INFORMATICA, 2005, Vol. 16, No. 3, 449–468

An Efficient Montgomery Exponentiation Algorithm for Cryptographic Applications

Chia-Long WU
Department of Aviation & Communication Electronics, Chinese Air Force Institute of Technology
Kaohsiung 820, Taiwan
e-mail: chialongwu@seed.net.tw

Der-Chyuan LOU, 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: April 2005

Abstract. Efficient computation of the modular exponentiations is very important and useful for public-key cryptosystems. In this paper, an efficient parallel binary exponentiation algorithm is proposed which based on the Montgomery multiplication algorithm, the signed-digit-folding (SDF) and common-multiplicand-multiplicand (CMM) techniques. By using the CMM technique of computing the common part from two modular multiplications, the same common part in two modular multiplications can be computed once rather twice, we can thus improve the efficiency of the binary exponentiation algorithm by decreasing the number of modular multiplications. By dividing the bit pattern of the minimal-signed-digit recoding exponent into three equal length parts and using the technique of recording the common parts in the folded substrings, the proposed SDF-CMM algorithm can improve the efficiency of the binary algorithm, thus can further decrease the computational complexity of modular exponentiation. Furthermore, by using the proposed parallel SDF-CMM Montgomery binary exponentiation algorithm, on average the total number of single-precision multiplications can be reduced by about 61.3% and 74.1% as compared with Chang-Kuo-Lin's CMM modular exponentiation algorithm and Ha-Moon's CMM Montgomery modular exponentiation algorithm, respectively.

Key words: Montgomery reduction algorithm, common-multiplicand-multiplication, signed-digit recoding, modular exponentiation, public-key cryptosystems.

1. Introduction

Modular exponentiation and modular multiplication are the cornerstone computations performed in public-key cryptography. Taking the RSA cryptosystem (Rivest *et al.*, 1978), for example, both the encryption and decryption operations are accomplished by modular exponentiation. The encryption and decryption operations are accomplished by modular exponentiation and can be described as follows. Given M (plain text), E (public

key), D (private key), and N (modulus), compute ciphertext $C = M^E \bmod N$ for encryption and $M = C^D \bmod N$ for decryption. Moreover, these operations are realized by multiple modular multiplications based on the value of the exponents E and D, where $D \times E \bmod \psi(N) = 1$ and $\psi(N)$ is an Euler's totient function (Koren, 2002).

As efficient computation of the modular exponentiations is important for RSA cryptosystem, we need novel algorithms such as the Montgomery modular multiplication method (Montgomery, 1985; Su *et al.*, 1999; Tenca and Koc, 2003), addition chains method (Kunihiro and Yamamoto, 2000), binary method (Knuth, 1997), residue number conversion method (Premkumar, 2002; Nozaki *et al.*, 2003), signed-digit recoding method (Joye and Yen, 2000; Wei *et al.*, 2002), exponent-folding method (Lou and Chang, 1996), common-multiplicand-multiplication method (Yen and Laih, 1993a), and key-size partitioning method (Lee *et al.*, 2002). Detailed surveys and analyses of fast exponentiation techniques are given in (Gordon, 1998; Nedjah and Mourelle, 2002).

The rest of the paper is organized as follows. In Section 2, we first review some related works of modular exponentiation that attempt to minimize the number of multiplication. The proposed parallel SDF-CMM Montgomery algorithm and its flow chart for fast modular exponentiation are depicted in Section 3. The computational complexity of the proposed parallel modular exponentiation algorithm is detailed analyzed in Section 4. Finally, we conclude our work in Section 5.

2. The Cryptographic Arithmetic

The modular exponentiation is composed of repetition of modular multiplications. Therefore, modular exponentiation can be time consuming, and is often the dominant part of modern cryptographic algorithms for key exchange (Nedjah and Mourelle, 2002), electronic signatures (Diffie and Hellmen, 1976), and authentication (Rivest *et al.*, 1978). Two different approaches are often used to reduce the execution time of the modular exponentiation operation. One approach is simply to reduce the number of modular multiplication. The other approach is to reduce the execution time of each modular multiplication. In this paper, we are concentrate on the first approach to effectively reduce the number of modular multiplication.

2.1. *The Binary Exponentiation Method*

The binary exponentiation method is also known as the "square-and-multiply" method (Knuth, 1997). The basic idea of binary method is to compute M^E using the binary expression of exponent E. The exponentiation operation is broken into a series of squaring and multiplication operations by the use of the binary method. Assume k denotes the bit-length of the exponent E, the exponent E can be expressed in binary representation as $E = (e_{k-1}e_{k-2}\ldots e_1e_0)_2$ and $E = \sum_{i=0}^{k-1} e_i \times 2^i$, where $e_i \in \{0, 1\}$.

The binary method is important for speeding up exponentiation calculation. There are two commonly used algorithms in binary method can convert the exponentiation of $C = M^E \bmod N$ for RSA cryptosystem into a series of multiplications, i.e., the LSB

(least significant bit) binary algorithm and the MSB (most significant bit) binary exponentiation algorithm. The LSB binary exponentiation algorithm computes the exponentiation starting from the least significant bit position of the exponent E and proceeding to the left, which is depicted as follows.

LSB Binary Exponentiation Algorithm
Input: Exponent: $E = (e_{k-1}e_{k-2}\dots e_1e_0)_2$; Message: M
Output: Ciphertext: $C = M^E \bmod N$
$C = 1$; $S = M$;
begin
  for $i = 0$ ***to*** $k-1$ ***do*** /*scan from right to left*/
    begin
      if $(e_i = 1)$ $C = C \times S \bmod N$; /*multiply*/
      $S = S \times S \bmod N$; /*square*/
    end;
end.

Different from the LSB binary exponentiation algorithm, the MSB binary exponentiation algorithm computes the exponentiation starting from the most significant bit position of the exponent E and proceeding to the right, which is depicted as follows.

MSB Binary Exponentiation Algorithm
Input: Exponent: $E = (e_{k-1}e_{k-2}\dots e_1e_0)_2$; Message: M
Output: Ciphertext: $M^E \bmod N$
$C = 1$;
begin
  for $i = k-1$ ***downto*** 0 ***do*** /*scan from left to right*/
    begin
      $C = C \times C \bmod N$; /*square*/
      if $(e_i = 1)$ $C = C \times M \bmod N$; /*multiply*/
    end;
end.

We should note that as the LSB and MSB binary exponentiation algorithms have the same computations for both multiplication and squaring operations, therefore they share the same computational complexity. The computational complexities for both LSB and MSB binary exponentiation algorithms are $2 \times (k/2) + 1 \times (k/2) = 1.5k$ multiplications to evaluate $M^E \bmod N$, where k is the bit-length of the exponent E.

