

## ALGORITHM DESIGN AND THEORETICAL ANALYSIS OF A NOVEL CMM MODULAR EXPONENTIATION ALGORITHM FOR LARGE INTEGERS

ABDALHOSSEIN REZAI<sup>1</sup> AND PARVIZ KESHAVARZI<sup>2</sup>

**Abstract.** Modular exponentiation is an important operation in public-key cryptography. The Common-Multiplicand-Multiplication (CMM) modular exponentiation is an efficient exponentiation algorithm. This paper presents a novel method for speeding up the CMM modular exponentiation algorithm based on a Modified Montgomery Modular Multiplication (M4) algorithm. The M4 algorithm uses a new multi bit scan-multi bit shift technique by employing a modified encoding algorithm. In the M4 algorithm, three operations (the zero chain multiplication, the required additions and the nonzero digit multiplication) are relaxed to a multi bit shift and one binary addition in only one clock cycle. Our computational complexity analysis shows that the average number of required multiplication steps (clock cycles) is considerably reduced in comparison with other CMM modular exponentiation algorithms.

**Mathematics Subject Classification.** 68P25, 94A60, 14G50, 11Yxx, 11Y16, 11Rxx.

---

*Keywords and phrases.* Modular multiplication, canonical recoding, modular exponentiation, public-key cryptosystem, high speed arithmetic.

<sup>1</sup> Academic Center for Education, Culture and Research (ACECR), Isfahan University of Technology (IUT) branch, Isfahan, Iran. [rezaie@acecr.ac.ir](mailto:rezaie@acecr.ac.ir)

<sup>2</sup> Electrical and Computer Engineering Faculty, Semnan University, Semnan, Iran. [pkeshavarzi@semnan.ac.ir](mailto:pkeshavarzi@semnan.ac.ir)

## 1. INTRODUCTION

The modular exponentiation plays a major role in many Public-Key Cryptosystems (PKCs) [9, 15, 20, 23, 24, 32]. Since the modular exponentiation consists of a series of modular multiplications; the performance of this operation is determined by the efficiency of the modular multiplication and the number of required modular multiplications [13, 17, 22].

There are many methods developed to speed up the performance of the modular multiplication such as Montgomery modular multiplication [12], systolic modular multiplication [7, 28, 33], high-radix modular multiplication [2, 22, 25, 27, 29], parallel calculation of the quotient and the partial result [10], scalable modular multiplication [25–27], bipartite technique [34] and Booth recoding technique [3, 6, 16, 26].

To reduce the number of required modular multiplications in the modular exponentiation, there are other methods such as using sliding window method [13, 14], signed-digit recoding technique [1, 5, 30, 31] and Common-Multiplicand-Multiplication (CMM) method [8, 11, 30–32].

Among them, Ha and Moon [8] proposed that the common part of the modular multiplications in the modular exponentiation can be computed once rather than twice and named it CMM method. Wu *et al.* [30] utilized the signed-digit recoded exponent in the CMM method to reduce the probability of nonzero digits. Moreover, Wu [31] divided the Minimal Signed-Digit (MSD) exponent into three equal lengths in the CMM method. He named it CMM-MSD modular exponentiation algorithm. This algorithm effectively enhances the efficiency of the modular exponentiation algorithm. It should be noted that, these algorithms utilized the Montgomery modular multiplication algorithm to perform the modular multiplication.

In this paper, a novel technique is applied to the CMM-MSD modular exponentiation algorithm [31]. This novel technique is based on a Modified Montgomery Modular Multiplication (M4) algorithm. The proposed CMM-MSD modular exponentiation algorithm has the following distinctive characteristics: (1) utilized a modified encoding algorithm for multiplier. The encoded multiplier has the sparse property and minimal Hamming weight. (2) Process each zero chain and its nonzero digit of the encoded multiplier in only one multiplication step (clock cycle). This property, which implies the multi bit scan-multi bit shift technique, reduces the number of required multiplication steps considerably. (3) Process the partial multiplication in each multiplication step in the binary method instead of the high-radix method. (4) Complexity analysis shows that the proposed CMM-MSD modular exponentiation algorithm has advantages in comparison with other CMM modular exponentiation algorithms.

The rest of this paper is organized as follows: Section 2 describes the Canonical Recoding (CR) technique, the Montgomery modular multiplication algorithm, and the CMM method for the Montgomery modular exponentiation algorithm. The proposed CMM-MSD modular exponentiation is presented in Section 3. Section 4

presents the computational complexity analysis of the proposed modular exponentiation algorithm. Finally, conclusion is given in Section 5.

## 2. PRELIMINARIES

### 2.1. THE CANONICAL RECODING TECHNIQUE

A signed-digit representation [3] of an integer  $X = (x_m, x_{m-1}, \dots, x_1, x_0)_{SD}$  is a sequence of digits such that  $X = \sum_{i=0}^m x_i 2^i$ , where  $x_i \in \{-1, 0, 1\}$ . Reitwinsner [21] presented a canonical recoding technique to convert an integer from the binary representation to the signed-digit representation. This recoding technique, which is also called Non-Adjacent Form (NAF), guarantees the minimal Hamming weight [1, 18, 19]. Algorithm 1 is used for converting an  $m$ -bit integer  $X$  from the binary representation to its canonical representation.

