Intern. J. Computer Math., Vol. 40, pp. 139–156
Reprints available directly from the publisher
Photocopying permitted by license only

HIGH-RADIX AND BIT RECODING TECHNIQUES FOR MODULAR EXPONENTIATION

ÇETIN K. KOÇ

Department of Electrical Engineering, University of Houston, Houston, TX 77204

(Received 22 November 1990)

Algorithms that make use of high-radix and bit recoding techniques to perform modular exponentiation are proposed. It is shown that the high-radix methods with optimal choice of the radix provide significant reductions in the number of multiplications required for modular exponentiation. It is then shown that bit recoding techniques similar to those used in binary multiplication algorithms can be used to further reduce the total number of multiplications. The algorithms presented are analyzed by counting the maximum and the average number of multiplications required.

KEY WORDS: Modular exponentiation, binary method, m-ary method, bit recoding, RSA cryptosystem.

C.R. CATEGORIES: E.3, F.2.1, G.1.0.

1 INTRODUCTION

The modular exponentiation problem, i.e., the computation of

$$C = M^E \pmod{N} \tag{1}$$

has an interesting and very important application in public key cryptography. The RSA cryptosystem is based on modular exponentiation of the plaintext M to produce the ciphertext C using the publicly available encryption key E [12]. The decryption, i.e., the computation of M given C, is also a modular exponentiation using the secret key. The RSA cryptosystem is considered secure if the integers E and N have several hundred decimal digits [11].

In this paper we propose algorithms based on high-radix representation and recoding of E for the modular exponentiation problem. The algorithms described here require fewer multiplications than the binary method, commonly known as the *square-and-multiply* method. We show that high-radix and bit recoding techniques provide significant reduction in the number of multiplications required to compute (1) when E has several hundred bits. The bit recoding techniques used are similar to those used in binary multiplication algorithms (for example, the Booth recoding algorithm and its variations) to reduce the total number of partial products [5, 15, 14].

In Section 2, we describe the binary and the high-radix algorithms given by Knuth [7]. We restrict our attention to the case where the radix is a power of 2. We count the maximum and the average number of multiplications for the binary and the

high-radix algorithms in terms of $n = \lfloor \log_2 E \rfloor + 1$ (i.e., the number of bits required to represent E) and the radix $m = 2^d$. We show that the number of multiplications required by the radix m method can be minimized by proper selection of m. When $n = 1024$, the optimum choice of m provides 39 % reduction in the maximum number of multiplications and 19% reduction in the average number of multiplications required for the computation of (1).

We then show in Section 3 that the use of bit recoding schemes can provide further reduction in the average number of multiplications required to compute (1). For example, the recoded binary method requires 17% percent fewer multiplications than the binary method. In Section 4, we give some examples and show step by step how (1) is computed by each one of the algorithms presented in this paper.

Finally, in Section 5, we discuss the previous work regarding the implementation of the RSA algorithm and also compare the high-radix and bit recoding algorithms to other proposed techniques, especially to the factor method and the power tree method given by Knuth [7]. While these methods are superior for small values of the exponent, the high-radix and bit recoding algorithms are efficient for large values of the exponent. Furthermore, the implementation of the high-radix and bit recoding algorithms is quite simple and requires very little overhead.

2 BINARY AND HIGH RADIX ALGORITHMS

Let n be the number of bits of E, i.e., $n = \lfloor \log_2 E \rfloor + 1$, with

$$E = [E_{n-1}E_{n-2}\cdots E_1E_0] = \sum_{i=0}^{n-1} E_i 2^i \tag{2}$$

for $E_i \in \{0, 1\}$. The best known algorithm for computing (1), the *binary method*, can be found in [7]. It is based on repated squaring of M and multiplications whenever the corresponding bit of E is 1. The binary method to compute (1) given below scans the bits of the exponent from left to right. Another version of this algorithm scans the bits of E from right to left.

Binary Method (BM)
Input: M, N, E, n where $n = \lfloor \log_2 E \rfloor + 1$.
Output: $C = M^E \pmod N$.
Step 1. If $E_{n-1} = 1$ then $C = M$ else $C = 1$.
Step 2. Repeat Step 2a and 2b for $i = n - 2, n - 3, \ldots, 0$.
Step 2a. $C = C * C \pmod N$.
Step 2b. If $E_i = 1$ then $C = C * M \pmod N$.
Step 3. Halt.

The binary method can be generalized to the *m-ary method* which scans the digits of E expressed in radix m [7]. We restrict our attention to the case when $m = 2^d$.

The exponent E is partitioned into k sections of d bit each for $kd = n$. If d does not divide n, the exponent is padded with at most $d - 1$ zeros. Now, we define

$$F^{(i)} = [E_{id+d-1}E_{id+d-2}\cdots E_{id}] = \sum_{r=0}^{d-1} E_{id+r}2^r, \tag{3}$$

then from (2) we have

$$E = \sum_{i=0}^{n-1} E_i 2^i = \sum_{i=0}^{k-1} F^{(i)}2^{id}. \tag{4}$$

First, the values of $X_j = M^j \pmod N$ for $j = 2, 3, \ldots, 2^d - 1$ are computed. Then, the bits of E are scanned d bits at a time from the most significant to the least significant. At each step the partial result is raised to the 2^d power and multiplied with $X_{F^{(i)}}$ where $F^{(i)}$ is the value of the current bit section.