2.2. *The Common-Multiplicand-Multiplication Exponentiation Method*

Yen and Laih (1993) developed the common-multiplicand-multiplication (CMM) exponentiation algorithm to enhance the exponentiation performance of the LSB binary algorithm. The basic idea of the CMM technique is to extract the common parts of multiplicands, and then save the number of binary additions for the computation of common

parts. The variables Y_{comm} (recording the common parts) and $Y_{i,c}$ (recording the different parts) required in the CMM exponentiation algorithm are defined as follows (where **AND** and **XOR** are the bitwise logical operators):

$$Y_{comm} = \mathbf{AND}_{i=1}^{t} Y_i, \tag{1}$$

$$Y_{i,c} = Y_i \ \mathbf{XOR}\ Y_{comm}, \quad \text{for } i = 1, 2, \ldots, t. \tag{2}$$

Hence, Y_i can be represented as

$$Y_i = Y_{i,c} + Y_{comm}. \tag{3}$$

Therefore, the common-multiplicand multiplications $X \times Y_i$ $(i = 1, 2, \ldots, t)$ can be computed with the assistance of $X \times Y_{comm}$ as

$$X \times Y_i = X \times Y_{i,c} + X \times Y_{comm} \quad \text{for } i = 1, 2, \ldots, t. \tag{4}$$

By using the CMM method, the computations of $\{X \times Y_1, X \times Y_2\}$ can be represented as $\{X \times Y_{1,c} + X \times Y_{comm}, X \times Y_{2,c} + X \times Y_{comm}\}$. Let both X and Y_is be k-bit integers, on average the Hamming weights of Y_i, Y_{comm} and $Y_{i,c}$ are $k/2, k/2^t$ and $(k/2 - k/2^t)$, respectively. The total number of binary addition for the common-multiplicand-multiplication of $\{X \times Y_i \mid i = 1, 2, \ldots, t;\ t \geqslant 2\}$ is $k/2^t + t \times (k/2 - k/2^t)$. Hence, the performance improvement of the CMM algorithm can be denoted as

$$\frac{(k \times t)/2}{k/2^t + t \times (k/2 - k/2^t)} = \frac{1}{2/(t \times 2^t) + (1 - 1/2^{t-1})} = \frac{t}{(1-t) \times 2^{1-t} + t}. \tag{5}$$

Based on (Yen and Laih, 1993b), the optimal performance of Yen-Laih's CMM algorithm can be obtained as $\frac{4}{3}$ when $t = 2$ which implies we need 1.5 multiplications by using the CMM algorithm for evaluating $X \times Y_1$ and $X \times Y_2$. Moreover, by applying the CMM algorithm and the LSB binary exponentiation algorithm, the exponentiation can be computed by using $\frac{(1.5+1)}{2} k = 1.25k$ multiplications for exponent E being a k-bit integer.

2.3. *The Montgomery Modular Reduction Algorithm*

Modular multiplication is normally considered to be a complicated arithmetic operation because of the inherent multiplication and division operations. Montgomery (1985) introduced the modular reduction algorithm for multiplying two integers (called N-residues) modulo N while avoiding division by N. This algorithm reverses the order of processing the digits of the multiplicand using the least significant bits of the intermediate result to perform an addition rather than a subtraction.

The Montgomery reduction algorithm speeds up the modular multiplications and squarings required for exponentiation. Suppose that we want to compute $A \times B \bmod N$, where A, B and N are n-digit integers represented in base 2. Hence,

$$A = \sum_{i=0}^{n-1} A_i \times 2^i, \quad B = \sum_{i=0}^{n-1} B_i \times 2^i, \quad N = \sum_{i=0}^{n-1} N_i \times 2^i, \tag{6}$$

where A_i, B_i and N_i are elements of $\{0, 1\}$ for all i.

Dusse and Kaliski (1990) proposed a modified Montgomery reduction (*REDC*) algorithm to perform both multiplication and modular reduction simultaneously. This efficient Montgomery *REDC* algorithm is processed in N-residue and allows the precomputation of $N'_0 = -N_0^{-1} \bmod 2$ instead of $N' = -N^{-1} \bmod 2^n$. Assume we denote X as $A \times B \bmod N$, where A, B, X and N are n-digit integer, the Montgomery modular *REDC* algorithm can be depicted as follows.

Montgomery Modular Reduction (*REDC*) Algorithm

Input: A, B, N /*A, B and N are n-digit integers in base 2*/
Output: X /*$X = REDC(AB)$*/
$X = 0$; /*$X = (X_n X_{n-1} \ldots X_1 X_0)_2$*/
begin
  for $i = 0$ ***to*** $n - 1$ ***do*** /*scan from integer A right to left*/
    begin
      $X = X + A_i \times B$;
      $k = X_0 \times N'_0 \bmod 2$; /*$N'_0 = -N_0^{-1} \bmod 2$ and $N' = -N^{-1} \bmod 2^n$*/
      $X = (X + k \times N) \times 2^{-1}$;
    end;
  if $(X \geqslant N)$ $X = X - N$; /*$X = A \times B \times 2^{-n} \bmod N$*/
end.

From the Montgomery modular *REDC* algorithm depicted above, note that both $(2^{-n} \bmod N)$ and $(N^{-1} \bmod N)$ can be precomputed using the Euclidean algorithm (Knuth, 1997). Moreover, the Montgomery *REDC* algorithm allows the precomputation of $N = N_{n-1} \times 2^{n-1} + N_{n-2} \times 2^{n-2} + \ldots + N_1 \times 2 + N_0$ and $X = X_{n-1} \times 2^{n-1} + X_{n-2} \times 2^{n-2} + \ldots + X_1 \times 2 + X_0$, we can compute X one digit X_i in every modular reduction step instead of computing the whole X at one time.

As modular multiplication using the Montgomery modular *REDC* algorithm requires the transformation of both multiplier and multiplicand into the N-residue, therefore modular multiplication using this *REDC* algorithm requires a longer processing time than other methods. If $(2^{-n} \bmod N)$ has been precomputed and stored before we using the Montgomery reduction algorithm,A' and B' in the N-residue can be easily computed from *REDC*$(A \times (2^{-n} \bmod N))$ and *REDC*$(B \times (2^{-n} \bmod N))$ respectively as follows.

$$A' = \textit{REDC}(A \times (2^{-n} \bmod N)) = A \times (2^n)^{-1} \bmod N, \tag{7}$$

$$B' = \textit{REDC}(B \times (2^{-n} \bmod N)) = B \times (2^n)^{-1} \bmod N, \tag{8}$$

$$X = A' \times B' = A \times (2^n)^{-1} \times B \times (2^n)^{-1} \bmod N, \tag{9}$$

$$C' = \textit{REDC}(X) = A \times B \times 2^{-n} \bmod N, \tag{10}$$

$$C = \textit{REDC}(C') = A \times B \bmod N. \tag{11}$$

Notice that, Eqs. 7 to 11 shown above describe modular multiplication using the Montgomery *REDC* algorithm require a longer processing time than other methods due to the residue transformation of the multiplier and the multiplicand. For example, the modular multiplication using Eq. 7 to 11 requires $7n^2 + 4n$ multiplications since every reduction operation takes $n^2 + n$ multiplications. The classical Montgomery reduction algorithm only requires $2n^2 + n$ multiplications and m divisions (Dusse and Kaliski, 1990).