---

**Algorithm 1.** The canonical recoding (CR) algorithm.

---

**Input:**  $X = (x_{m-1}x_{m-2} \dots x_1x_0)_2$

**Output:**  $D = (d_m d_{m-1} \dots d_1 d_0)_{SD}$

1.  $c_0 = 0$ ;
  2. **For**  $i = 0$  **To**  $m - 1$
  3.  $c_{i+1} = \lfloor (k_i + k_{i+1} + c_i)/2 \rfloor$ ;
  4.  $d_i = k_i + c_i - 2c_{i+1}$ ;
  5. **End For**
  6. **Return**  $D$ ;
- 

This algorithm scans input integer  $X$  from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). The average Hamming weight of an  $m$ -bit canonical recoded integer is about  $\frac{m}{3}$  [5, 19].

### 2.2. THE MONTGOMERY MODULAR MULTIPLICATION ALGORITHM

Montgomery modular multiplication algorithm [12] is an efficient algorithm for the modular multiplication. This algorithm speeds up the modular multiplication and modular exponentiation algorithm by replacing the trial division by the modulus with a simple right shift [17, 22]. Algorithm 2 shows the radix-2 Montgomery modular multiplication algorithm.

The inputs of this algorithm are  $n$ -bit integers  $X$ ,  $Y$  and  $M$ . The output is  $S(n) = X.Y.R \bmod M$  where  $S(i)$  denotes  $S$  in the  $i$ th iteration,  $x_i$  denotes the  $i$ th bit of  $X$ , and  $R = 2^{-n}$ . In this algorithm, the output  $S(n)$  is computed in  $n$  clock cycles. So, it is a time-consuming operation [17, 22].

---

**Algorithm 2.** The radix-2 Montgomery modular multiplication algorithm.

---

**Input:**  $X, Y, M$ ;

**Output:**  $S(n) = X.Y.R \bmod M$ ;

1.  $S(0) = 0$ ;
  2. **For**  $i = 0$  **To**  $n - 1$
  3.      $q_i = (S(i) + x_i.Y) \bmod 2$ ;
  4.      $S(i + 1) = (S(i) + x_i.Y + q_i.M)/2$ ;
  5. **End For**
  6. If  $S(n) \geq M$  Then  $S(n) = S(n) - M$ ;
  7. **Return**  $S(n)$ ;
- 

### 2.3. THE CMM METHOD FOR THE MONTGOMERY MODULAR EXPONENTIATION ALGORITHM

The modular exponentiation algorithm is usually performed using the binary methods (square-and-multiply) and Montgomery modular multiplication algorithm [9, 17, 20, 22, 29]. There are two basic algorithms in the binary methods for computing the modular exponentiation algorithm: the Left-to-Right (L2R) modular exponentiation algorithm and the Right-to-Left (R2L) modular exponentiation algorithm [17, 22, 29]. The L2R modular exponentiation algorithm processes the exponent bits from the MSB to the LSB, while the R2L exponentiation algorithm scans the exponent bits from the LSB to the MSB. The R2L exponentiation algorithm using the Montgomery modular multiplication algorithm is shown in Algorithm 3 [8].

---

**Algorithm 3.** The R2L Montgomery modular exponentiation algorithm.

---

**Input:**  $A, E, R, N$ ;

**Output:**  $C = A^E \bmod N$ ;

1.  $S = AR \bmod N, C = R \bmod N$ ; /\*  $R = 2^{-n} *$  /
  2. **For**  $i = 0$  **To**  $k - 1$
  3.     If  $(e_i = 1)$  Then  $\{C = \text{Mont}(S, C), S = \text{Mont}(S, S)\}$ ;
  4.     Else  $S = \text{Mont}(S, S)$ ;
  5. **End For**
  6.  $C = \text{Mont}(C, 1)$ ;
  7. **Return**  $C$ ;
- 

In Algorithm 3,  $A$  and  $N$  are two  $n$ -bit integers in radix-2, and  $E$  is  $k$ -bit integer in radix-2. In this algorithm, when the exponent bit is not zero (*i.e.*  $e_i = 1$ ), both  $\text{Mont}(S, C)$  and  $\text{Mont}(S, S)$  are executed. Ha and Moon [8] proposed that the common part in  $\text{Mont}(S, C)$  and  $\text{Mont}(S, S)$  can be computed once rather than twice. There are several attempts [8, 30, 31] to speed up the performance of the modular exponentiation algorithm based on this idea. One of the recent attempts

is the CMM-MSD Montgomery modular exponentiation algorithm which is shown in Algorithm 4 based on the original format in [31].

---

**Algorithm 4.** The CMM-MSD Montgomery modular exponentiation algorithm.

---

**Input:**  $A, E_{MSD}, R, N$ ;  
**Output:**  $C_1 = A^{E_{Common[1]}}, C_2 = A^{E_{1,c[1]}}, C_3 = A^{E_{2,c[1]}}, C_4 = A^{E_{3,c[2]}}$   
 $D_1 = A^{E_{Common[\overline{1}]}}, D_2 = A^{E_{1,c[\overline{1}]}}, D_3 = A^{E_{2,c[\overline{1}]}}, D_4 = A^{E_{3,c[\overline{2}]}}$ ;