M-ary Method (MM)

Input: M, N, E, n, d where $n = \lfloor \log_2 E \rfloor + 1$ and $n = kd$ for $k \geq 1$.
Output: $C = M^E \pmod N$.
Step 1. Set $X_0 = 1$ and $X_1 = M$.
Step 2. Repeat Step 2a for $j = 2, 3, \ldots, 2^d - 1$.
  Step 2a. $X_j = X_{j-1} * M \pmod N$.
Step 3. Repeat Step 3a for $i = k - 1, k - 2, \ldots, 0$.
  Step 3a. $F^{(i)} = \sum_{r=0}^{d-1} E_{id+r}2^r$.
Step 4. Set $C = X_{F^{(k-1)}}$.
Step 5. Repeat Step 5a and Step 5b for $i = k - 2, k - 3, \ldots, 0$.
  Step 5a. Repeat Step 5aa for $j = 0, 1, \ldots, d - 1$.
    Step 5aa. $C = C * C \pmod N$.
  Step 5b. If $F^{(i)} \neq 0$ then $C = C * X_{F_i} \pmod N$.
Step 6. Halt.

First, we observe that Step 2a of Binary Method (BM) requires $n - 1$ multiplications, and Step 2b requires at most $n - 1$ multiplications. Thus, BM requires at most

$$T_{\mathrm{BM}}^{\max}(n) = 2(n - 1) \tag{5}$$

multiplications. We give the following theorem regarding the maximum number of multiplications required by the radix m algorithm.

THEOREM 1 *M-ary Method requires at most* $T_{\mathrm{MM}}^{\max}(n, d) = n + (n/d) + 2^d - d - 3$ *multiplications.*

Proof Step 2a is executed $2^d - 2$ times. Step 5a requires $(k - 1)d$ multiplications, while Step 5b requires at most $k - 1$ multiplications. Summing these numbers, we

obtain

$$T_{\mathrm{MM}}^{\max}(n, d) = 2^d - 2 + d(k-1) + k - 1 = n + \frac{n}{d} + 2^d - d - 3, \tag{6}$$

since $n = kd$. ■

When $d = 1$ the number of multiplications becomes

$$T_{\mathrm{MM}}^{\max}(n, 1) = 2(n-1) = T_{\mathrm{BM}}^{\max}(n),$$

since MM reduces to BM. However, there is an additional interesting property. It is possible to select d for a given n such that $T_{\mathrm{MM}}^{\max}(n, d)$ is minimized. Let $DT_{\mathrm{MM}}^{\max}(n, d)$ denote the derivative of $T_{\mathrm{MM}}^{\max}(n, d)$ with respect to d. In order to find the optimal value of d minimizing the maximum number of multiplications executed by MM, we need to solve

$$DT_{\mathrm{MM}}^{\max}(n, d) = -\frac{n}{d^2} + 2^d \log_e 2 - 1 = 0. \tag{7}$$

This equation contains integers n and m and transcendental number $\log_e 2 = 0.69314718, \ldots$; thus, $DT_{\mathrm{MM}}^{\max}(n, d)$ can never be made exactly zero for an integral value of d. However, by enumeration, a value of $d = d^*$ can be found such that $|DT_{\mathrm{MM}}^{\max}(n, d^*)|$ is as close to zero as possible. Table 1 contains the optimum values of d for $n = 4, 8, 16, \ldots, 65536$ together with the values of $\lceil T_{\mathrm{BM}}^{\max}(n) \rceil$ and $\lceil T_{\mathrm{MM}}^{\max}(n, d^*) \rceil$.

Theorem 1 gives the maximum number of multiplications required by MM. If the binary representation of the exponent has fewer 1s, then the total number of

Table 1 Maximum number of multiplications required by algorithms BM, MM, RBM, and RMM.

	BM	MM		RBM		RMM
n	$\lceil T_{\mathrm{BM}}^{\max}(n) \rceil$	d^*	$\lceil T_{\mathrm{MM}}^{\max}(n, d^*) \rceil$	$\lceil T_{\mathrm{RBM}}^{\max}(n) \rceil$	d^*	$\lceil T_{\mathrm{RMM}}^{\max}(n, d^*) \rceil$
4	6	2	5	5	1	5
8	14	2	11	12	1	12
16	30	2,3	23	25	1,2	25
32	62	3	45	52	2	49
64	126	3	87	105	3	93
128	254	4	169	212	3	179
256	510	4	329	425	4	343
512	1022	5	638	852	4	663
1024	2046	6	1250	1705	5	1283
2048	4094	6	2444	3412	6	2506
4096	8190	7	4799	6825	6	4896
8192	16382	8	9461	13652	7	9606
16384	32766	8	18677	27305	8	18931
32768	65534	9	36909	54612	8	37363
65536	131070	10	73101	109225	9	73828

multiplications required by the binary method is less than $2(n-1)$. Respectively, the number of multiplications required by MM is a function of the number of nonzero digits of the exponent expressed in radix m. Thus, it is worthwhile to examine the average behavior of these algorithms for all possible values of the exponent.

THEOREM 2 *M-ary Method requires* $T_{\mathrm{MM}}^{\mathrm{avg}}(n, d) = n + ((n/d) - 1)(1 - (1/2^d)) + 2^d - d - 2$ *multiplications on the average.*

Proof Step 2a of MM requires $2^d - 2$ multiplications for any value of the exponent. Similarly, Step 5a requires $(k-1)d$ multiplications. Whenever $F^{(i)} = 0$, we do not perform multiplication in Step 5b. The average number of multiplications required in Step 5b is $(k-1)((m-1)/m)$ since $F^{(i)} \in \{0, 1, 2, 3, \ldots, m-1\}$, i.e., multiplication is *not* performed in only one in m cases. Since $m = 2^d$ and $n = kd$, we obtain

$$\begin{aligned} T_{\mathrm{MM}}^{\mathrm{avg}}(n, d) &= 2^d - 2 + n - d + \left(\frac{n}{d} - 1\right)\left(\frac{2^d - 1}{2^d}\right) \\ &= n + \left(\frac{n}{d} - 1\right)\left(1 - \frac{1}{2^d}\right) + 2^d - d - 2 \end{aligned} \tag{8}$$

as claimed. ■

The average number of multiplications for the binary method can also be found simply by substituting $d = 1$ in the above formulae.