3. The Proposed Algorithm

In the following, we will first introduce the basic concept of minimal-signed-digit recoding arithmetic, and then we will summarize some important mathematical preliminaries including formulas for the common-multiplicand-multiplication (CMM) and signed-digit-folding (SDF) technique. Finally, we will give detailed description for the proposed parallel SDF-CMM Montgomery binary algorithm for fast exponentiation.

3.1. *Signed-Digit Recoding Arithmetic*

A signed-digit (SD) representation of an integer n in radix r is a sequence of digits $a = (a_k, \ldots, a_2, a_1, a_0)_{\mathrm{SD}_r}$ with $a_i \in \{0, \pm 1, \ldots, \pm(r-1)\}$ and r is the radix number for $k \geqslant i \geqslant 0$ such that $a = \sum_{i=0}^{k} a_i \times r^i$. The signed-digit (redundant) representations number system was first proposed by Avizienis (Avizienis, 1961) to make it possible to perform carry-free addition. Recently, redundant representations of this form have been used successfully in various arithmetic applications and many signed-digit number systems have been used to increase the efficiency of computer arithmetic (DeBrunner *et al.*, 2002; Syuto *et al.*, 2002).

Arno and Wheeler (1993) proposed the signed-digit representations for minimal hamming weight arithmetic. Assume we refer to S_r as the set of all signed digit radix-r representations of elements of Z, the mapping π: $S_r \rightarrow$Z defined by

$$\pi(a) = \sum_{i=0}^{\infty} a_i r^i \tag{12}$$

associates an integer with each element $a \in S_r$.

Let $a, b \in S_r$, we define negative digit $\overline{x}$ by $x - \mathrm{sgn}\,(x) \times r$, where $\mathrm{sgn}\,(x)$ is 0, 1, -1 depending on whether x is zero, positive, or negative, respectively. If

$a = (a_k, \ldots, a_2, a_1, a_0)$ and $b = (b_k, \ldots, b_2, b_1, b_0)$, then we define the addition of $c = a + b = (c_k, \ldots, c_2, c_1, c_0)$ in S_r where $\varepsilon_{-1} = 0$ as

$$\left.\begin{array}{l} \varepsilon_i = 0 \\ c_i = a_i + b_i + \varepsilon_{i-1} \end{array}\right\} \quad \text{if } -r < a_i + b_i + \varepsilon_{i-1} < r,$$
$$\left.\begin{array}{l} \varepsilon_i = \operatorname{sgn}(a_i + b_i + \varepsilon_{i-1}) \\ c_i = a_i + b_i + \varepsilon_{i-1} - \varepsilon_i \times r \end{array}\right\} \text{ otherwise.} \tag{13}$$

We denote the generic notation of the packet of data $(a_t, a_{t+1}, \varepsilon_t)$ as (x, y, ε) and produce the output digit $\hat{y}$ and the new carry $\hat{\varepsilon}$, and the new data packet becomes $(a_{t+1}, a_{t+2}, \varepsilon_{t+1})$. Therefore, the output digit $\hat{y}$ and the carry $\hat{\varepsilon}$ are generated as

$$\left.\begin{array}{l} \hat{\varepsilon} = 0 \\ \hat{y} = a_i + b_i + \varepsilon \end{array}\right\} \quad \text{if } (a_i + b_i + \varepsilon)\big(x + \operatorname{sgn}(a_i + b_i + \varepsilon)\big) \neq 0,$$
$$\left.\begin{array}{l} \hat{\varepsilon} = \operatorname{sgn}(a_i + b_i + \varepsilon) \\ \hat{y} = a_i + b_i + \varepsilon - \hat{\varepsilon} \times r \end{array}\right\} \text{ otherwise.} \tag{14}$$

The problem of finding minimal binary representations is usually referred to as "canonical Booth recoding" (DeBrunner *et al.*, 2002). The signed-digit recoding is canonical if its signed-digit representation contains no adjacent nonzero digits. The canonical signed-digit recoding is unique if its binary representation is viewed as padded with an initial zero. Based on (Arno and Wheeler, 1993), the MSD recoding algorithm for generating the signed-digit representation of minimal-Hamming-weight is depicted as follows.

Minimal-Signed-Digit (MSD) Recoding Algorithm
Input: $a \in S_r$ with $\pi(a) = n$; /*a is redundant representation of n*/
Output: $A(a) \in S_r$; /*$A(a)$ denotes the action of this algorithm on a*/
  $t = 0$;
  while $(\ldots, a_{t+2}, a_{t+1}, a_t) \neq (\ldots, 0, 0, 0)$ ***do***
  begin
    if $a_t \neq 0$ ***then***
    begin
      $b = (\ldots, \operatorname{sgn}(a_t), -\operatorname{sgn}(a_t) \times r, 0, \ldots, 0)$ /*nonzeros at t and $t+1$*/
      $c = a + b$;
      if $c_{t+1} = 0$ $a = c$;
    end;
  $t = t + 1$;
  end.

If we take signed-digit recoding system with radix-2 $(r = 2)$ for example, three symbols $\{\overline{1}, 0, 1\}$ are allowed for the digit set, in which 1 and $\overline{1}$ in digit position k represent $+2^k$ and -2^k, respectively. Based on MSD recoding arithmetic algorithm show above, the signed-digit arithmetic representation can be depicted as follows.

Notice that, in order to obtain signed-digit $\overline{1}$ for our signed-digit representation in Fig. 1, the subtraction operation executed between $3r$ and r is a "no-borrow (carry) sub-

$$
\begin{aligned}
2r &= (r_{k-1}, r_{k-2}, r_{k-3}, \ldots, r_1, r_0, 0)_2 \\
+\ r &= (r_{k-1}, r_{k-2}, \ldots, r_2, r_1, r_0)_2 \\
\hline
3r &= (s_k, s_{k-1}, s_{k-2}, s_{k-3}, \ldots, s_1, s_0, r_0)_2 \\
\text{“no-borrow subtraction”} \rightarrow -\ r &= (r_{k-1}, r_{k-2}, \ldots, r_2, r_1, r_0)_2 \\
\hline
2r &= (e_k, e_{k-1}, e_{k-2}, e_{k-3}, \ldots, e_1, e_0, 0)_{\mathrm{SD2}}
\end{aligned}
$$

Fig. 1. Signed-digit arithmetic representation.

traction”. On average, the probability of the digit “0” appearance is “2/3”, and the total occurrence probability of nonzero digits “1” and “$\overline{1}$” is “1/3” (Arno and Wheeler, 1993).