1.  $C_1 = C_2 = C_3 = C_4 = D_1 = D_2 = D_3 = D_4 = 2^n$ ;
2.  $S = AR \bmod N, C = R \bmod N$ ; /\*  $R = 2^{-n}$  \*/
3. **For**  $i = 0$  **To**  $k - 1$
4. If( $e_{c_i} = 1$ ) Then  $C_1 = \text{Mont}(S, C_1)$ ; /\*evaluate  $A^{E_{Common}}$  for positive signed-digit \*/;
5. If( $e_{c_i} = \overline{1}$ ) Then  $D_1 = \text{Mont}(S, D_1)$ ; /\*evaluate  $A^{E_{Common}}$  for negative signed-digit\*/;
6. If( $e_{1_i} = 1$ ) Then  $C_2 = \text{Mont}(S, C_2)$ ; /\*evaluate  $A^{E_{1,c}}$  for positive signed-digit\*/;
7. If( $e_{1_i} = \overline{1}$ ) Then  $D_2 = \text{Mont}(S, D_2)$ ; /\* evaluate  $A^{E_{1,c}}$  for negative signed-digit\*/;
8. If( $e_{2_i} = 1$ ) Then  $C_3 = \text{Mont}(S, C_3)$ ; /\*evaluate  $A^{E_{2,c}}$  for positive signed-digit\*/;
9. If( $e_{2_i} = \overline{1}$ ) Then  $D_3 = \text{Mont}(S, D_3)$ ; /\*evaluate  $A^{E_{2,c}}$  for negative signed-digit\*/;
10. If( $e_{3_i} = 1$ ) Then  $C_4 = \text{Mont}(S, C_4)$ ; /\*evaluate  $A^{E_{3,c}}$  for positive signed-digit\*/;
11. If( $e_{3_i} = \overline{2}$ ) Then  $D_4 = \text{Mont}(S, D_4)$ ; /\*evaluate  $A^{E_{3,c}}$  for negative signed-digit \*/;
12.  $S = \text{Mont}(S, S)$ ;
13. **End for**

---

In Algorithm 4, the exponent  $E_{MSD}$ , which is obtained using canonical recoding algorithm [21], is divided into three equal lengths as  $E_1, E_2$  and  $E_3$ . Moreover, the following variables are used in this algorithm:

$$E_{common} = E_1 \text{ AND } E_2 \text{ AND } E_3 \tag{2.1}$$

$$E_{i,c} = E_i \text{ XOR } E_{common} \tag{2.2}$$

where operators AND and XOR are depicted in Table 2.1 in [31] and  $1 \leq i \leq 3$ .

In this algorithm, the  $C_i$  and  $D_i$  registers,  $1 \leq i \leq 4$ , are utilized to save the intermediate results of the positive and negative  $e_{ji}$ , respectively. Moreover, the multiplicative inverse operation is replaced by multiplication operation based on Lemma 2 in [31]. The common part of several multiplications is also computed just once by using the CMM method. Using the post computation, the exponentiation operation  $A^E$  is depicted as (2.3).

$$A^E = A^{E_{MSD}} = A^{E_1 \cdot 2^n} \cdot A^{E_2 \cdot 2^{\frac{2n}{3}}} \cdot A^{E_3} \tag{2.3}$$

where

$$\begin{aligned} E_1 &= E_{1[1]} + E_{1[\overline{1}]}, \\ E_{1[1]} &= E_{1,c[1]} + E_{common[1]} \text{ and} \\ E_{1[\overline{1}]} &= E_{1,c[\overline{1}]} + E_{common[\overline{1}]} \end{aligned} \tag{2.4}$$

Although this algorithm is efficient, but it can be improved as described in the next sections.

### 3. THE PROPOSED MODULAR ALGORITHMS

The proposed CMM-MSD modular exponentiation algorithm is based on the new serial-parallel modular computation. In the serial-parallel multiplication, the partial result is shifted only one bit per iteration. Although the multiplication by zero bit results in zero, but this multiplication is performed and is implemented per iteration. In this paper, we propose that  $k$  time multiplication by zero bits in sequence in  $k$  multiplication steps, followed by the required additions are relaxed to one  $k + 1$  bits shift in only one multiplication step followed by only a binary addition operation as we will describe later. So, we require determining the number of zero bits in sequence to specify the number of shifts in each multiplication step.

This section utilized an encoding algorithm based on a Modified Canonical Recoding (MCR) technique, which shows the number of zero bits in sequence in each digit. In addition, the MCR representation is applied to the Montgomery modular multiplication algorithm, which results in a modified modular multiplication algorithm named here as the M4 algorithm. Then, map the results to derive the CMM-MSD modular exponentiation algorithm.

#### 3.1. THE PROPOSED ENCODING ALGORITHM

The MCR representation of an integer  $X$  is defined here as  $Z = (z_{r-1}z_{r-2}, z_{r-3}z_{r-4}, \dots, z_jz_{j-1}, \dots, z_3z_2, z_1z_0)$ . This representation includes a sequence of pairs, which are separated by comma. Each pair,  $z_jz_{j-1}$ , shows the nonzero digit and the number of zero bits in sequence consecutively. Figure 1 shows the block diagram for converting an integer from the binary representation to the MCR representation.

As it is shown in Figure 1, the MCR encoding algorithm first uses the canonical recoding technique, and then replaces  $k$  zero bits in sequence by the integer  $k$ . Each pair in the MCR representation contains two digits:  $k$  which shows the number of zero bits in sequence and another digit which is 1 or  $-1$ .