$$\begin{aligned} T_{\mathrm{BM}}^{\mathrm{avg}}(n) &= n + (n-1)\left(1 - \frac{1}{2}\right) + 2 - 1 - 2 \\ &= \frac{3}{2}(n-1). \end{aligned} \tag{9}$$

Similar to the worst case analysis, the average number of multiplications required by MM can be minimized. The optimal value of d can be found by solving

$$DT_{\mathrm{MM}}^{\mathrm{avg}}(n, d) = -\frac{n}{d^2}\left(1 - \frac{1}{2^d}\right) + \left(\frac{n}{d} - 1\right)2^{-d} \log_e 2 + 2^d \log_e 2 - 1 = 0. \tag{10}$$

In Table 2 we show the optimal values of d (found by enumeration) and also $\lceil T_{\mathrm{BM}}^{\mathrm{avg}}(n) \rceil$ and $\lceil T_{\mathrm{MM}}^{\mathrm{avg}}(n, d^*) \rceil$.

3 BIT RECODING TECHNIQUES

The number of multiplications required by the binary and m-ary algorithms to compute (1) consists of two parts. First, M^2, M^4, $M^8, \ldots, M^{2^{n-1}} \pmod N$ are computed (in Step 2a of BM or Step 5a of MM). This step requires $n-1$ multiplications (squaring) operations for either the binary or the m-ary method. There is

Table 2 Average number of multiplications required by algorithms BM, MM, RBM, and RMM.

	BM		MM	RBM		RMM
n	$\lceil T_{\mathrm{BM}}^{\mathrm{avg}}(n) \rceil$	d^*	$\lceil T_{\mathrm{MM}}^{\mathrm{avg}}(n, d^*) \rceil$	$\lceil T_{\mathrm{RBM}}^{\mathrm{avg}}(n) \rceil$	d^*	$\lceil T_{\mathrm{RMM}}^{\mathrm{avg}}(n, d^*) \rceil$
4	5	1,2	5	5	1	5
8	11	2	10	10	1	10
16	23	2	21	21	1	21
32	47	2,3	43	43	1,2	43
64	95	3	85	87	2	85
128	191	3,4	167	175	2,3	168
256	383	4	325	351	3	329
512	767	5	635	703	4	644
1024	1535	5	1246	1407	5	1263
2048	3071	6	2439	2815	5	2473
4096	6143	7	4795	5631	6	4855
8192	12287	8	9457	11263	7	9563
16384	24575	8	18669	22527	7	18881
32768	49151	9	36902	45055	8	37268
65536	98303	10	73094	90111	9	73722

no apparent technique to reduce the number of multiplications in this step. Second, we compute $M^E \pmod N$ by multiplying the partial result with 2^i power of M if the ith bit of the exponent is nonzero. No multiplication is performed if this bit is zero. Therefore, the number of multiplications required is equal to the number of nonzero bits in the binary representation of E. It seems thus worthwhile to investigate techniques to recode E in order to increase the number of zero bits in its representation.

Recoding techniques (Booth recoding, bit-pairs recoding, etc.) have been effectively used in binary multiplication [5, 15]. The original Booth recoding technique [1] scans the bits of the multiplier one bit at a time, and adds or subtracts the multiplicand to or from the partial product, depending on the value of the current bit and the previous bit. If these two bits are 00 or 11, no action is performed. We add or subtract the multiplicand to or from the partial product if the bits are 01 or 10, respectively. Thus, the Booth algorithm skips over a string of 1s as well as a string of 0s, in addition to the standard multiplication algorithms which skip over a string of 0s only. The modified versions of the Booth algorithm scans the bits of the multiplier two bits at a time [9] or three bits at a time [15]. These techniques are equivalent in the sense that a signed digit representation of the multiplier is utilized.

The bit recoding techniques can be applied to the modular exponentiation problem provided that $M^{-1} \pmod N$ is supplied with M. Suppose we compute $M^E \pmod N$ where E contains a string of 1s of length i starting from jth bit. The value of this part of the exponent is equivalent to

$$E = 2^j + 2^{j+1} + \cdots + 2^{j+i-2} + 2^{j+i-1} = 2^j(1 + 2 + 2^2 + \cdots + 2^{i-1})$$
$$= 2^j(2^i - 1) = 2^{i+j} - 2^j.$$

Let $\bar{1}$ represent -1. The above operation can be represented as a recoding of E in the following way:

bit location	$j+i$	$j+i-1$	$j+i-2$	$\cdots$	$j+1$	j	$j-1$
number E	0	1	1	$\cdots$	1	1	0
recoded number D	1	0	0	$\cdots$	0	$\bar{1}$	0

The computation of $M^E \pmod N$ requires i multiplications (in addition to the number of multiplications performed to compute all power of 2 powers of M). However, we compute $M^D = M^E$ using only 2 multiplications since D contains only 2 nonzero digits. Also note that

$$M^D = M^{2^{i+j}-2^j} = M^{2^{i+j}} * M^{-2^j} = M^{2^{i+j}} * (M^{-1})^{2^j},$$

i.e., we need to have $M^{-1} \pmod N$ available to compute M^D. The inverse of M modulo N can be computed using Euclid's extended algorithm [7, 8]. Throughout this paper, we will assume that $M^{-1} \pmod N$ is supplied with M.

We now give an algorithm which first computes the recoded exponent and then scans the recoded exponent d bits at a time in order to reduce the number of multiplications for the computation of (1). Rigorous analyses of the algorithms treating the worst and the average cases separately are also provided.