3.2. *Mathematical Preliminaries*

The basic idea of our proposed SDF-CMM Montgomery binary exponentiation algorithm is try to extract the common substring of the signed-digit recoding exponent E, and then save the number of required for the computation of common substring. Let the exponent E have the radix-2 representation $(e_{k-1}e_{k-2}\ldots e_1e_0)_2$, i.e., $E = \sum_{i=0}^{k-1} e_i \times 2^i$, where $e_i \in \{0, 1, \overline{1}\}$ and k is the bit-length of the signed-digit recoding exponent E_{MSD}.

In the first phase of exponent-folding method, by folding the signed-digit recoding exponent E_{MSD} in half n times, and E_{MSD} is then divided into2^n equal sized substrings. Let each substring of signed-digit recoding E_{MSD} be denoted as E_i for $i = 1, 2, \ldots, 2^n$, i.e., $E_{MSD} = E_{2^n} \parallel E_{2^n-1} \parallel E_{2^n-2} \parallel \ldots \parallel E_2 \parallel E_1$, where $\parallel$ is the concatenation operator. Hence

$$M^E = \prod_{i=1}^{2^n} S^{(i-1)(\frac{k}{2^n})}(M^{E_i}), \tag{15}$$

where $S^{(m)}(z)$ represents performing m squares on the related value z, and E_i is denoted as $(e_i^{\frac{k}{2^n}-1} e_i^{\frac{k}{2^n}-2} \ldots e_i^1 e_i^0)_2$.

In the second phase of exponent-folding method, we define the following:

$$E_{comm\_i} = E_{comm\_(i+1)} = E_i \ \textbf{AND}\ E_{i+1} \ \text{ for } \ i = 1, 3, \ldots, 2^n - 3, 2^n - 1, \tag{16}$$

$$E_{excl\_i} = E_{comm\_i} \ \textbf{XOR}\ E_i \ \text{ for } \ i = 1, 2, \ldots, 2^n, \tag{17}$$

where **AND** and **XOR** are the bitwise logical operators. Then, E_i can be denoted as

$$E_i = E_{comm\_i} + E_{excl\_i}. \tag{18}$$

In the third phase of exponent-folding method, the exponentiation of the consecutive pairs of $M^{E_{2^n}}, M^{E_{2^n-1}}, \ldots, M^{E_1}$ can be computed as follows:

$$M^{E_i} = M^{E_{comm\_i}} \times M^{E_{excl\_i}}, \tag{19}$$

and

$$M^{E_{i+1}} = M^{E_{comm\_i}} \times M^{E_{excl\_(i+1)}} \text{ for } i = 1, 3, \ldots, 2^n - 3, 2^n - 1. \tag{20}$$

The E_{comm_i} operation will record the common part for every two consecutive segments E_i and E_{i+1}. Note, we record the differences for "E_i and E_{comm_i}" and "E_{i+1} and E_{comm_i}" in E_{excl_i} and $E_{excl\_(i+1)}$, respectively. We define the logical operations **AND** and **XOR** operators over the set $\{0, 1, \overline{1}\}$ for the SDF-CMM Montgomery exponentiation algorithm in Table 1.

By using signed-digit-folding technique, the CMM binary exponentiation algorithm (depicted in Section 2.2) can be generalized as follows.

Let

$$Y^*_{comm} = \mathbf{AND}_{i=1}^{2^n} Y_i, \tag{21}$$

and

$$Y_{excl\_i} = Y_{comm\_i} \mathbf{XOR} Y^*_{comm} \text{ for } i = 1, 2, \ldots, 2^n. \tag{22}$$

Thus, each Y_{common_i} can be represented as

$$Y_{comm\_i} = Y^*_{comm} + Y_{excl\_i}. \tag{23}$$

Suppose that Y^*_{comm} is folded n-times, and every part of Y^*_{comm} is denoted as $Y_{comm\_i} (i = 1, 2, \ldots, 2^n)$, where $Y_{comm} = Y_{comm\_1} || \ldots || Y_{comm\_2^n - 1} || Y_{comm\_2^n}$. Therefore, the generalized common-multiplicand multiplications algorithm using the signed-digit-folding technique can compute $X \times Y_i (i = 1, 2, \ldots, 2^n)$ with the assistance of $X \times Y_{common}$ as

$$X \times Y_{comm} = \sum_{i=1}^{2^n} S^{(m/2^n)(2^n - i)} (X \times Y_{excl\_i} + X \times Y^*_{comm}), \tag{24}$$

where $S^{(m)}(z)$ function represents performing m squares (or left shift of m bits) on the related value z.

For example, for Y_{comm} is folded exact one-time (i.e., $n = 1$), we can obtain the following relation from the Eq. 4 defined previously in Section 2.2:

$$X \times Y_{comm} = \sum_{i=1}^{2} S^{(m/2)(2-i)} (X \times Y_{excl\_i} + X \times Y^*_{comm})$$

Table 1

Bitwise logical operators **AND** and **XOR**

AND	$\overline{1}$	0	1
$\overline{1}$	$\overline{1}$	0	0
0	0	0	0
1	0	0	1

XOR	$\overline{1}$	0	1
$\overline{1}$	0	$\overline{1}$	0
0	$\overline{1}$	0	1
1	0	1	0

$$= S^{(m/2)(2-1)}(X \times Y_{excl\_1} + X \times Y^*_{comm}) + S^{(m/2)(2-2)}(X \times Y_{excl\_2} + X \times Y^*_{comm}). \quad (25)$$

Hence, the computations $\{X \times Y_1, X \times Y_2\}$ can be identically represented as

$$\Big\{X \times Y_{1,c} + X \times Y_{excl\_2} + X \times Y^*_{comm} + S^{(m/2)}(X \times Y_{excl\_1} + X \times Y^*_{comm}), \\ X \times Y_{2,c} + X \times Y_{excl\_2} + X \times Y^*_{comm} + S^{(m/2)}(X \times Y_{excl\_1} + X \times Y^*_{comm})\Big\}. \quad (26)$$

Here we respectively introduce three lemmas to give the MSD representation with minimal-weight, to efficiently calculate the modular inverse result for handling the negative signed-digits, and to better describe the expected value of signed-digit recoding number based on the probability distribution property as follows.

Lemma 1. *Let S be a string and the sequence $[S]^l$ denote $S, S, \ldots, S$ (repeat l times), in order to find the canonical representation of minimal Hamming weight for exponent E_{MSD}, the following two equivalences exist in our radix-2 signed-digit representation:*
(1) $([0,1]^l, 1)_{\mathrm{SD}_2} = (1, [0,\overline{1}]^l)_{\mathrm{SD}_2}$,
(2) $([0,\overline{1}]^l, \overline{1})_{\mathrm{SD}_2} = (\overline{1}, [0,1]^l)_{\mathrm{SD}_2}$.

