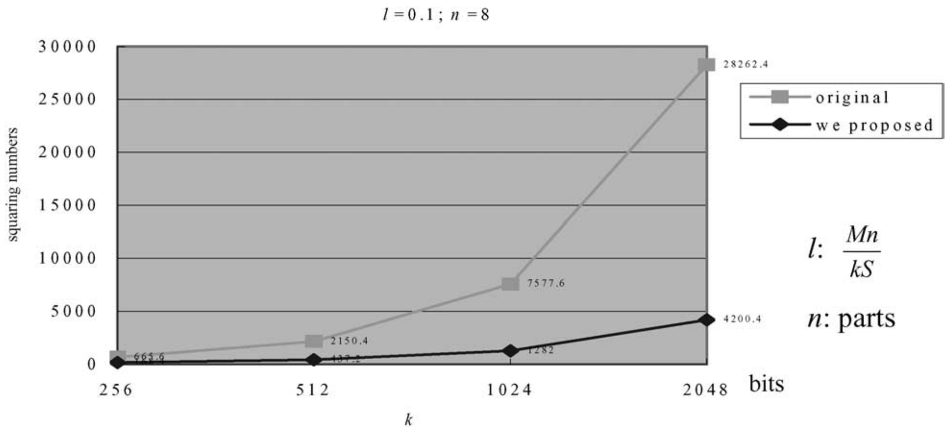

Fig. 8. The relationship between squaring numbers and k bits when $l = 0.1$, $n = 8$.

Procedure 8:

The MSB in the exponentiation evaluation of the third part is multiplied by the value $(1111)_2$, which is pre-stored in Procedure 2. $2^{15} * 15 = 491520$, i.e., this is the result of

$$(1111000000000000000)_2. \tag{17}$$

Procedure 9:

We add the results of $(11111)_2$ pre-stored in Procedure 2, Eqs. 15, 16, and 17 to get the answer: $(1111111111111111111)_2 = (524287)_{10}$.

8. Conclusions

Generally RSA is a very well tool for electronically signed business contracts, electronic checks, electronic purchase orders, and other electronic communications that must be authenticated. In computation of the modular exponentiation $M^E \bmod N$ is a fundamental and important arithmetic operation in many scientific investigations, especially in the area of cryptography. In this paper, we discuss the cost of exponentiation evaluation by means of multiplication numbers and squaring numbers. The technique we proposed is always better than the binary method. We can decrease the numbers (including the multiplication numbers and the squaring numbers) to speed up the exponentiation evaluation. The speedup ratio can be 1.06 even 2.75.

Most importantly, the computers are much cheaper today than before. We can therefore easily integrate several PCs utilizing parallel softwares like PVMPI, PVM, and MPI to accomplish this job. Take a look at the popular WWW (World Wide Webs), we can utilize computers to integrate huge computation power. If we adopt the parallel-processing and hardware-designing techniques properly, we can therefore further reduce the computational complexity of exponentiation evaluation effectively. Our proposed technique can not only implement the computation of exponentiation to accelerate the speed of RSA cryptosystem, but also much faster than Savas–Tenca–Koc algorithm (Savas *et al.*, 2000) in time complexity. No matter is in the domain of cryptology and the other domains such as satellite image processing, weather prediction, medicine development, gene exploration, and many mysterious questions, there is still more improvement for us to be expected.

References

Blum, T., and C. Paar (2001). High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers*, **50**(7), 759–764.

Chang, C.-C., and Y.-P. Lai (2005). A fast modular square computing method based on the generalized Chinese remainder theorem for prime moduli. *Applied Mathematics and Computation*, **161**(1), 181–194.

Chen, T.-S. (2005). A threshold signature scheme based on the elliptic curve cryptosystem. *Applied Mathematics and Computation*, **162**(3), 1119–1134.

Diffie, W., and M.-E. Hellman (1976). New directions in cryptography. *IEEE Transaction Information Theory*, **22**(6), 644–654.

Gueron, S., and O. Zuk (2005). Applications of the Montgomery exponent. *International Conference on Information Technology*: *Coding and Computing*, **1**, 620–625.