So, each pair of the MCR representation includes two numbers: the number of zero bits in sequence in canonical representation, which is shown here by underline numbers  $\underline{0}, \underline{1}, \underline{2}, \underline{3}, \dots$ , and a nonzero digit in the canonical recoding representation, 1 and  $-1$ .

For example, for  $X = (474)_{10} = (111011110)_2$  the CR representation is  $D = (1000\bar{1}000\bar{1}0)_{CR}$  and the MCR representation is  $Z = (\underline{1}\bar{3}\bar{3}\bar{1}\bar{1})_{MCR}$ , where  $\bar{1} = -1$ .

In this example, the MCR representation requires only 6 digits; however, the CR representation and the binary representation require 10 digits and 9 bits, respectively.

We propose Algorithm 5 for converting an  $n$ -bit integer  $X$  from the binary representation to the MCR representation.

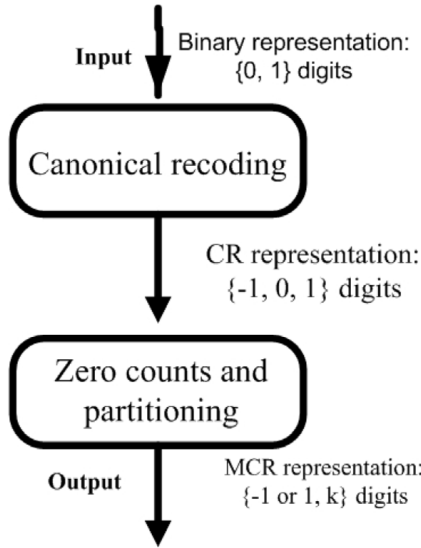


FIGURE 1. The block diagram of the MCR encoding algorithm.

---

**Algorithm 5.** The proposed encoding algorithm  $MCR(X)$ .

---

**Input:**  $X = (x_{n-1}x_{n-2} \dots x_1x_0)_2$  /\* $X$  is a positive integer where  $x_i \in \{0, 1\}$  \*/;

**Output:**  $Z = (z_{j-1}z_{j-2} \dots z_1z_0)_{MCR}$  /\*  $Z$  is MCR representation where  $z_i \in \{-1, 1, k\}$  \*/;

1.  $c_0 = 0, k = 1, j = x_0, d_{-1} = 1, Z = 0$ ;

2. **For**  $i = 0$  **To**  $n$

{*Canonical recoding phase*}

3.  $c_{i+1} = \lfloor (x_i + x_{i+1} + c_i) / 2 \rfloor$ ;

4.  $d_i = x_i + c_i - 2c_{i+1}$ ;

{*Zero counts and partitioning phase*}

5. **If**  $(d_i = 0)$  **Then**

6. **If**  $(d_{i-1} = 1)$  **Then**  $k = k + 1$ ; /\* Zero chain continue \*/

7. **Else**  $k = 1$ ;

8. **End if**

9. **Else**

10. **If**  $(d_{i-1} = 0)$  **Then**

11.  $z_j = k; j = j + 1$ ; /\* End of zero chains, assign zero count  $k$  to  $z_{j-1}$  and increment  $j$  \*/

\*

12. **End if**

13.  $z_j = d_i$ ; /\* Assign the nonzero digit to count  $k$  to  $z_j$  \*/

14.  $j = j + 1$ ;

15. **End if**

16. **End For**

17. **Return**  $Z$ ;

---

The output of Algorithm 5 is  $Z$ , which is the MCR representation of  $X$ . In the first stage of this encoding algorithm, Steps 3 and 4, the canonical recoding algorithm is applied to the input  $X$ . It results in a sparse form of digits with the minimal Hamming weight in the redundant signed-digit representation using only three digits  $\{\bar{1}, 0, 1\}$ . The nonzero digits divide the canonical representation into different size parts. Each part includes one nonzero digit and a chain of zero bits, except in the least significant part in which there may be no zero bit.

In the second stage of Algorithm 5, Steps 5 to 15, zero bits between nonzero digits in the canonical recoding representation are counted and put in  $k$  to produce  $z_{j-1}$  in the MCR representation of  $X$ . In Steps 5 and 6, zero chain continuation is checked. The end of zero chain is determined in Steps 9 to 15. Then, the zero count  $k$  is assigned to  $z_{j-1}$  and the nonzero digit in the CR is assigned to  $z_j$ .

### 3.2. THE PROPOSED MONTGOMERY MODULAR MULTIPLICATION ALGORITHM

In this section, we present a modified Montgomery modular multiplication algorithm by applying the MCR encoding algorithm to the multiplier and using the multi bit scan-multi bit shift technique. The M4 executes the high-radix multiplication,  $2^k$ -radix, by  $k$ -bit left shift, while only the binary multiplication is required for the hardware implementation.  $k$  is the number of zero bits in sequence and is a variable number. The M4 Algorithm is shown in Algorithm 6.