Proof. (1) Based on the minimal-Hamming-weight signed-digit recoding arithmetic representation defined in Section 3.1, we can have the following result:

$$\big([0,1]^l, 1\big)_{\mathrm{SD}_2} = 1 + \sum_{i=1}^{l} 2^{2i-1} = 2^{2l} - \sum_{i=1}^{l} 2^{2i-2}.$$

However, $2^{2l} - \sum_{i=1}^{l} 2^{2i-2}$ can also be represented as $(1, [0,\overline{1}]^l)_{\mathrm{SD}_2}$.
Therefore, the first equivalence $([0,1]^l, 1)_{\mathrm{SD}_2} = (1, [0,\overline{1}]^l)_{\mathrm{SD}_2}$ holds.
(2) Again, based on the minimal-Hamming-weight signed-digit recoding arithmetic representation defined in Section 3.1, we know

$$\big([0,\overline{1}]^l, \overline{1}\big)_{\mathrm{SD}_2} = -1 - \sum_{i=1}^{l} 2^{2i-1} = \sum_{i=1}^{l} 2^{2i-2} - 2^{2l}.$$

However, $\sum_{i=1}^{l} 2^{2i-2} - 2^{2l}$ can also be represented as $(\overline{1}, [0,1]^l)_{\mathrm{SD}_2}$.
Therefore, the second equivalence $([0,\overline{1}]^l, \overline{1})_{\mathrm{SD}_2} = (\overline{1}, [0,1]^l)_{\mathrm{SD}_2}$ holds.

By using Lemma 1 shown above, we can find the canonical representation for radix-2 signed-digit representation of minimal Hamming weight.

Lemma 2. *The modular arithmetic relation "$a^{-1} = r^{-1} \bmod N$" holds as "$r = a \bmod N$".*

Proof. As from the extended Euclid algorithm (Knuth, 1997), yields

$$a^{-1} \bmod N = x \Leftrightarrow ax \bmod N = 1 \Leftrightarrow a \times x - -N \times y = 1.$$

Assume $r = a \bmod N$, hence, $a = Q \times N + r$.
As $a \times a^{-1} = 1 \bmod N$, we have $r \times a^{-1} = 1 \bmod N$.

$$\because r \times r^{-1} = 1 \bmod N,$$
$$\therefore r \times a^{-1} = r \times r^{-1} \bmod N,$$

hence, $a^{-1} = x = r^{-1} \bmod N$.

By using Lemma 2, we know as long as the relation $a = Q \times N + r$ holds, we can have the modular equivalence relation of $a^{-1} = r^{-1} \bmod N$.

Lemma 3. *Let k_r be a random variable on the space of m-digit radix-r integers denoting the radix-r signed-digit representation of minimal Hamming weight. As $E_{xp}(a)$ denote the expected value of a, we can have the expected value of k_r as*

$$E_{xp}(k_r) = \frac{(r-1)}{r+1} m \quad (\textit{where } \ m \to \infty).$$

Proof. Let n be a random m-digit radix-r integer with $0 \leqslant n < r^m$, whose standard representation is given by $a = (\ldots, 0, a_m, a_{m-1}, a_{m-2}, \ldots, a_1, a_0)$, and whose minimal-weight representation is given by $\hat{a} = (\ldots, 0, \hat{a}_m, \hat{a}_{m-1}, \hat{a}_{m-2}, \ldots, \hat{a}_1, \hat{a}_0)$.

Let $P_{rob}(a)$ denote the probability distribution of a. Assume we have the input to signed-digit recoding algorithm is the standard radix-r representation of n. Let w_m^+ and w_m^- denote the number of positive digits and the number of negative digits of $\hat{a}$, respectively.

Based on (Arno and Wheeler, 1993), we can have the following

$$P_{rob}(w_m^+) = \frac{(r-1)^2}{r(r+1)} \text{ and } P_{rob}(w_m^-) = \frac{(r-1)}{r(r+1)} \quad (\text{where } \ m \to \infty). \tag{27}$$

Therefore, we can have the following expected values as

$$E_{xp}(w_m^+) = \frac{(r-1)^2}{r(r+1)} m, \quad E_{xp}(w_m^-) = \frac{(r-1)}{r(r+1)} m \quad (\text{where } \ m \to \infty). \tag{28}$$

We should note that k_r is independent of the radix-r signed-digit representation as defined in (Arno and Wheeler, 1993), thus we have $k_r = w_m^+ + w_m^-$, and we can therefore have the following equivalence:

$$E_{xp}(w_m) = E_{xp}(w_m^+) + E_{xp}(w_m^-). \tag{29}$$

Therefore, we have

$$E_{xp}(k_r) = \frac{(r-1)^2}{r(r+1)}m + \frac{(r-1)}{r(r+1)}m = \frac{(r-1)}{(r+1)}m \quad (\text{where } m \to \infty). \tag{30}$$

By using Lemma 3, we can better describe the expected value of minimal-signed-digit recoding number based on the probability distribution property.

3.3. *The Proposed SDF-CMM Montgomery Binary Exponentiation Algorithm*

After we have introduced the definitions and lemmas shown above, we here propose a fast parallel Montgomery modular exponentiation algorithm using Montgomery modular reduction algorithm, SDF technique, and CMM method as follows.

SDF-CMM Montgomery Binary Exponentiation Algorithm
Input: M, E_{MSD}, N, R /*M, R and N are n-digit integers in base 2*/
Output: $C = M^{E_{\text{MSD}}} \bmod N$ /*$E_{MSD} = (e_m e_{m-1} \dots e_1 e_0)$, where $e_i \in \{0, 1, \overline{1}\}$*/
begin
$S = M \times R, C = R \bmod N$; /*$R = 2^n$*/
$C_1 = C_1 = C_1 = 1$;
for $b = 1$ ***to*** $k/2^n$ ***do*** /*scan the MSD recoding exponent from right to left*/
begin
if $(e^b_{comm\_i} = 1)$ $C_1 =$ *REDC*(SC_1); /*handle the common part*/
if $(e^b_{comm\_i} = \overline{1})$ $C_1 =$ *REDC*$(S^{-1}C_1)$; /*handle the common part*/
if $(e^b_{excl\_i} = 1)$ $C_2 =$ *REDC*(SC_2); /*handle the positive digit*/
if $(e^b_{excl\_i} = \overline{1})$ $C_2 =$ *REDC*$(S^{-1}C_2)$; /*handle the negative digit*/
if $(e^b_{excl\_(i+1)} = 1)$ $C_3 =$ *REDC*(SC_3); /*handle the positive digit*/
if $(e^b_{excl\_(i+1)} = \overline{1})$ ***then*** $C_3 =$ *REDC*$(S^{-1}C_3)$; /*handle the negative digit*/
$S =$ *REDC*(SS);
end;
$C_2 =$ *REDC*(C_1C_2); /*$C_2 = M^{E_{\text{MSD}[1]}} \bmod N, 0 \leqslant C < N$*/
$C_3 =$ *REDC*(C_1C_3); /*$C_3 = M^{E_{\text{MSD}[\overline{1}]}} \bmod N, 0 \leqslant C < N$*/
$C =$ *REDC*(C_2C_3); /*$C_3 = M^{E_{\text{MSD}[\overline{1}]}} \bmod N, 0 \leqslant C < N$*/
end.