The Booth recoding technique (and its modified versions) recodes 011110 as $100\bar{1}0$. When the number contains an isolated 1, it is recoded as $1\bar{1}$ while 0110 is recoded as $10\bar{1}0$. The latter case is acceptable since it does not increase the total number of 1s, while the former case will introduce more nonzero bits to the number. For example 0101 is recoded as $1\bar{1}1\bar{1}$. The recoding technique given in Table 3 scans the bits of the exponent 4 bits at a time sharing 1 bit with the previous and 2 bits with the next case. This technique recodes the number in such a way that the isolated

Table 3 Recoding of the exponent.

E_{i+1}	E_i	E_{i-1}	E_{i-2}	D_i	H_iG_i
0	0	0	0	0	00
0	0	0	1	0	00
0	0	1	0	0	00
0	0	1	1	1	01
0	1	0	0	1	01
0	1	0	1	1	01
0	1	1	0	0	00
0	1	1	1	0	00
1	0	0	0	0	00
1	0	0	1	0	00
1	0	1	0	0	00
1	0	1	1	1	01
1	1	0	0	$\bar{1}$	11
1	1	0	1	$\bar{1}$	11
1	1	1	0	0	00
1	1	1	1	0	00

1s stay untouched. Also 0110 is recoded as $10\bar{1}0$ and any 1-string of length $i \geq 3$ is recoded as $10\cdots0\bar{1}$.

In order to represent $\bar{1}$ we introduce two binary numbers, H and G such that

$$H_iG_i = \begin{cases} 00 & \text{if } D_i = 0 \\ 01 & \text{if } D_i = 1 \\ 11 & \text{if } D_i = \bar{1} \end{cases} \tag{11}$$

for $i = 0, 1, \ldots, n-1$. The value of D_i can be computed using H_iG_i as

$$D_i = -2H_i + G_i \qquad \text{for} \quad 0 \leq i \leq n-1. \tag{12}$$

We now redefine $F^{(i)}$ to represent the value of a bit section of length d of the *recoded exponent* D, i.e.,

$$F^{(i)} = [D_{id+d-1}D_{id+d-2}\cdots D_{id}] = \sum_{r=0}^{d-1} D_{id+r}2^r = \sum_{r=0}^{d-1} (-2H_{id+r} + G_{id+r})2^r. \tag{13}$$

Similar to the computation of X_j for $0 \leq j \leq 2^d - 1$, we introduce the variables Y_j for $-2^d + 1 \leq j \leq 2^d - 1$ such that

$$Y_j = M^j \pmod{N} \qquad \text{where} \quad j = F^{(i)}.$$

Note that the number of Y_js is $2^{d+1} - 1 = 2m - 1$ because the current bit section may take the values

$$F^{(i)} \in \{-2^d + 1, -2^d, \ldots, -2, -1, 0, 1, 2, \ldots, 2^d - 1\}.$$

The following algorithm first computes Y_j for $j = -2^d + 1, \ldots, -2, -1, 0, 1, 2, \ldots, 2^{d+1} - 1$ given M and $M^{-1} \pmod{N}$. It then proceeds to compute $C = M^E \pmod{N}$ by repeated squaring of the partial result C, and by multiplying the partial result with $Y_{F^{(i)}}$ if $F^{(i)} \neq 0$.

Recoded M-ary Method (RMM)

Input: M, N, E, n, d where $n = \lfloor \log_2 E \rfloor + 1$ and $n = kd$ for $k \geq 1$.
Output: $C = M^E \pmod{N}$.
Step 1. Set $Y_0 = 1$ and $Y_1 = M$.
Step 2. Repeat Step 2a for $j = 2, 3, 4, \ldots, 2^d - 1$
  Step 2a. $Y_j = Y_{j-1} * M \pmod{N}$.
Step 3. Set $Y_{-1} = M^{-1} \pmod{N}$.
Step 4. Repeat Step 4a for $j = -2, -3, -4, \ldots, -2^d + 1$
  Step 4a. $Y_j = Y_{j+1} * M^{-1} \pmod{N}$.
Step 5. Repeat Step 5a for $i = n-1, n-2, \ldots, 0$.
  Step 5a. If $E_iE_{i-1}E_{i-2} = 011$ then $H_iG_i = 01$,
    Else if $E_{i+1}E_iE_{i-1} = 010$ then $H_iG_i = 01$,
    Else if $E_{i+1}E_iE_{i-1} = 110$ then $H_iG_i = 11$,
    Else $H_iG_i = 00$.

Step 6. Repeat Step 6a for $i = k-1, k-2, \ldots, 0$.
Step 6a. $F^{(i)} = \sum_{r=0}^{d-1}(-2H_{id+r} + G_{id+r})2^r$.
Step 7. Set $C = Y_{F^{(k-1)}}$.
Step 8. Repeat Step 8a and 8b for $i = k-2, k-3, \ldots, 0$.
Step 8a. Repeat Step 8aa for $j = 0, 1, \ldots, d-1$.
Step 8aa. $C = C * C \pmod N$.
Step 8b. If $F^{(i)} \neq 0$ then $C = C * Y_{F^{(i)}} \pmod N$.
Step 9. Halt.

Recoded Binary Method (RBM) corresponds to the case $d = 1$ in the above algorithm. The number of multiplications required by RBM is maximum when E contains all 1-strings of length 2 each, e.g, $E = 011011011011$. Recoded exponent D contains as many nonzero bits as there are in E since 011 is recoded as $10\bar{1}$. Thus, the maximum number of multiplications required by RBM is found as

$$T_{\text{RBM}}^{\max}(n) = n - 1 + \frac{2}{3}(n-1) = \frac{5}{3}(n-1). \tag{14}$$

The maximum number of multiplications performed by RMM is given in the following theorem.