---

**Algorithm 6.** Modified Montgomery modular multiplication (M4) algorithm.

---

**Input:**  $Z = \text{MCR}(X), Y, M; /* X, Y$  and  $M$  are  $n$ -bit binary integers,  $Z$  is an MCR representation of  $X */$   
**Output:**  $S(J) = X.Y.R \text{ mod } M;$

1.  $S(0) = 0;$
2. **For**  $j = 1$  **To**  $J - 1$  **Step** 2
3.  $k = z_{j-1};$
4.  $p(j) = S_{k...0}(j - 1) + 2^k.z_j.Y_{k...0};$
5.  $q(j) = p(j)(2^{k+1}.M_{k...0}^{-1}) \text{ mod } 2^{k+1};$
6.  $s(j + 1) = (S(j - 1) + 2^k.z_j.Y + q(j).M)/2^{k+1};$
7. **EndFor**
8. **If**  $S(J) \geq M$  **Then Return**  $S(J) = S(J) - M;$
9. **Else Return**  $S(J);$

---

The inputs of the M4 algorithm are  $Z = \text{MCR}(X)$ ,  $Y$  and  $M$ . The output is  $S(J) = X.Y.R \text{ mod } M$ , which is the result of the Montgomery modular multiplication,  $p(j)$  denotes the value of  $p$  in  $j$ th iteration,  $S_{k...0} = S \text{ mod } 2^{k+1}$ ,  $Y_{k...0} = Y \text{ mod } 2^{k+1}$  and  $q^{(j)}$  denotes the  $j$ th digit of  $q$ . Since the multiplier  $Z$  is shown in the MCR representation, the M4 algorithm requires on average  $\frac{n}{3}$  clock cycles instead of  $n$  clock cycles in the binary Montgomery modular multiplication. In each iteration of the M4 algorithm, two digits of  $Z$  are scanned:  $z_{j-1}$  and  $z_j$



(two digits of each pair) which are the number of zero bits in sequence and the nonzero digit either 1 or  $-1$  in the MCR representation, respectively.

In Step 3,  $k$  is assigned by content of  $z_{j-1}$  which is the number of zero bits in sequence. This assignment implies the multi bit scan.  $k$  is also used for multi bit shift later in Step 6. To implement the main idea of the Montgomery modular multiplication algorithm (the  $k + 1$  least significant bits of the partial result  $S$  are all zero bits) a multiple of the modulus  $M$ , *i.e.*  $q^{(j)}.M$ , is added to the partial result. This step is needed to make sure that there are no significant digits lost in the right shift operation in Step 6.

In Step 6, the partial result  $S(j + 1)$  is computed first by adding  $2^k.z_j.Y$  and  $q^{(j)}.M$  to the previous partial result  $S(j - 1)$ , then this result is shifted by  $k + 1$  bits to the right (dividing by  $2^{k+1}$ ). Note that,  $k$  determines the number of shifts and is a variable number which is changed based on  $z_{j-1}$  in the MCR representation. As a result, the hardware implementation of the M4 algorithm needs a multi-bit shifter. In this case, we utilized a modified, limited number of shifts; Barrel shifter for implementing the multi-bit shift.

Since  $z_j$  is only 1 or  $-1$ , the multiplication of  $2^k.z_j.Y$  is implemented by using a binary multiplication and  $k$ -bit shift modified Barrel shifter. As a result,  $S(j - 1) + 2^k.z_j.Y$  is relaxed to a binary addition by  $2^k.Y$  or  $-2^k.Y$ . So, the implementation of  $2^k.z_j.Y$  and  $S(j - 1) + 2^k.z_j.Y$  in the M4 algorithm are simple.

It should be noted that, in the hardware implementation of the M4 algorithm, the maximum number of shifts in the Barrel shifter should have a limit. As a result, in the MCR representation, the number of zero bits in each pair should be limited. Our statistical analysis on the number of zero bits in sequence for large integers in the CR representation shows that the average number of zero bits in sequence greater than 6 is very small. Hence, we can use this limit on zero bits in sequence in each pair of the MCR representation without significant loss in the efficiency, but this increases the number of pairs in the MCR representation in case there are 6 zero bits in sequence followed zero instead of nonzero digit.

### 3.3. THE PROPOSED CMM-MSD MODULAR EXPONENTIATION ALGORITHM

The modular exponentiation is a series of modular multiplications. So, we proposed applying the M4 algorithm to the modular exponentiation to speed up the CMM-MSD Montgomery exponentiation algorithm [31] as shown in Algorithm 7.

Similar to Algorithm 4, the exponent  $E_{MSD}$  in Algorithm 7 is divided into three equal lengths as  $E_1$ ,  $E_2$  and  $E_3$ . The following variables are also utilized in this algorithm:

$$E_{\text{common}} = E_1 \quad \text{AND} \quad E_2 \quad \text{AND} \quad E_3 \tag{3.1}$$

$$E_{i,c} = E_i \quad \text{XOR} \quad E_{\text{common}} \tag{3.2}$$

where  $1 \leq i \leq 3$ .

In addition, the  $C_i$  and  $D_i$  registers,  $1 \leq i \leq 4$ , are used to save the intermediate results of the positive and negative  $e_{ji}$ , respectively.

