Information Sciences 223 (2013) 331–334

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

# A comment on "An efficient common-multiplicand-multiplication method to the Montgomery algorithm for speeding up exponentiation"

Da-Zhi Sun [a,b,*], Jin-Peng Huai [b], Zhen-Fu Cao [c]

[a] School of Computer Science and Technology, Tianjin University, No. 92 Weijin Road, Nankai District, Tianjin 300072, PR China
[b] School of Computer Science, Beihang University, Beijing 100083, PR China
[c] Department of Computer Science and Engineering, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 200240, PR China

ARTICLE INFO

*Article history:*
Received 20 July 2010
Received in revised form 21 September 2012
Accepted 29 September 2012
Available online 9 October 2012

*Keywords:*
Modular arithmetic
Modular exponentiation
Single-precision multiplication
Computational efficiency
Public-key cryptography

ABSTRACT

In 2009, Wu proposed a fast modular exponentiation algorithm and claimed that the proposed algorithm on average saved about 38.9% and 26.68% of single-precision multiplications as compared to Dussé–Kaliski's Montgomery algorithm and Ha–Moon's Montgomery algorithm, respectively. However, in this comment, we demonstrate that Wu's algorithm on average reduces the number of single-precision multiplications by at most 22.43% and 6.91%, when respectively compared with Dussé–Kaliski's version and Ha–Moon's version. That is, the computational efficiency of Wu's algorithm is obviously overestimated.


## 1. Introduction

The modular exponentiation is the dominant part of the implementation costs in many prevailing public-key cryptosystems. Therefore, Wu [6] proposed a fast modular exponentiation algorithm, of which the idea is to combine the common-multiplicand-multiplication (CMM) Montgomery method [4], the folding exponent method [3,5], and the minimal-signed-digit (MSD) recoding method [1]. According to Wu's claim, the proposed algorithm on average saved about 38.9% and 26.68% of single-precision multiplications as compared to Dussé–Kaliski's Montgomery algorithm [2] and Ha–Moon's Montgomery algorithm [4], respectively.

However, we demonstrate that Wu's algorithm on average reduces the number of single-precision multiplications by at most 22.43% and 6.91%, when respectively compared with Dussé–Kaliski's version and Ha–Moon's version. Our computational efficiency result is accurate, because all crucial operations in Wu's algorithm are considered exactly.

## 2. Brief description of Wu's method

For a self-contained discussion, we briefly review Wu's algorithm and refer the readers to [6] for more details about it. To compute the modular exponentiation $M^E (\bmod N)$, Wu's algorithm can be restated as follows:

**Step 1.** Divide the MSD representation $(e_{k-1} \cdots e_1 e_0)_{MSD}$ for the exponent $E$ into three equal-length bit strings $E_1$, $E_2$, and $E_3$, i.e. $E = E_1 \| E_2 \| E_3$, where $\|$ denotes the bit string concatenation.

* Corresponding author at: School of Computer Science and Technology, Tianjin University, No. 92 Weijin Road, Nankai District, Tianjin 300072, PR China. Tel./fax: +86 22 27406538.
*E-mail addresses:* sundazhi@tju.edu.cn, sundazhi1977@126.com (D.-Z. Sun).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.09.052

**Step 2.** Compute

$$E_{com} = E_1 \textbf{ AND } E_2 \textbf{ AND } E_3 = (e^0_m \cdots e^0_1 e^0_0), \quad (1)$$
$$E'_1 = E_1 \textbf{ XOR } E_{com} = (e^1_m \cdots e^1_1 e^1_0), \quad (2)$$
$$E'_2 = E_2 \textbf{ XOR } E_{com} = (e^2_m \cdots e^2_1 e^2_0), \quad (3)$$
$$E'_3 = E_3 \textbf{ XOR } E_{com} = (e^3_m \cdots e^3_1 e^3_0), \quad (4)$$

where $m = \lceil \frac{k}{3} \rceil - 1$ and $\lceil\ \rceil$ denotes the usual ceiling function. The definitions of the bitwise logical "**AND**" and "**XOR**" operators are presented in **Table 1** of [6]. Next, let the bit strings $E_{com[1]} = \left(e^{0[1]}_m \cdots e^{0[1]}_1 e^{0[1]}_0\right)$ and $E_{com[-1]} = \left(e^{0[-1]}_m \cdots e^{0[-1]}_1 e^{0[-1]}_0\right)$ separately store all bits of 1 and all bits of $-1$ in the bit string $E_{com}$. Similarly, let the bit strings $E'_{i[1]} = \left(e^{i[1]}_m \cdots e^{i[1]}_1 e^{i[1]}_0\right)$ and $E'_{i[-1]} = \left(e^{i[-1]}_m \cdots e^{i[-1]}_1 e^{i[-1]}_0\right)$ separately store all bits of 1 and all bits of $-1$ in the corresponding bit strings $E'_i$ for $i$ = 1, 2, 3.