THEOREM 3 *Recoded M-ary Method requires at most* $T_{\text{RMM}}^{\max}(n, d) = n + (n/d) + 2^{d+1} - d - 5$ *multiplications for* $d \geq 2$.

Proof Step 2a is executed $2^d - 2$ times, and Step 4a is executed $2^d - 2$ times. Step 8a requires $(k-1)d$ multiplications for any value of the exponent. Since the recoded exponent may contain strings of the form $10\bar{1}$, there exists a recoded exponent for which $F^{(i)} \neq 0$ for $0 \leq i \leq k-1$. Thus, Step 8b may require as many as $k-1$ *multiplications for* $d \geq 2$. We thus obtain

$$\begin{aligned} T_{\text{RMM}}^{\max}(n, d) &= 2^d - 2 + 2^d - 2 + d(k-1) + k - 1 \\ &= n + \frac{n}{d} + 2^{d+1} - d - 5. \quad \blacksquare \end{aligned} \tag{15}$$

In Table 1 we give the maximum number of multiplications required by RBM and RMM for $n = 4, 8, 16, \ldots, 65536$. This table also contains the optimum values of d which minimize the maximum number of multiplications required by RMM.

The average number of multiplications performed by RMM requires computation of the average number of nonzero bits (i.e., 1s and $\bar{1}$s) in the *recoded* binary expansion of E. The *unrecoded* binary expansion of E contains an average of $n/2$ nonzero bits since each bit is equal to 0 or 1.

LEMMA 1 *The average number of nonzero bits in the recoded binary expansion of* E *is* $(3n+1)/8 + (5n-1)/2^n \approx \frac{3}{8}n$. *Thus, a bit of the recoded exponent is equal to zero with probability* $\frac{5}{8}$.

Proof First we count the total number of 1-strings of length i. A string of 1s of length i starting at bit position j will end at bit position $j+i$ for $j=0, 1, 2, \ldots, n-1-i$, as illustrated below.

$$
\begin{array}{lll}
j=1 & \overbrace{xx\cdots xxxx}^{n-i-1}0\overbrace{11\cdots 11}^{i} & 2^{n-i-1} \\
j=2 & \overbrace{xx\cdots xxx}^{n-i-2}0\overbrace{11\cdots 11}^{i}0 & 2^{n-i-2} \\
j=3 & \overbrace{xx\cdots xx}^{n-i-3}0\overbrace{11\cdots 11}^{i}0x & 2^{n-i-2} \\
j=4 & \overbrace{xx\cdots x}^{n-i-4}0\overbrace{11\cdots 11}^{i}0xx & 2^{n-i-2} \\
 & \cdots & \\
j=n-i & 0\overbrace{11\cdots 11}^{i}0\overbrace{xx\cdots xxx}^{n-i-2} & 2^{n-i-2} \\
j=n-i+1 & \overbrace{11\cdots 11}^{i}0\overbrace{xx\cdots xxxx}^{n-i-1} & 2^{n-i-1}
\end{array}
$$

where $x\in\{0, 1\}$. The last column of the above table shows the total number of numbers containing a string of 1s of length i starting from jth location.

For $i\geq 3$, we recode each of the numbers above by replacing a string of 1s of length i with a $\bar{1}$ at bit position j and a 1 at bit position $j+i+1$. Thus, the number of nonzero bits becomes $i-2$ less than it previously was, except for the last entry in the above table where the reduction in the number of nonzero bits is $i-1$. Notice that when $j=n-i+1$ (the last entry of the above table), the numbers are recoded as

$$\overbrace{00\cdots 0\bar{1}}^{i}\ 0\ \overbrace{xx\cdots xx}^{n-i-1},$$

since the recoding technique treats E as a negative number if the most significant bit of E is equal to 1. As an example, let $n=4$ and $E=1110$. We recode E as $00\bar{1}0$, and compute $M^{-2}\pmod N$ instead of $M^{14}\pmod N$. If this is not desired, then we must extend E by appending a zero on the left, prior to the execution of RMM.

The total number of bits in numbers $0, 1, 2, \ldots, 2^n-1$ is equal to $n2^n$, half of which are nonzero bits. After the recoding the number of nonzero bits becomes

$$
\begin{aligned}
W(n) &= \frac{n2^n}{2}-(n-1)-\sum_{i=3}^{n-1} \\
&\quad\times\left[2^{n-i-1}(i-2)+\overbrace{(2^{n-i-2}+\cdots+2^{n-i-2})}^{n-i-1}(i-2)+2^{n-i-1}(i-1)\right] \\
&= n2^{n-1}-(n-1)-\sum_{i=3}^{n-1}[2^{n-i-1}(2i-3)+2^{n-i-2}(n-i-1)(i-2)]
\end{aligned}
$$

where the second term (i.e., $(n-1)$) corresponds to the case $i = n$, in which we recode $2^n - 1$ as -1. We find the average number of nonzero bits (i.e., average Hamming weight) in the recoded binary expansion of E as $W^{\text{avg}}(n) = 2^{-n}W(n)$, i.e.,

$$\begin{aligned} W^{\text{avg}}(n) &= \frac{n}{2} - \frac{n-1}{2^n} - \sum_{i=3}^{n-1} [2^{-i-1}(2i-3) + 2^{-i-2}(n-i-1)(i-2)] \\ &= \frac{n}{2} - \frac{n-1}{2^n} - \sum_{i=3}^{n-1} 2^{-i}\left[\frac{2i-3}{2} + \frac{(n-i-1)(i-2)}{4}\right] \\ &= \frac{n}{2} - \frac{n-1}{2^n} + \frac{1}{4}\sum_{i=3}^{n-1} i^2 2^{-i} - \frac{n+5}{4}\sum_{i=3}^{n-1} i2^{-i} + \frac{n+2}{2}\sum_{i=3}^{n-1} 2^{-i}. \end{aligned}$$