---

**Algorithm 7.** The proposed CMM-MSD Montgomery modular exponentiation algorithm.

---

**Input:**  $A, E_{\text{MSD}}, R, N$ ;  
**Output:**  $C_1 = A^{E_{\text{Common}[1]}}, C_2 = A^{E_{1,c[1]}}, C_3 = A^{E_{2,c[1]}}, C_4 = A^{E_{3,c[2]}}$   
 $D_1 = A^{E_{\text{Common}[\bar{1}]}, D_2 = A^{E_{1,c[\bar{1}]}, D_3 = A^{E_{2,c[\bar{1}]}, D_4 = A^{E_{3,c[\bar{2}]}$ ;  
 1.  $C_1 = C_2 = C_3 = C_4 = D_1 = D_2 = D_3 = D_4 = 2^n$ ;  
 {Parallel begin}  
 2.  $S = AR \bmod N, C = R \bmod N$ ; /\*  $R = 2^{-n}$  \*/  
 3.  $T = \text{MCR}(S)$ , after performing  $a_0.R$ ;  
 {Parallel end}  
 4. **For**  $i = 0$  **To**  $m$   
 {Parallel begin}  
 5. If  $(e_{c_i} = 1)$  Then  $C_1 = M4(T, C_1)$ ; /\*evaluate  $A^{E_{\text{Common}}}$  for positive signed-digit\*/;  
 6. If  $(e_{c_i} = \bar{1})$  Then  $D_1 = M4(T, D_1)$ ; /\*evaluate  $A^{E_{\text{Common}}}$  for negative signed-digit\*/;  
 7. If  $(e_{1_i} = 1)$  Then  $C_2 = M4(T, C_2)$ ; /\* evaluate  $A^{E_{1,c}}$  for positive signed-digit \*/;  
 8. If  $(e_{1_i} = \bar{1})$  Then  $D_2 = M4(T, D_2)$ ; /\*evaluate  $A^{E_{1,c}}$  for negative signed-digit\*/;  
 9. If  $(e_{2_i} = 1)$  Then  $C_3 = M4(T, C_3)$ ; /\* evaluate  $A^{E_{2,c}}$  for positive signed-digit \*/;  
 10. If  $(e_{2_i} = \bar{1})$  Then  $D_3 = M4(T, D_3)$ ; /\* evaluate  $A^{E_{2,c}}$  for negative signed-digit \*/;  
 11. If  $(e_{3_i} = 1)$  Then  $C_4 = M4(T, C_4)$ ; /\* evaluate  $A^{E_{3,c}}$  for positive signed-digit \*/;  
 12. If  $(e_{3_i} = \bar{2})$  Then  $D_4 = M4(T, D_4)$ ; /\* evaluate  $A^{E_{3,c}}$  for negative signed-digit \*/;  
 13.  $S = M4(T, S)$ ;  
 14.  $T = \text{MCR}(S)$ , after performing  $t_0.S$ ;  
 {Parallel end}  
 15. **End for**

---

In Step 2 of Algorithm 7,  $S$  is computed using Algorithm 6. In Step 3, the MCR representation of  $S$  shown by  $T$  is computed by applying Algorithm 5 to  $S$  after executing  $a_0.R$ . This result is utilized in next steps to increase the speed of the modular multiplication in each iteration of Algorithm 7. In Steps 5–12 of Algorithm 7,  $A^{E_{\text{common}}}, A^{E_{1,c}}, A^{E_{2,c}}, A^{E_{3,c}}$  are computed based on the value of the  $e_{ji}$ . These values are computed by executing Algorithm 6. In Steps 13 and 14 of Algorithm 7, the partial result  $S$  and its MCR representation  $T$  are computed by executing Algorithms 6 and 5, respectively. In this algorithm,  $T$  is computed after executing  $t_0.S$  in Step 13. The exponentiation operation  $A^E$  can be computed as (2.3).

#### 4. COMPUTATIONAL COMPLEXITY ANALYSIS OF THE PROPOSED MODULAR EXPONENTIATION

In the proposed CMM-MSD modular exponentiation algorithm, radix-3 signed-digit representation is utilized for the exponent. So, the probability of the digits is as follows:

$$P(0) = \frac{2}{3}, P(1) = P(-1) = P(2) = P(-2) = \frac{1}{12}.$$

TABLE 1. The average number of required multiplication steps in different modular exponentiation algorithms.

Reference	Required multiplication steps (clock cycles)
[4]	$1.5k(2n^2 + n)$
[8]	$0.5k(5n^2 + 4n)$
[31]	$1.833k(n^2 - n - 2)$
This paper	$0.611k(n^2 - 5n - 3)$ .

Based on the computational complexity analysis in [8, 31], for  $n$ -bit modulus and  $k$ -bit exponent, the average number of the required multiplication steps (clock cycles) for the following four operations  $M4(T, C_1)$ ,  $M4(T, C_2)$ ,  $M4(T, C_3)$  and  $M4(T, C_4)$  is as follows:

$$6 \times \frac{1}{12}(1.5k \left(\frac{n}{3} - 2\right) (n + 1)) = \frac{3k}{4} \left(\frac{n}{3} - 2\right) (n + 1).$$

Similarly, the average number of the required multiplication steps (clock cycles) for the following four operations  $M4(T, D_1)$ ,  $M4(T, D_2)$ ,  $M4(T, D_3)$  and  $M4(T, D_4)$  is as follows:

$$6 \times \frac{1}{12}(1.5k \left(\frac{n}{3} - 2\right) (n + 1)) = \frac{3k}{4} \left(\frac{n}{3} - 2\right) (n + 1).$$

In addition, the average number of the required multiplication steps (clock cycles) for the operation  $M4(T, S)$  is as follows:

$$\frac{2}{3} \left(0.5k \left(\frac{n}{3} - 2\right) (n + 1)\right) = \frac{k}{3} \left(\frac{n}{3} - 2\right) (n + 1).$$

Furthermore, converting the integer from the binary representation to the MCR representation is executed in parallel with the previous multiplication. As a result, the average number of the required multiplication steps in the proposed modular exponentiation algorithm is increased only one multiplication step per iteration.