Hwang, R.-J., F.-F. Su, Y.-S.i Yeh and C.-Y. Chen (2005). An efficient decryption method for RSA cryptosystem. In *19th International Conference on Advanced Information Networking and Applications*, vol. 1. pp. 585–590.

Karatza, H. (2004). An excellent resource for parallel computing. *Distributed Systems Online*, *IEEE*, **5**(7), 1–4.

Knuth, D.-E. (1998). *The Art of Computer Programming*, *Fundamental Algorithms*, vol. 1. Addison-Wesley, 3rd edition.

Koc, C.-K. (1994). High-speed RSA implementation. RSA Laboratories, 2nd edition, November.

L, M.-M., and N. Nedjah (2005). Reconfigurable hardware for addition chains based modular exponentiation. *International Conference on Information Technology*: *Coding and Computing*, **1**, 603–607.

Lam, K.-Y., I. Shparlinski, H. Wang and C. Xing (2001). *Cryptography and Computational Number Theory*. Series: Progress in Computer Science and Applied Logic (PCS), vol. 20, 7th Edition.

Lastovetsky, A., and R. Reddy (2004). On performance analysis of heterogeneous parallel algorithms. *Parallel Computing*, **30**(11), 1195–1216.

Liu, B.-Q., D.-H. Chen, C.-G. Xiong and K. Xing (2005). New methods for binary multiplication. *International Journal of Computer Mathematics*, **82**(1), 13–22.

Lou, D.-C., and C.-C. Chang (1996). Fast exponentiation method obtained by folding the exponent in half. *IEE Electronics Letters*, **32**(11), 984–985.

Lou, D.-C., and C.-L. Wu (2004). Parallel modular exponentiation using signed-digit-folding technique. *Informatica*, **28**(2), 197–205.

Lou, D.-C., C.-L. Wu and C.-Y. Chen (2003). Fast exponentiation by folding the signed-digit exponent in half. *International Journal of Computer Mathematics*, **80**(10), 1251–1259.

`http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa160.html`

Mao, W. (2004). *Modern Cryptography: Theory and Practice*, Prentice Hall PTR, Upper Saddle River, NJ.

McIvor, C., M. McLoone and J.-V. McCanny (2004). Modified Montgomery modular multiplication and RSA exponentiation techniques. *IEE Proceedings of Computers and Digital Techniques*, **151**(6), 402–408.

Menezes, A., P.-V. Oorschot and S. Vanstone (1996). *Handbook of Applied Cryptography*, CRC Press, 3rd edition.

Rivest, R.-L., A. Shamir and L. Adleman (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communication of the ACM*, **21**(2), 120–126.

Rosen, K.-H. (2000). *Elementary Number Theory and Its Application*. Addison-Wesley, 4th edition.

Samoa, K.S., O. Semay and T. Takagi (2006). Analysis of fractional window recoding methods and their application to elliptic curve cryptosystems. *IEEE Transactions on Computers*, **55**(1), 48–57.

Savas, E., A.F. Tenca and C.K. Koc (2000). A scalable and unified multiplier architecture for finite field GF(p) and GF(2^m). In *Cryptographic Hardware and Embedded Systems. Workshop on Cryptographic Hardware and Embedded Systems*. Springer-Verlag, Berlin. pp. 277–292.

Seiffert, U. (2004). Artificial neural networks on massively parallel computer hardware. *Neurocomputing*, **57**, 135–150.

Shand, M., and J. Vuillemin (1993). Fast implementations of RSA cryptography. In *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*. pp. 252–259.

Stalling, W. (2003). *Cryptography and Network Security-Principles and Practices*, Prentice-Hall, 3rd edition.

Wu, C.-H., J.-H. Hong, and C.-W. Wu (2001). RSA cryptosystem design based on the Chinese remainder theorem. In *IEEE Proceedings of the Asia and South Pacific*. pp. 391–395.

Yasuyuki, S., and S. Kouichi (2006). Simple power analysis on fast modular reduction with generalized Mersenne prime for elliptic curve cryptosystems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **E89-A**(1), 231–237.