Note, as the modular multiplication operation and the modular inversion operation in this improved exponentiation algorithm can be concurrently executed, we can further have the proposed SDF-CMM algorithm work more efficient for evaluating M^E using parallel computing technique.

3.4. *Parallel Processing of The Proposed Algorithm*

As we can execute the modular multiplication (for positive digits of folding-exponent) and the multiplicative inverse (for negative digits of folding-exponent) operations separately, we can apply the "parallel-processing" technique on the improved SDF-CMM algorithm to speed up the total exponentiation efficiency as follows.

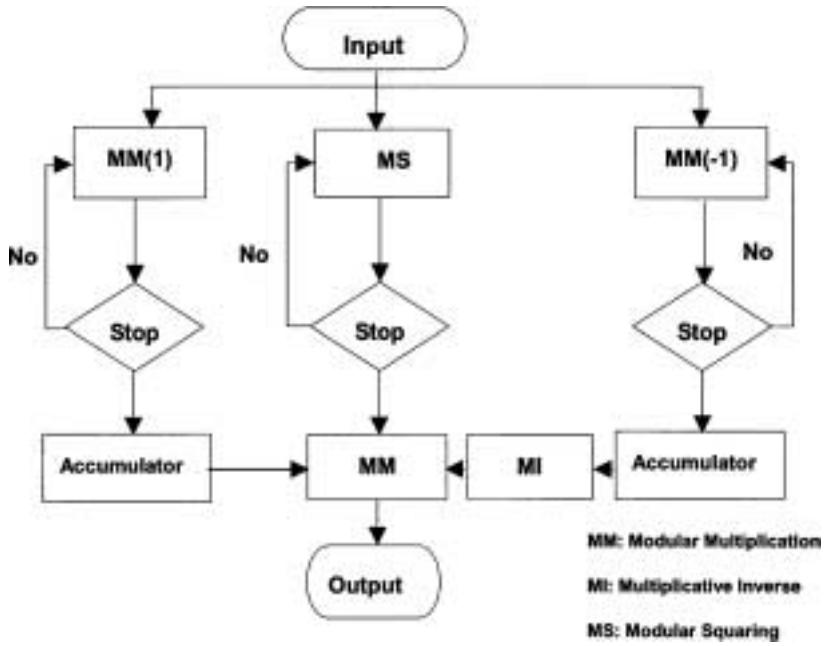

Fig. 2. Flow chart of the parallel SDF-CMM algorithm.

The parallel computing flow chart of the SDF-CMM binary exponentiation algorithm is depicted in Fig. 2.

By adopting lemma 1 shown in Section 3.2, we can therefore transform the original multiplicative inverse operation into normal positive (multiplication) calculation, and use the operation result for our original multiplicative inverse calculation. Here we can speed the whole modular exponentiation process as we let the multiplicative inverse operation performed only once in the last step before our final output result for the improved SDF-CMM Montgomery algorithm.

Here we put an example to explain how the SDF-CMM Montgomery binary exponentiation algorithm works.

EXAMPLE. Let $N = 37 \times 41 = 1517$.

By folding the signed-digit exponent one time, we can get the optimal computational complexity (Lou and Chang, 1996), i.e., $n = 1$.

Message $M = 127$,

$$E = 1213 = (10010111101)_2 = (10100\overline{2}00\overline{1}01)_{\mathrm{SD}}.$$

By using the concatenation operation, yields

$$E = E_1 \parallel E_2 = (10100\overline{2})_{\mathrm{SD}} \parallel (000\overline{1}01)_{\mathrm{SD}}.$$

For positive folding-exponent,

$$C_1 \equiv M^{(101000)_{\mathrm{SD}}} (\mathrm{mod} N) \equiv M^{(40)_{10}} (\mathrm{mod} N);$$
$$C_2 \equiv M^{(00001)_{\mathrm{SD}}} (\mathrm{mod} N) \equiv M^{(1)_{10}} (\mathrm{mod} N).$$

For negative folding-exponent,

$$D_1 \equiv M^{(0000\bar{2})_{\mathrm{SD}}} (\mathrm{mod} N) \equiv M^{(2)_{10}} (\mathrm{mod} N);$$
$$D_2 = M^{(00\bar{1}00)_{\mathrm{SD}}} (\mathrm{mod} N) = M^{(4)_{10}} (\mathrm{mod} N).$$

Similarly, we can also have

$$\begin{aligned} E &= (10100000001)_{SD} + (0\bar{2}000\bar{1}00)_{\mathrm{SD}} \\ &= (10100000001)_2 - (1000100)_2 = A - B. \end{aligned}$$

Hence, $A = (10100000001)_2 = 1281$, $B = (1000100)_2 = 68$.

Using the proposed SDF method shown below, we can obtain $M^E \bmod N = 127^{1213} \bmod 1517 = 1048$.

$$r_{M^A} \equiv C_1^{2^{(k-\frac{k}{2^n})}} \times C_2 (\mathrm{mod} N),$$
$$\Rightarrow r_{M^A} \equiv (127^{40})^{2^5} \times 127^1 (\mathrm{mod} N) \equiv 127^{1281} (\mathrm{mod} 1517).$$
$$r_{M^B} \equiv D_1^{2^{(n-\frac{k}{2^n})}} \times D_2 (\mathrm{mod} N),$$
$$\Rightarrow r_{M^B} \equiv (127^2)^{2^5} \times 127^4 (\mathrm{mod} N) \equiv 127^{68} (\mathrm{mod} 1517).$$
$$\therefore M^A \bmod N \equiv r_{M^A} = 127^{1281} \bmod 1517 = 988,$$
$$\therefore M^B \bmod N \equiv r_{M^B} = 127^{68} \bmod 1517 = 551.$$

Based on the calculation shown above, we can have:

$$(M^B)^{-1} \bmod N \equiv r_{M^B}^{-1} \bmod N = 551^{-1} \bmod 1517 = 1082.$$

Finally,

$$\because M^E \bmod N \equiv M^{A-B} \bmod N,$$
$$(M^A \bmod N)(M^{-B} \bmod N) \bmod N \equiv r_{M^A} \times (r_{M^B})^{-1} \bmod N.$$

Therefore, we obtain $M^E \bmod N = 988 \times 1082 \bmod 1517 = 1048$. This example shown that we can obtain the correct result using the proposed SDF-CMM Montgomery method.

We should note that, by using the computation over Galois Field, the time complexity of multiplicative inverse operation is equivalence to the bit shift operation (Lee *et al.*, 2002). To further speed up the SDF-CMM algorithm, we can put the multiplicative

inverse operation in SDF-CMM algorithm over a finite filed (such as GF(2^n) (Wu and Chang, 1995; Yen, 1997). The detailed computational complexity analyses will be given in Section 4 as follows.