**Step 3.** Use the so-called improved CMM–MSD Montgomery algorithm described in Section 3.4 of [6] to compute the values $M^{E_{com[1]}}(\text{mod } N)$, $M^{-E_{com[-1]}}(\text{mod } N)$, $M^{E'_{i[1]}}(\text{mod } N)$, and $M^{-E'_{i[-1]}}(\text{mod } N)$, for $i$ = 1, 2, 3.

**Step 4.** Compute the intermediate exponentiation values as:

$$M^{E_i}(\text{mod } N) = M^{E_{com[1]}} M^{E'_{i[1]}} \left(M^{-E_{com[-1]}} M^{-E'_{i[-1]}}\right)^{-1} (\text{mod } N) \text{ for } i = 1,\ 2,\ 3. \quad (5)$$

**Step 5.** The modular exponentiation $M^E(\text{mod} N)$ can be calculated as follows:

$$M^E = M^{E_1 \| E_2 \| E_3} = ((M^{E_1})^{2^{m+1}} (M^{E_2}))^{2^{m+1}} M^{E_3} (\text{mod } N). \quad (6)$$

For efficiency evaluation, the improved CMM–MSD Montgomery algorithm mentioned in **Step 3** can be rewritten as Fig. 1. Here, MMR() denotes the CMM Montgomery method [4].

## 3. Computational efficiency of Wu's method

### 3.1. Preliminaries

Let Pr ($EV$) denote the probability that the event $EV$ occurs. There is a well-known property of the MSD representation [1] as follows.

**Algorithm 1**

INPUT : $M, N, R = b^n (\text{mod } N), E_{com[1]} = \left(e^{0[1]}_m \cdots e^{0[1]}_1 e^{0[1]}_0\right), E_{com[-1]} = \left(e^{0[-1]}_m \cdots e^{0[-1]}_1 e^{0[-1]}_0\right),$
$E'_{i[1]} = \left(e^{i[1]}_m \cdots e^{i[1]}_1 e^{i[1]}_0\right)$, and $E'_{i[-1]} = \left(e^{i[-1]}_m \cdots e^{i[-1]}_1 e^{i[-1]}_0\right)$, for $i = 1, 2, 3$.

OUTPUT : $C_0 = M^{E_{com[1]}} (\text{mod } N), D_0 = M^{-E_{com[-1]}} (\text{mod } N), C_i = M^{E'_{i[1]}} (\text{mod } N)$, and
$D_i = M^{-E'_{i[-1]}} (\text{mod } N)$, for $i = 1, 2, 3$.

Step A1-1: $C_0 = C_1 = C_2 = C_3 = D_0 = D_1 = D_2 = D_3 = R(\text{mod } N), S = MR(\text{mod } N)$;
Step A1-2: for $i = 0$ to $m$ do{
Step A1-2.1: if $e^{0[1]}_i = 1$ then $C_0 = \text{MMR}(C_0 S)$; // evaluate $M^{E_{com[1]}} (\text{mod } N)$
Step A1-2.2: if $e^{0[-1]}_i = -1$ then $D_0 = \text{MMR}(D_0 S)$; // evaluate $M^{-E_{com[-1]}} (\text{mod } N)$
Step A1-2.3: if $e^{1[1]}_i = 1$ then $C_1 = \text{MMR}(C_1 S)$; // evaluate $M^{E'_{1[1]}} (\text{mod } N)$
Step A1-2.4: if $e^{1[-1]}_i = -1$ then $D_1 = \text{MMR}(D_1 S)$; // evaluate $M^{-E'_{1[-1]}} (\text{mod } N)$
Step A1-2.5: if $e^{2[1]}_i = 1$ then $C_2 = \text{MMR}(C_2 S)$; // evaluate $M^{E'_{2[1]}} (\text{mod } N)$
Step A1-2.6: if $e^{2[-1]}_i = -1$ then $D_2 = \text{MMR}(D_2 S)$; // evaluate $M^{-E'_{2[-1]}} (\text{mod } N)$
Step A1-2.7: if $e^{3[1]}_i = 1$ then $C_3 = \text{MMR}(C_3 S)$; // evaluate $M^{E'_{3[1]}} (\text{mod } N)$
Step A1-2.8: if $e^{3[-1]}_i = -1$ then $D_3 = \text{MMR}(D_3 S)$; // evaluate $M^{-E'_{3[-1]}} (\text{mod } N)$
Step A1-2.9: $S = \text{MMR}(SS)$;}
Step A1-3: $C_0 = \text{MMR}(C_0), D_0 = \text{MMR}(D_0), C_1 = \text{MMR}(C_1), D_1 = \text{MMR}(D_1),$
$C_2 = \text{MMR}(C_2), D_2 = \text{MMR}(D_2), C_3 = \text{MMR}(C_3), D_3 = \text{MMR}(D_3)$;
Step A1-4: Return $(C_0, D_0, C_1, D_1, C_2, D_2, C_3, D_3)$.

**Fig. 1.** Improved Montgomery modular exponentiation algorithm.