Here we note that

$$\sum_{i=3}^{n-1} i^2 2^{-i} = \frac{9}{2} - \frac{n^2+2n+3}{2^{n-1}}, \qquad \sum_{i=3}^{n-1} i2^{-i} = 1 - \frac{n+1}{2^{n-1}}, \quad \text{and} \quad \sum_{i=3}^{n-1} 2^{-i} = \frac{1}{4} - \frac{1}{2^{n-1}},$$

all of which can be proved by induction on n. The average number of nonzero bits in the recoded binary expansion of E is found as

$$\begin{aligned} W^{\text{avg}}(n) &= \frac{n}{2} - \frac{n-1}{2^n} + \frac{1}{4}\left(\frac{9}{2} - \frac{n^2+2n+3}{2^{n-1}}\right) \\ &\quad - \frac{n+5}{4}\left(1 - \frac{n+1}{2^{n-1}}\right) + \frac{n+2}{2}\left(\frac{1}{4} - \frac{1}{2^{n-1}}\right) \\ &= \left(\frac{n}{2} - \frac{n+5}{4} + \frac{n+2}{8}\right) + \frac{9}{8} + 2^{-n}\left[-\frac{n^2+2n+3}{2} + \frac{n^2+6n+5}{2} - (n+2)\right] \\ &= \frac{3n-1}{8} + \frac{9}{8} + 2^{-n}(5n-1) \\ &= \frac{3n+1}{8} + \frac{5n-1}{2^n} \approx \frac{3}{8}n \end{aligned}$$

as claimed. ■

In the following theorem we give the average number of multiplications required by RMM.

THEOREM 4 *Recoded M-ary Method requires*

$$T_{\text{RMM}}^{\text{avg}}(n, d) = n + \left(\frac{n}{d} - 1\right)\left[1 - \left(\frac{5}{8}\right)^d\right] + 2^{d+1} - d - 4$$

multiplications on the average.

Proof It follows from the above lemma that a bit section of the recoded exponent, $F^{(i)}$ is equal to zero with probability

$$\left(\frac{5}{8}\right)^d,$$

since all of its d bits have to be equal to zero. Thus, Step 7b of RMM requires

$$(k-1)\left[1-\left(\frac{5}{8}\right)^d\right]$$

multiplications on the average. The total number of multiplications becomes

$$\begin{aligned} T^{\text{avg}}_{\text{RMM}}(n,d) &= 2^d - 2 + 2^d - 2 + d(k-1) + (k-1)\left[1-\left(\frac{5}{8}\right)^d\right] \\ &= n + \left(\frac{n}{d} - 1\right)\left[1-\left(\frac{5}{8}\right)^d\right] + 2^{d+1} - d - 4, \end{aligned} \tag{16}$$

as claimed. ■

Recoded Binary Method (RBM) corresponds to the case when $d = 1$ in Recoded M-ary Method (RMM). The average number of multiplications required by RBM is found as

$$T^{\text{avg}}_{\text{RBM}}(n) = \frac{11}{8}(n-1). \tag{17}$$

The average number of multiplications required by RBM is tabulated for $n = 4, 8, 16, \ldots, 65536$ in Table 2. This table also contains the values of d^*, which minimize the average number of multiplications required by Recoded M-ary Method, and $\lceil T^{\text{avg}}_{\text{RBM}}(n) \rceil$ and $\lceil T^{\text{avg}}_{\text{RMM}}(n, d^*) \rceil$.

4 EXAMPLES

Given $n = 8$ and $E = (122)_{10} = (01111010)_2$, we show step by step the computation of $M^E \pmod N$ using Binary Method, M-ary Method, Recoded Binary Method, and Recoded M-ary Method. In the following we summarize the operations performed by these methods to compute $M^{122} \pmod N$ for each value of the loop parameter i.

Binary Method

The initial value of $C = 1$ since $E_{n-1} = E_7 = 0$.

i	E_i	Step 2a	Step 2b
7	0	1	1
6	1	$1 * 1 = 1$	$1 * M = M$
5	1	$M * M = M^2$	$M^2 * M = M^3$
4	1	$M^3 * M^3 = M^6$	$M^6 * M = M^7$
3	1	$M^7 * M^7 = M^{14}$	$M^{14} * M = M^{15}$
2	0	$M^{15} * M^{15} = M^{30}$	M^{30}
1	1	$M^{30} * M^{30} = M^{60}$	$M^{60} * M = M^{61}$
0	0	$M^{61} * M^{61} = M^{122}$	M^{122}

The total number of multiplications performed is $7 + 5 = 12$.

M-ary Method

From Table 2 we see that $d^* = 2$ minimizes the maximum as well as the average number of multiplications when $n = 8$. We have $d = 2$ and $k = 4$. First, X_j for $j = 0$, 1, 2, 3 is computed.

j	X_j
0	1
1	M
2	$M * M = M^2$
3	$M^2 * M = M^3$

This requires 2 multiplications. At Step 4, we assign $C = X_{F^{(3)}} = X_1 = M$.

i	$E_{2i+1}E_{2i}$	$F^{(i)}$	Step 5a	Step 5b
3	01	1	M	M
2	11	3	$M * M = M^2$	
			$M^2 * M^2 = M^4$	$M^4 * X_3 = M^7$
1	10	2	$M^7 * M^7 = M^{14}$	
			$M^{14} * M^{14} = M^{28}$	$M^{28} * X_2 = M^{30}$
0	10	2	$M^{30} * M^{30} = M^{60}$	
			$M^{60} * M^{60} = M^{120}$	$M^{120} * X_2 = M^{122}$

Thus, $2 + 6 + 3 = 11$ multiplications are performed by M-ary Method.