Therefore, the average number of the required multiplication steps (clock cycles) for the proposed modular exponentiation algorithm is as follows:

$$\frac{3k}{4} \left(\frac{n}{3} - 2\right) (n + 1) + \frac{3k}{4} \left(\frac{n}{3} - 2\right) (n + 1) + \frac{k}{3} \left(\frac{n}{3} - 2\right) (n + 1) = 0.611k(n^2 - 5n - 3).$$

However, the conventional Montgomery modular exponentiation algorithms such as [4], the Ha-Moon’s modular exponentiation algorithm [8] and the Wu’s CMM-MSD modular exponentiation algorithm [31] require  $1.5k(2n^2 + n)$ ,  $0.5k(5n^2 + 4n)$ , and  $1.833k(n^2 - n - 2)$  multiplication steps (clock cycles) respectively as shown in Table 1.

This new CMM-MSD modular exponentiation algorithm reduces the average number of the required multiplication steps (clock cycles) in comparison

TABLE 2. The multiplication steps improvement.

Reference	Improvement (%)
[4]	79.6
[8]	75.6
[31]	66.7

with [4, 8, 31] by about

$$1 - \frac{0.611k(n^2 - 5n - 6)}{1.5k(2n^2 + n)} \approx 1 - \frac{0.611}{3} \approx 79.6\%$$

$$1 - \frac{0.611k(n^2 - 5n - 6)}{0.5k(5n^2 + 4n)} \approx 1 - \frac{0.611}{2.5} \approx 75.6\%$$

$$1 - \frac{0.611k(n^2 - 5n - 6)}{1.833k(n^2 - n - 2)} \approx 1 - \frac{0.611}{1.833} \approx 66.7\%.$$

Table 2 summarizes these multiplication steps (clock cycles) improvements for the proposed CMM-MSD modular exponentiation algorithm in comparison with the exponentiation algorithms in [4, 8, 31].

Based on our analysis which is shown in Table 2, the proposed CMM-MSD modular exponentiation algorithm reduces the multiplication steps considerably regarding to the original CMM-MSD proposal to 66.7%.

## 5. CONCLUSION

This paper presents and evaluates a modified Montgomery modular multiplication algorithm based on a multi bit scan-multi bit shift technique and a new integer encoding algorithm. The main idea is that  $k$  time multiplication by zero bits in sequence in  $k$  multiplication steps, followed by the required additions and the nonzero multiplication can be relaxed to one  $k + 1$  bits shift and one binary multiplication in only one clock cycle. Thus, we require specifying the number of zero bits in sequence in the multiplier to determine the number of shifts in each multiplication step. We utilized the MCR encoding algorithm to provide the number of zero bits in sequence in the multiplier. This new encoding algorithm also makes multi bit scan possible. The proposed M4 algorithm is suitable for the hardware and software implementations. We also proposed using a modified, limited number 25 of shifts, Barrel shifter to make multi bit shift possible in each multiplication step in the hardware implementation. Moreover, this new modular multiplication is applied to the CMM-MSD modular exponentiation algorithm [31].

Our complexity analysis shows that the average number of the required multiplication steps (clock cycles) in the proposed CMM-MSD modular exponentiation

algorithm is reduced by about 79.6%, 75.6% and 66.7% in comparison with the Montgomery modular exponentiation algorithms such as [4], the Ha-Moon's CMM modular exponentiation algorithm [8] and the Wu's CMM-MSD modular exponentiation algorithm [31] respectively.