C.-L. Wu was born in Kaouhsiung, Taiwan, Republic of China (R.O.C.), on Dec. 8th, 1965. He received his BS degree in electrical engineering from the Chung Cheng Institute of Technology (CCIT), National Defense University, Taiwan, R.O.C., in 1988, and the MS degree in computer science from the United States Air Force Institute of Technology, Dayton, Ohio, in 1995. He received his PhD degree in the Department of Electrical Engineering at the Chung Cheng Institute of Technology, National Defense University, Taiwan, R.O.C., in 2004. He is currently an assistant professor and the director of the Department of Avionics Communication & Electronics Engineering, Chinese Air Force Institute of Technology (AFIT), Taiwan. He is also a member of Computer Security Committee of Crisis Management Society of Taiwan, R.O.C. He has been selected and included both in the 6th (Vol. VI, No. 83) edition of Asia/Pacific Who's Who in the World which has been published in 2005 and in the 1st edition of Afro-Asian Who's Who in the World which has been published in 2006. He has also been selected and included in Distinguished & Admirable Achievers 2005–2006, 2nd edition. His research interests include information security, cryptography, number theory, information system, algorithm design, complexity analysis, cryptography, computer arithmetic, and parallel computing.

D.-C. Lou was born in Chiayi, Taiwan, Republic of China (R.O.C.), on Mar. 18th, 1961. He received the BS degree from Chung Cheng Institute of Technology (CCIT), National Defense University, Taiwan, R.O.C., in 1987, and the MS degree from National Sun Yat-Sen University, Taiwan, R.O.C., in 1991, both in electrical engineering. He received the PhD degree in 1997 from the Department of Computer Science and Information Engineering at National Chung Cheng University, Taiwan, R.O.C. Since 1987, he has been with the Department of Electrical Engineering at CCIT, where he is currently a professor and the director of Computer Center of CCIT. His research interests include cryptography, steganography, algorithm design and analysis, computer arithmetic, parallel and distributed system. Prof. Lou is currently an area editor for security technology of Elsevier Science's Journal of Systems and Software. He is an honorary member of the Phi Tau Phi Scholastic Honor Society. He is a member of the IEICE Society and the Chinese Cryptology and Information Security Association. He is the owner of the 11th AceR Dragon PhD Dissertation Award. He has been selected and included in the 15th and 18th edition of Who's Who in the World which has been published in 1998 and 2001, respectively.

J.-C. Lai was born in Taichung, Taiwan, Republic of China (R.O.C.), on Dec. 20th, 1960. He received his MS degree in electrical engineering from the Chung Cheng Institute of Technology (CCIT), National Defense University, Taiwan, R.O.C., in 1999. He is currently pursuing his PhD degree in the Department of Electrical Engineering at CCIT, National Defense University, Taiwan, R.O.C. Since Aug. 2002, he has been on the faculty of the Chienkuo Technology University, where he is now a Lecturer in the Information Management Department. His research interests include information security, cryptography, computer arithmetic, algorithm design, and parallel computing.

T.-J. Chang was born in Taichung, Taiwan, Republic of China (R.O.C.), on Dec. 18th, 1970. He received his BS and the MS degree in electrical engineering from the Chung Cheng Institute of Technology (CCIT), Taiwan, Republic of China (R.O.C.) in 1993 and 2001, respectively. He is currently pursuing his PhD degree in the Department of Electrical Engineering at CCIT, National Defense University, Taiwan, R.O.C. His research interests include information system, information security, cryptography, complexity analysis, public-key cryptography, computer arithmetic, algorithm design, and parallel computing.

Greitas lygiagretus laipsnių skaičiavimo algoritmas RSA viešo rakto kriptosistemoms

Chia-Long WU, Der-Chyuan LOU, Jui-Chang LAI, Te-Jen CHANG

Mes žinome, kad informacijos saugumo poreikis šiomis dienomis plinta, ypač techninės įrangos realizacijoms, kaip protingų kortelių mikroschemos arba kodavimo greitintuvai. RSA kriptosistema yra viena iš plačiausiai naudojamų technologijų informacijos saugumui. Pagrindinė RSA kriptosistemos kodavimo ir dekodavimo užduotis yra skaičiuoti modulinius laipsnius. Šiame straipsnyje efektyvus lygiagretus modulinių laipsnių skaičiavimo būdas yra pasiūlytas ir parodyta, kad pasiūlytas algoritmas yra greitesnis negu Savas–Tenca–Koc algoritmas.