Pergamon

Computers Math. Applic. Vol. 30, No. 10, pp. 17–24, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00153-0

Analysis of Sliding Window Techniques for Exponentiation

C. K. Koç
Department of Electrical & Computer Engineering
Oregon State University, Corvallis, OR 97331, U.S.A.

(Received October 1993; accepted June 1995)

Abstract—The m-ary method for computing x^E partitions the bits of the integer E into words of constant length, and then performs as many multiplications as there are nonzero words. Variable length partitioning strategies have been suggested to reduce the number of nonzero words, and thus, the total number of multiplications. Algorithms for exponentiation using such partitioning strategies are termed sliding window techniques. In this paper, we give algorithmic descriptions of two recently proposed sliding window techniques, and calculate the average number of multiplications by modeling the partitioning process as a Markov chain. We tabulate the optimal values of the partitioning parameters, and show that the sliding window algorithms require up to 8% fewer multiplications than the m-ary method.

Keywords—Analysis of algorithms, Exponentiation, Binary method, m-ary method, Markov chain.

1. INTRODUCTION

The computation of x^E for a positive integer E is required in many important applications in computer science and engineering. The well-known binary method computes x^E using at most $2(n-1)$ multiplications, where n is the number of bits in the binary expansion of E. A summary of the algorithms for computing x^E can be be found in [1]. Some of these algorithms, e.g., the power tree and the factor methods, are applicable only when the exponent is small. Recent applications in cryptography, for example, the RSA algorithm [2], the ElGamal signature scheme [3], and the digital signature standard (DSS) of the National Institute for Standards and Technology [4], require the computation of $x^E \pmod M$ for a very large value of E, (usually $n = \log_2 E \geq 512$). The binary method is very suitable for these applications, requiring $1.5(n-1)$ multiplications on the average. A generalization of the binary method, the m-ary method, is given by Knuth [1]. When m is a power of 2, the implementation of the m-ary method is rather simple, since x^E can be computed by partitioning the bits of the exponent expressed in binary. With a proper choice of $m = 2^d$ for each n, the m-ary method requires fewer multiplications than the binary method [1,5].

Let $E = (E_{n-1}E_{n-2}\cdots E_1E_0)$ be the binary expansion of the exponent. This representation of E is partitioned into k words of length d, such that $kd = n$. The exponent is padded with at most $d-1$ zeros, if d does not divide n. We define

$$F_i = (E_{id+d-1}E_{id+d-2}\cdots E_{id}) = \sum_{j=0}^{d-1} E_{id+j}2^j$$

This research was supported in part by RSA Data Security, Inc., Redwood City, CA.

such that $0 \leq F_i \leq 2^d - 1$ and $E = \sum_{i=0}^{k-1} F_i 2^{id}$. The m-ary method first computes the values of x^w for $w = 2, 3, \ldots, 2^d - 1$. The exponent E is then 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 current nonzero word.

The m-ary Method
Input: x, E.
Output: $y = x^E$.
1. Compute and store x^w for all $w = 2, 3, 4, \ldots, 2^d - 1$.
2. Decompose E into d-bit words F_i for $i = 0, 1, 2, \ldots, k - 1$.
3. $y := x^{F_{k-1}}$
4. **for** $i = k - 2$ **downto** 0
  4a. $y := y^{2^d}$
  4b. **if** $F_i \neq 0$ **then** $y := y \cdot x^{F_i}$
5. **return** y

Step 1 of the m-ary method requires $2^d - 2$ preprocessing multiplications. The number of multiplication operations in Step 4a is equal to $(k-1)d = n - d$. We perform a multiplication in Step 4b if $F_i \neq 0$. Since $2^d - 1$ out of 2^d values of F_i are nonzero, the average number of multiplications required in Step 4b is $(k-1)(1 - 2^{-d})$. Thus, we find the average number of multiplications as

$$T(n, d) = 2^d - 2 + n - d + \left(\frac{n}{d} - 1\right)\left(1 - 2^{-d}\right). \tag{1}$$

The average number of multiplications for the binary method can be found simply by substituting $d = 1$ in (1), which gives $T = 1.5(n-1)$. Also note that there exists an optimal $d = d^*$ for each n such that $T(n, d)$ is minimized. The optimal values of d can be found by enumeration [1,5].

2. SLIDING WINDOW TECHNIQUES

The m-ary method decomposes the bits of the exponent into d-bit words. The probability of a word of length d being zero is 2^{-d}, assuming that the zero and one bits are produced with equal probability. In Step 4b of the m-ary method, we skip a multiplication whenever the current word is equal to zero. Thus, as d grows larger, the probability that we have to perform a multiplication operation in Step 4a becomes larger. However, the total number of multiplications as given by (1) increases as d decreases. The sliding window algorithms provide a compromise by allowing zero and nonzero words of variable-length; this strategy aims to increase the average number of zero words, while using relatively large values of d.

A sliding window exponentiation algorithm first decomposes E into zero and nonzero words (*windows*) F_i of length $L(F_i)$. The number of windows k may not be equal to n/d. In general, it is also not required that the length of the windows be equal. We take d to be the length of the longest window, i.e., $d = \max(L(F_i))$ for $i = 0, 1, \ldots, k - 1$. Furthermore, if F_i is a nonzero window, then the least significant bit of F_i must be equal to 1. This is because we partition the exponent starting from the least significant bit, and there is no point in starting a nonzero window with a zero bit. Consequently, the number of preprocessing multiplications (Step 1) are halved, since x^w are computed for odd w only.