Recoded Binary Method

RBM starts with computation of Y_j for $j = 0, 1, -1$.

j	Y_j
0	1
1	M
-1	M^{-1}

The above is achieved using only assignment operations, since M^{-1} is supplied with M. At Step 5, we recode $E = 01111010$ *as* $D = 1000\bar{1}010$. The initial value of $C = Y_{F^{(7)}} = Y_1 = M$.

i	E_i	D_i	$F^{(i)}$	Step 8a	Step 8b
7	0	1	1	M	M
6	1	0	0	$M * M = M^2$	M^2
5	1	0	0	$M^2 * M^2 = M^4$	M^4
4	1	0	0	$M^4 * M^4 = M^8$	M^8
3	1	$\bar{1}$	-1	$M^8 * M^8 = M^{16}$	$M^{16} * Y_{-1} = M^{15}$
2	0	0	0	$M^{15} * M^{15} = M^{30}$	M^{30}
1	1	1	1	$M^{30} * M^{30} = M^{60}$	$M^{60} * Y_1 = M^{61}$
0	0	0	0	$M^{61} * M^{61} = M^{122}$	M^{122}

The total number of multiplications performed by RBM method is $7 + 2 = 9$.

Recoded M-ary Method

When $n = 8$, the optimum value of d for RMM is equal to 1, as it can be seen from Tables 1 and 2. RBM will yield the minimum number of multiplications for $n \leq 16$. We, however, illustrate Recoded M-ary Method below by taking $d = 2$. The reader should realize the Recoded M-ary Method for a non-optimum value of d may require more multiplications than Recoded Binary Method. Taking $d = 2$, we find $k = 4$. First, Y_j for $j = 0, 1, 2, 3, -1, -2, -3$ is computed.

j	Y_j
0	1
1	M
2	$M * M = M^2$
3	$M^2 * M = M^3$
-1	M^{-1}
-2	$M^{-1} * M^{-1} = M^{-2}$
-3	$M^{-2} * M^{-1} = M^{-3}$

This requires 4 multiplications. At Step 5, we recode $E = 01111010$ *as* $D = 1000\bar{1}010$.

We then assign $C = Y_{F(3)} = Y_2 = M^2$.

i	$E_{2i+1}E_{2i}$	$D_{2i+1}D_{2i}$	$F^{(i)}$	Step 8a	Step 8b
3	01	10	2	M^2	M^2
2	11	00	0	$M^2 * M^2 = M^4$	
				$M^4 * M^4 = M^8$	M^8
1	10	$\bar{1}0$	-2	$M^8 * M^8 = M^{16}$	
				$M^{16} * M^{16} = M^{32}$	$M^{32} * Y_{-2} = M^{30}$
0	10	10	2	$M^{30} * M^{30} = M^{60}$	
				$M^{60} * M^{60} = M^{120}$	$M^{120} * Y_2 = M^{122}$

Thus, RMM requires $4 + 6 + 2 = 12$ multiplications to compute M^{122}, which is larger than that of RBM, as we expected.

5 CONCLUSIONS

As it can be seen from Tables 1 and 2, significant reductions in the maximum and the average number of multiplications can be achieved with the use of high-radix and bit recoding techniques. When $n \geq 1024$, M-ary Method provides more than 39% reduction in the maximum number of multiplications and more than 19% reduction in the average number of multiplications with respect to Binary Method. Also, Recoded Binary Method requires 17% fewer multiplications in the maximum case, and 8% fewer multiplications in the average case than Binary Method. RMM requires 25% fewer multiplications in the maximum case and 10% fewer multi-

Table 4 Comparison of methods for small E.

E	BM	RBM	PTM	Winner(s)
16	4	4	4	BM, RBM, PTM
17	5	5	5	BM, RBM, PTM
18	5	5	5	BM, RBM, PTM
19	6	6	6	BM, RBM, PTM
20	5	5	5	BM, RBM, PTM
21	6	6	6	BM, RBM, PTM
22	6	6	6	BM, RBM, PTM
23	7	6	6	RBM, PTM
24	5	6	5	BM, PTM
25	6	7	6	BM, PTM
26	6	7	6	BM, PTM
27	7	8	6	PTM
28	6	6	6	BM, RBM, PTM
29	7	7	7	BM, RBM, PTM
30	7	6	6	RBM, PTM
31	8	6	7	RBM
Average	6.00	6.00	5.75	PTM

plications in the average case than RBM, Recoded M-ary Method, however, requires approximately the same number of multiplications as M-ary Method. This is due to the fact that the bits of the recoded exponent scanned at d bits at a time may not contain as many as d contiguous zeros, therefore 1s and $\bar{1}$s will occur, which will require additional multiplications.

For small values of E (when $E \leq 10^5$, i.e., $n \leq 17$), these algorithms are inferior to the factor method (FM) and the power tree method (PTM) given by Knuth (see, Section 4.6.3 in [7]). According to Knuth, FM is better than BM on the average. The first case in which FM beats PTM is when $E = 19879 = 103 * 193$. For $E \leq 10^5$, PTM is 88,803 times better than FM; it ties 11,191 times and loses only 6 times. In Table 4 we compare BM, RBM, and PTM for $16 \leq E \leq 31$. As it can be seen from this table, PTM is the clear winner. It loses to RBM only once when $E = 31$. It is asserted in [7] that a minimum of 7 multiplications is required to compute x^{31}, as illustrated below.