## REFERENCES

- [1] S. Arno and F.S. Wheeler, Signed digit representations of minimal Hamming weight. *IEEE Trans. Comput.* **42** (1993) 1007–1010.
- [2] T. Blum and C. Paar, High-radix Montgomery multiplication on reconfigurable hardware. *IEEE Trans. Comput.* **50** (2001) 759–764.
- [3] A. Booth, A signed binary multiplication technique. *J. Mech. Appl. Math.* **4** (1951) 236–240.
- [4] S.R. Dusse and B.S. Kaliski, A cryptographic library for the Motorola DSP 56000. *Proc. of Adv. Cryptol. EUROCRYPT'90*. In vol. 73 of *Lect. Notes Comput. Sci.* (1990) 230–244.
- [5] O. Egecioglu and C.K. Koc, Exponentiation using Canonical Recoding. *Theoret. Comput. Sci.* **129** (1994) 407–417.
- [6] Y. Fan, T. Ikenaga and S. Goto, A high-speed design of Montgomery multiplier. *IEICE Trans. Fundam. Electron., Commun. Computers* **E91-A** (2008) 971–977.
- [7] A.P. Fournaris and O. Koufopavlou, A new RSA encryption architecture and hardware implementation based on optimized Montgomery multiplication. In *Proc. of IEEE ISCAS* (2005) 4645–4648.
- [8] J.C. Ha and S.J. Moon, A common-multiplicand method to the Montgomery algorithm for speeding up exponentiation. *Inf. Process. Lett.* **66** (1998) 105–107.
- [9] A. Ibrahim, H. Elsimary and F. Gebali, Low-power, high-speed unified and scalable word-based radix-8 architecture for Montgomery modular multiplication. *Arab J. Sci. Eng.* **39** (2014) 7847–7863.
- [10] P. Keshavarzi and C. Harrison, A New Modular Multiplication Algorithm for VLSI Implementation of Public-Key Cryptography. In *Proc. of 1st Int. Symp. Commun. Syst. Digit. Signal Process. CSDSP98, Sheffield, UK* (1998) 516–519.
- [11] D.C. Lee and C.L. Wu, Parallel exponentiation using common-multiplicand-multiplication and signed-digit-folding techniques. *Int. J. Comput. Math.* **81** (2004) 1187–1202.
- [12] P.L. Montgomery, Modular multiplication without trial division. *Math. Comput.* **44** (1985) 519–521.
- [13] N. Nedjah and L.M. Mourelle, High-performance hardware of the sliding-window method for parallel computation of modular exponentiations. *Int. J. Parallel program.* **37** (2009) 537–555.
- [14] N. Nedjah and L.M. Mourelle, A hardware/software co-design versus hardware-only implementation of modular exponentiation using the sliding-window method. *J. Circuits Syst. Comput.* **18** (2009) 295–310.
- [15] J.C. Neto, A.F. Tenca and W.V. Ruggiero, A parallel and uniformly k-partition method for Montgomery multiplication. *IEEE Trans. Comput.* **63** (2014) 2122–2133.
- [16] A. Rezaei and P. Keshavarzi, High-Performance Implementation Approach of Elliptic Curve Cryptosystem for Wireless Network Applications. In *Proc. of IEEE. Int. Conf. Consum. Electron. Commun. Netw. (CECNet 2011)*. Xianning, China (2011) 1323–1327.
- [17] A. Rezaei and P. Keshavarzi, High-performance modular exponentiation algorithm by using a new modified modular multiplication algorithm and common-multiplicand-multiplication method. In *Proc. of IEEE. World Congr. Internet Secur. London, UK* (2011) 192–197.
- [18] A. Rezaei and P. Keshavarzi, CCS Representation: A new non-adjacent form and its application in ECC. *J. Basic. Appl. Sci. Res.* **2** (2012) 4577–4586.
- [19] A. Rezaei and P. Keshavarzi, A new finite field multiplication algorithm to improve elliptic curve cryptosystem implementations. *J. Inf. Syst. Telecom.* **1** (2013) 119–129.

- [20] A. Rezaei and P. Keshavarzi, High-throughput modular multiplication and exponentiation algorithms using multibit-scan-multibit-shift technique. *IEEE Trans. Very Large Scale Integr. Syst.* **23** (2015) 1710–1719.
- [21] G.W. Reitwiesner, Binary arithmetic. *Adv. Comput.* **1** (1960) 231–308.
- [22] G.D. Sutter, J.P. Deschamps and J.L. Imana, Modular multiplication and exponentiation architectures for fast RSA cryptosystem based on digit serial computation. *IEEE Trans. Indust. Electron.* **58** (2011) 3101–3109.
- [23] I. San and N. At, Improving the computational efficiency of modular operations for embedded systems. *J. Syst. Arch.* **60** (2014) 440–451.
- [24] M. Tunstall and M. Joye, The distributions of individual bits in the output of multiplicative operations. *Crypto. Commun.* **7** (2015) 71–90.
- [25] L.A. Tawalbeh, A.F. Tenca and C.K. Koc, A radix-4 scalable design. *IEEE Potentials.* **24** (2005) 16–18.
- [26] A.F. Tenca and L. Tawalbeh, An Efficient and Scalable Radix-4 Modular Multiplier Design using Recoding Techniques. In *Proc. of IEEE. Asilomar Conf. signals, syst., comput., Monterey, CA* (2003) 1445–1450.
- [27] N. Pinckney and D. Harris, Parallelized radix-4 scalable Montgomery multipliers. *J. Integr. Syst.* **3** (2008) 39–45.
- [28] C.D. Walter, Systolic modular multiplication. *IEEE Trans. Comput.* **42** (1993) 376–378.
- [29] T. Wu, S. Li and L. Liu, Fast, compact and symmetric modular exponentiation architecture by common-multiplicand Montgomery modular multiplication. *Integr. VLSI J.* **36** (2013) 323–332.
- [30] C.L. Wu, D.C. Lou and T.J. Chang, An Efficient Montgomery Exponentiation Algorithm for Public-key Cryptosystem. In *Proc. of IEEE. Int. Conf. Intell. Security Inform., Taipei, Taiwan* (2008) 284–285.
- [31] C.L. Wu, An efficient common-multiplicand-multiplication method to the Montgomery algorithm for speeding up exponentiation. *Inform. Sci.* **179** (2009) 410–421.
- [32] C.L. Wu, D.C. Lou and T.J. Chang, Fast modular multiplication based on complement representation and canonical recoding. *Int. J. comput. Math.* **87** (2010) 2871–2879.
- [33] J.J. Xie, J. He and P.K. Meher, Low latency systolic Montgomery multiplier for finite field  $GF(2^m)$  based on pentanomial. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **21** (2013) 385–389.
- [34] M. Yoshino, K. Okeya and C. Vuillaume, Bipartite modular multiplication with twice the bit-length of multipliers. *Int. J. Inform. Security* **8** (2009) 13–23.

Communicated by S. Mesnager.

Received May 16, 2015. Accepted November 2, 2015.