4. Complexity Analyses

In this section, we will detailed describe the theoretical analyses for the performance of the proposed parallel SDF-CMM algorithm in public key cryptography. We use the number of modular multiplications to express the speed-up efficiency. By using the proposed SDF-CMM algorithm to evaluate the modular exponentiation computation of $M^E \pmod N$ where E is the k-bit minimal-signed-digit exponent.

Based on the lemma 2 given in Section 3.2, the probability for the occurrence of positive digit "r" and negative digit "$\overline{r}$" (or "$-r$") is $\frac{(r-1)^2}{r(r+1)}$ and $\frac{(r-1)}{r(r+1)}$, respectively, and the probability for the occurrence of digit "0" is $\frac{2}{(r+1)}$. Assume that the two possible nonzero digits, "1" and "$\overline{1}$", occur with equal probability (Nozaki *et al.*, 2003), then we can have the occurrence probability for each element recorded as $\{\Pr(0) = 2/3, \Pr(1) = \Pr(\overline{1}) = 1/6\}$.

Let the three m-bit exponent segments (decompositions) in the parallel SDF-CMM algorithm be denoted as E_i, E_{i+1} and E_{i+2}, based on the bit level arithmetic defined in Section 3.2, we can compute E_{common} using "E_i **AND** E_{i+1} **AND** E_{i+2}" where for $i = 1, 2, \ldots, 2^n$ and n is the exponent-folding times. Hence, E_{common} will be equal to 1 (or $\overline{1}$) if and only if for E_i, E_{i+1} and E_{i+2} are all equal to 1.

Based on Eq. 22, on average, we have the following occurrence probabilities:

$$P_{rob}(E_{common} = 1) = P_{rob}(E_{common} = \overline{1}) = \left(\frac{1}{6}\right)^3.$$

Based on Eqs. 23, 24, and 25, on average, we have:

$$P_{rob}(E_{i,i+1}=1)=P_{rob}(E_{i,i+2}=1)=P_{rob}(E_{i+1,i+2}=1)=(P_{rob}(1))^3=\left(\frac{1}{6}\right)^3.$$

Based on Eqs. 26, 27, and 28, on average, we have:

$$\begin{aligned} &P_{rob}(E_{i,common} = 1) = P_{rob}(E_{i+1,common} = 1) \\ &\quad = P_{rob}(E_{i+2,common} = 1) = (P_{rob}(1))^3 = \left(\frac{1}{6}\right)^3. \end{aligned}$$

In the parallel SDF-CMM binary exponentiation algorithm, the exponentiations $M^{E_{common}}$, $M^{E_{i,i+1}}$, $M^{E_{i,i+2}}$, $M^{E_{i+1,i+2}}$, $M^{E_{i,common}}$, $M^{E_{i+1,common}}$, $M^{E_{i+2,common}}$ for processing positive signed-digit bits and negative signed-digit bits are performing in parallel. We can check that the parallel SDF-CMM exponentiation algorithm has its optimal performance when the minimal-signed-digit recoding exponent E is folded exact one time $(n = 1)$.

On average the probability of performing one, two, three, or four multiplications in each iteration are $\frac{1}{6^3}, \frac{3}{6^3}, \frac{3}{6^3}, \text{and} \frac{1}{6^3}$, respectively. The SDF-CMM binary algorithm therefore takes $\left[\frac{1}{6^3} + \frac{2\times3}{6^3} + \frac{3\times3}{6^3} + \frac{4}{6^3}\right] \times m + 9$ multiplications to obtain all exponentiation results of $M^{E_i}, M^{E_{i+1}}, M^{E_{i+2}}$. Given these three partial results, when applying Eq. 21, the parallel SDF-CMM algorithm takes $2m + 2$ additional multiplications to perform the squaring operation in each bit position and get the final result of M^E.

The entire computation in the parallel SDF-CMM algorithm needs on average $\left[\frac{1}{6^3} + \frac{2\times3}{6^3} + \frac{3\times3}{6^3} + \frac{4}{6^3} + 2\right] \times m + 11 = \left(\frac{452}{216}\right)m \approx 2.0926m + 11 \approx 0.6975k + 11$ multiplications (where k is the bit-length of exponent and $m = k/3$). The proposed parallel SDF-CMM exponentiation algorithm outperforms both the Wu-Chang's exponentiation algorithm in (Wu and Chang, 1995) which needs totally $1.219k$ multiplications plus some extra additions and shift operations and the Yen's exponentiation algorithm in (Yen, 1997) which needs totally $1.292k + 11$ multiplications.

Furthermore, by using the proposed parallel SDF-CMM Montgomery binary exponentiation algorithm, on average the total number of single-precision multiplications can be reduced by about 61.3% and 74.1% as compared with Chang-Kuo-Lin's CMM modular exponentiation algorithm (Ha and Moon, 1998) and Ha-Moon's CMM Montgomery modular exponentiation algorithm (Chang *et al.*, 2003), respectively.

5. Conclusions

As we know the modular exponentiation is on of the most important operations in public-key cryptography. In this paper, an efficient parallel modular exponentiation algorithm is proposed which based on both the common-multiplicand-multiplicand (CMM) and signed-digit-folding (SDF) techniques. By dividing the bit pattern of the minimal-signed-digit recoding exponent E into three equal length parts and using the technique of recording the common parts in the folded substrings, the proposed SDF-CMM Montgomery algorithm can improve the efficiency of both the binary algorithm and common-multiplicand-multiplication algorithm, thus can further decrease the computational complexity of modular exponentiation.

By adopting the parallel processing technique for signed-digit recoding exponent E, we can further enhance our SDF-CMM Montgomery algorithm and obtain the optimal overall computational complexity as $0.6975k+11$ multiplications by folding the minimal-signed-digit radix-2 recoding exponent E exactly one time where k denotes the digit-length of the exponent. We can check that the SDF-CMM Montgomery algorithm outperforms other existing exponentiation methods. Moreover, as the multiplicative inverse operation in GF(2^n) finite field can be done by a simple shift operation (Takagi *et al.*, 2001; Watanabe *et al.*, 2002), in the proposed SDF-CMM Montgomery algorithm, we can further decrease the overall computational complexity.

References

Arno, S., and F.S. Wheeler (1993). Signed digit representations of minimal hamming weight. *IEEE Transactions on Computers*, **42**(8), 1007–1010.

Avizienis, A. (1961). Signed digit number representation for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, **EC-10**(3), 389–400.

Chang, C.-C., Y.-T. Kuo and C.-H. Lin (2003). Fast algorithms for common-multiplicand multiplication and exponentiation by performing complements. In *Proceedings of the 17th IEEE Symposium on Advanced Information Networking and Applications*. pp. 807–811.