$$
\begin{aligned}
&1: \quad x * x = x^2 \\
&2: \quad x^2 * x = x^3 \\
&3: \quad x^3 * x^2 = x^5 \\
&4: \quad x^5 * x^2 = x^7 \\
&5: \quad x^7 * x^7 = x^{14} \\
&6: \quad x^{14} * x^{14} = x^{28} \\
&7: \quad x^{28} * x^3 = x^{31}.
\end{aligned}
$$

If, however, we have access to x^{-1} without a cost, then the computation of x^{31} requires only 6 multiplications:

$$
\begin{aligned}
&1: \quad x * x = x^2 \\
&2: \quad x^2 * x^2 = x^4 \\
&3: \quad x^4 * x^4 = x^8 \\
&4: \quad x^8 * x^8 = x^{16} \\
&5: \quad x^{16} * x^{16} = x^{32} \\
&6: \quad x^{32} * x^{-1} = x^{31}.
\end{aligned}
$$

For the security of the RSA cryptosystem, it is required that the numbers N and E have several hundred decimal digits [11]. Thus, E may have as many as 1024 bits, i.e., $E \approx 2^{1024}$. The high-radix and bit recoding algorithms are quite efficient when E is large, as can be seen in Tables 1 and 2. They are also simple to implement and more suitable for RSA encryption/decryption. The implementations of the factor method and the optimum power tree method, on the other hand, require the factorization of E and the generation of the optimum tree, respectively. These operations are time consuming and will increase the overhead for the computation of (1) when E is large.

Several implementations of the RSA cryptosystem have been reported in the literature. Brickell proposed the use of the 2's complement of the exponent, if the number of 1s in the binary expansion of E is larger than $n/2$ [2]. Brickell's algorithm computes $(M^{-1})^{-E} \pmod N$, and requires at most $\frac{3}{2}(n-1)$ multiplications. General

issues regarding the implementation of the RSA cryptosystem, such as the generation of the keys, implementation of the hierarchical key structure, encryption and decryption, etc., have been discussed by Jung [6]. Jung also reports several successful implementations on different computers. Signed digit representation (also known as redundant binary representation) and the binary method were used in [17] to reduce the length of systolic arrays designed for the RSA algorithm. The algorithm in [16] utilizes the binary method and an $O(n)$ parallel multiplier given earlier by the same authors [3]. A straightforward parallelization of the exponentiation problem in a Galois field is given in [4]. This algorithm assumes that the powers M^{2^i} are precomputed. It then proceeds to compute M^E on a parallel/pipelined architecture with the topology of a binary tree. Quisquater and Couvreur proposed the use of the Chinese Remainder theorem to compute (1) whenever $N = pq$ [10], which works well for RSA decryption. Since the senders do not know the factoring of N, this method is not applicable for RSA encryption. By approximately halving the number of bits in the operands, this algorithm reduces the complexity of the multiplication operation itself, rather than the total number of multiplications required to compute $M^E \pmod N$. Another implementation of the binary method, which uses a partitioning approach to fit the numbers to the wordsize of the computer for fast modular multiplication, is given in [13].

References

[1] A. D. Booth. A signed binary multiplication technique. *Q. J. Mech. Appl. Math.*, **4**(2) 236–240, 1951. (Also reprinted in [14], pp. 100–104.)

[2] E. F. Brickell. A fast modular multiplication algorithm with application to two key cryptography. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology, Proceedings of Crypto 82*, pages 51–60. Plenum Press, 1982.

[3] I-Ngo Chen and R. Willoner. An $O(n)$ parallel multiplier with bit-sequential input and output. *IEEE Transactions on Computers*, **28**(10) 721–727, October 1979.

[4] B. W. Fam. A parallel/pipeline processor for fast exponentiation. In *Proceedings of the International Conference on Parallel Processing*, pages 316–318, August 1982.

[5] K. Hwang. *Computer Arithmetic, Principles, Architecture, and Design.* John Wiley and Sons, Inc., 1979.

[6] A. Jung. Implementing the RSA cryptosystem. *Computers & Security*, 4(4) 342–350, August 1987.

[7] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Addison-Wesley Publishing Co., 2nd edition, 1981.

[8] J. D. Lipson. *Elements of Algebra and Algebraic Computing.* Addison-Wesley Publishing Co., 1981.

[9] O. L. MacSorley. High-speed arithmetic in binary computers. *Proceedings of the IRE*, **49** 67–91, January 1961. (Also reprinted in [14], pp. 14–38.)

[10] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, **18**(21) 905–907, October 1982.

[11] R. L. Rivest. RSA chips (Past/Present/Future). In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology, Proceedings of EUROCRYPT 84*, Lecture Notes in Computer Science, No. 209, pages 159–165. Springer-Verlag, 1984.

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

[13] A. Selby and C. Mitchell. Algorithms for software implementations of RSA. *IEE Proceedings, Part E: Computers and Digital Techniques*, **136**(6) 167–170, May 1989.

[14] E. E. Swartzlander, editor. *Computer Arithmetic*, volume I and II. IEEE Computer Society Press, 1990.

[15] S. Waser and M. J. Flynn. *Introduction to Arithmetic for Digital System Designers.* Holt, Rinehart and Winston, CBS College Publishing, 1982.

[16] R. Willoner and I-Ngo Chen. An algorithm for modular exponentiation. In *Proceedings, 5th Symposium on Computer Arithmetic*, pages 135–138, Ann Arbor, Michigan, May 18–19 1981. IEEE Computer Society Press.

[17] C. N. Zhang, H. L. Martin, and D. Y. Y. Yun. Parallel algorithms and systolic arrays designs for RSA cryptosystem. In K. Bromley, S. Y. Kung, and E. Swartzlander, editors, *Proceedings of the International Conference on Systolic Arrays*, pages 341–350, San Diego, California, May 25–27, 1988. IEEE Computer Society Press.