DeBrunner, L.S., V.E. DeBrunner and D. Bhogaraju (2002). Defining canonical-signed-digit number systems as arithmetic codes. In *Proceedings of the 36th Asilomar Conference on Signals, Systems and Computers*, vol. 2. pp. 1593–1597.

Diffie, W., and E. Hellmen (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, **22**(6), 644–654.

Dusse, S.R., and B.S. Kaliski (1990). A cryptographic library for the Motorola DSP 56000. In *Proceedings of EUROCRYPT Workshop on Theory and Application of Cryptographic Techniques, Advances in Cryptology, LNCS* 473. Springer-Verlag. pp. 230–244.

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, **31**(4), 469–472.

Joye, M., and S.-M. Yen (2000). Optimal left-to-right binary signed-digit recoding. *IEEE Transactions on Computers*, **49**(7), 740–748.

Ha, J.-C., and S.-J. Moon (1998). A common-multiplicand method to the Montgomery algorithm for speeding up exponentiation. *Information Processing Letters*, **66**(2), 105–107.

Gordon, D.M. (1998). A survey of fast exponentiation methods. *Journal of Algorithms*, **27**(1), 129–146.

Knuth, D.E. (1997). *The Art of Computer Programming*, Vol. II: *Seminumerical Algorithms*, 3rd edition. MA: Addison-Wesley.

Koren, I. (2002). *Computer Arithmetic Algorithms*, 2nd edition. A.K. Peters, Natick, MA.

Kunihiro, N., and H. Yamamoto (2000). New methods for generating short addition chains. *IEICE Transactions on Fundamentals*, **E83-A**(1), 60–67.

Lee, S.-Y., Y.-J. Jeong and O.-J. Kwon (2002). A faster modular multiplication based on key size partitioning for RSA public-key cryptosystem. *IEICE Transactions on Information and Systems*, **E85-D**(4), 789–791.

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

Montgomery, P.L. (1985). Modular multiplication without trial division. *Mathematics of Computation*, **44**(170), 519–521.

Nedjah, N., and L.M. Mourelle (2002). Efficient parallel modular exponentiation algorithm. In *Proceedings of the 2nd Advances in Information Systems* (*ADVIS*) *International Conference*. pp. 405–414.

Nozaki, H., A. Shimbo and S. Kawamura (2003). RNS Montgomery multiplication algorithm for duplicate processing of base transformations. *IEICE Transactions Fundamentals*, **E86-A**(1), 89–97.

Premkumar, A.B. (2002). A formal framework for conversion from binary to residue numbers. *IEEE Transactions on Circuits and Systems II*: *Analog and Digital Signal Processing*, **49**(2), 135–144.

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

Su, C.-Y., S.-A. Hwang, P.-S. Chen and C.-W. Wu (1999). An improved Montgomery's algorithm's for high-speed RSA public-key cryptosystem. *IEEE Transactions on Very Large Scale Integration* (*VLSI*) *Systems*, **7**(2), 280–284.

Syuto, M., E. Satake, K. Tanno and O. Ishizuka (2002). A high-speed binary to residue converter using a signed-digit number representation. *IEICE Transactions on Information and Systems*, **E85-D**(5), 903–905.

Takagi, N., J. Yoshiki and K. Takagi (2001). A fast algorithm for multiplicative inversion in GF(2^n) using normal basis. *IEEE Transactions on Computers*, **50**(5), 394–398.

Tenca, A.F., and C.K. Koc (2003). A scalable architecture for modular multiplication based on Montgomery's algorithm. *IEEE Transactions on Computers*, **52**(9), 1215–1221.

Watanabe, Y., N. Takagi and K. Takagi (2002). A VLSI algorithm for division in GF(2^m) based on extended binary GCD algorithm. *IEICE Transactions on Fundamentals*, **E85-A**(5), 994–999.

Wei, S., S. Chen and K. Shimizu (2002). Fast modular multiplication using booth recoding based on signed-

digit number arithmetic. In *Proceedings of the 2nd IEEE Asia-Pacific Conference on Circuits and Systems*. pp. 31–36.

Wu, T.-C., and Y.-S. Chang (1995). Improved generalization common-multiplicand multiplications algorithm of Yen and Laih. *Electronics Letters*, **31**(20), 1738–1739.

Yen, S.-M. (1997). Improved common-multiplicand multiplication and fast exponentiation by exponent decomposition. *IEICE Transactions on Fundamentals*, **E80-A**(6), 1160–1163.

Yen, S.-M., and C.-S. Laih (1993a). Common-multiplicand multiplication and its applications to public key cryptography. *IEE Electronics Letters*, **29**(17), 1583–1584.

Yen, S.-M., and C.-S. Laih (1993b). Common-multiplicand multiplication and its applications to public key cryptography. *Electronics Letters*, **29**(17), 1583–1584.

C.-L. Wu was born in Kaouhsiung, Taiwan, Republic of China, 1965. He received his BS degree in electrical engineering from the Chung Cheng Institute of Technology, Taiwan, Republic of China, 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 (Republic of China). He is currently an assistant professor at the Department of Avionics Communication & Electronics, 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. His research interests include information security, cryptography, number theory, information system, algorithm design, complexity analysis, cryptography, computer arithmetic and parallel and distributed computing.

D.-C. Lou was born in Chiayi, Taiwan, Republic of China, 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 eleventh AceR Dragon PhD Dissertation Award. He has been selected and included in the fifteenth and eighteenth edition of Who,s Who in the World which has been published in 1998 and 2001, respectively.

T.-J. Chang was born in Taichung, Taiwan, Republic of China, 1970. He received his BS degree in electrical engineering from the Chung Cheng Institute of Technology, Taiwan, Republic of China, in 1993, and the MS degree in electrical engineering from the Chung Cheng Institute of Technology, National Defense University, Taiwan, Republic of China, in 2001. He is currently studying his PhD degree in the Department of Electrical Engineering at the Chung Cheng Institute of Technology, National Defense University, Taiwan (Republic of China). His research interests include information security, number theory, cryptography, computer arithmetic, complexity analysis, and parallel computing.

Efektyvus Montgomerio kėlimo laipsniu algoritmas kodavimui

Chia-Long WU, Der-Chyuan LOU, Te-Jen CHANG

Efektyvus modulinis kėlimas laipsniu yra labai svarbus ir naudingas atviro rakto kodavimo sistemoms. Šiame straipsnyje yra pasiūlytas efektyvus lygiagretus dvinaris kėlimo laipsniu algoritmas, pagrįstas Montgomery daugybos algoritmu, skaitmenų su ženklu suvėrimo ir bendro-daugiklio-daugiklio technikas. Pasiūlytas algoritmas naudoja vidutiniškai 61,3% ir 74,1% mažiau viengubo tikslumo daugybų negu Chang-Kuo-Lin ir Ha-Moon bendro-daugiklio-daugiklio Montgomerio modulinio kėlimo laipsniu algoritmai atitinkamai.