The Sliding Window Method
Input: x, E.
Output: $y = x^E$.
1. Compute and store x^w for all $w = 3, 5, 7, \ldots, 2^d - 1$.
2. Decompose E into zero and nonzero windows F_i of length $L(F_i)$
for $i = 0, 1, 2, \ldots, k - 1$.
3. $y := x^{F_{k-1}}$

4. for $i = k - 2$ **downto** 0
  4a. $y := y^{2^{L(F_i)}}$
  4b. if $F_i \neq 0$ **then** $y := y \cdot x^{F_i}$
5. return y

We will analyze two sliding window partitioning strategies which have been proposed in [1,6]. These methods differ in that the length of a nonzero window can be either constant ($= d$), or it can be variable (however, $\leq d$). In the following sections, we give algorithmic descriptions of these two partitioning strategies, and calculate the average number of multiplications by modeling the partitioning process as a Markov chain.

3. CONSTANT LENGTH NONZERO WINDOWS

The constant length nonzero window (CLNW) partitioning algorithm scans the bits of the exponent from the least significant to the most significant. At any step, the algorithm is either forming a zero window (ZW) or a nonzero window (NW). The algorithm is described below:

ZW: Check the incoming single bit: if it is a 0, then stay in ZW; else go to NW.

NW: Stay in NW until all d bits are collected. Then check the incoming single bit: if it is a 0, then go to ZW; else go to NW.

Notice that while in NW, we distinguish between staying in NW and going to NW. The former means that we continue to form the same nonzero window, while the latter implies the beginning of a new nonzero window. The CLNW partitioning strategy produces zero windows of arbitrary length, and nonzero windows of length d. Two adjacent zero windows are necessarily concatenated, while two nonzero windows may be adjacent. For example, for $d = 3$, we partition $E = 3665 = (111001010001)$ as

$$E = \underline{111}\ 00\ \underline{101}\ 0\ \underline{001}.$$

The CLNW sliding window algorithm first performs the preprocessing multiplications and obtains x^w for $w = 3, 5, 7$. Starting with $y = x^{F_4} = x^7$, it proceeds to compute x^{3665} as follows:

i	F_i	$L(F_i)$	Step 4a	Step 4b
3	00	2	$(x^7)^4 = x^{28}$	x^{28}
2	101	3	$(x^{28})^8 = x^{224}$	$x^{224} \cdot x^5 = x^{229}$
1	0	1	$(x^{229})^2 = x^{458}$	x^{458}
0	001	3	$(x^{458})^8 = x^{3664}$	$x^{3664} \cdot x^1 = x^{3665}$

In order to compute the average number of nonzero windows, which represents the number of multiplications required in Step 4b, we model the partitioning process using a Markov chain. An n-bit binary number E uniformly distributed in the range $[0, 2^n - 1]$ can be viewed as a random process that generates one bit at a time. Each bit assumes a value of zero or one with equal probability, and there is no dependency between any two bits, i.e., $\mathcal{P}(E_i = 0) = \mathcal{P}(E_i = 1) = 1/2$ for $0 \leq i \leq n-1$. State variable S of the Markov chain is equal to 0 when zero windows are being formed (ZW), and S is equal to the length of the current nonzero window when nonzero windows are being formed (NW). Thus, we have $S = 0, 1, 2, 3, \ldots, d$. The probability that state $S = j$ succeeds state $S = i$ is denoted by $\mathcal{P}_{ij}$. Since $\mathcal{P}(E_i = 0) = \mathcal{P}(E_i = 1) = 1/2$, we have

$$\mathcal{P}_{00} = \mathcal{P}_{01} = \frac{1}{2}.$$

Furthermore, once the first bit is equal to 1, we collect $d-1$ more bits to obtain a nonzero window of length d, which implies

$$\mathcal{P}_{i,i+1} = 1, \qquad \text{for } 1 \leq i \leq d-1.$$

After all d-bits are collected, depending on the value of E_i, the next state is either $S = 0$ (when $E_i = 0$) or $S = 1$ (when $E_i = 1$), i.e.,

$$\mathcal{P}_{d0} = \mathcal{P}_{d1} = \frac{1}{2}.$$

All the other $\mathcal{P}_{ij}$s are zero. The state table of the Markov chain is given in Table 1.

Table 1. The CLNW state table.

S	S^+	$\mathcal{P}$	S^+	$\mathcal{P}$
0	0	1/2	1	1/2
1	2	1		
2	3	1		
3	4	1		
⋮	⋮	⋮		
$d-1$	d	1		
d	0	1/2	1	1/2

The one-step transition probability matrix for $d = 5$ is given as

$$\mathcal{P} = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1/2 & 1/2 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Let C_1 be the average number of nonzero windows after all n bits have been received. This can be found by counting the number of transitions from state 0 to state 1. Let $\mathcal{P}^{(n)}$ denote the n-step transition probability matrix, which is simply the n^{th} power of the matrix $\mathcal{P}$. We define the matrix $\mathcal{Q}^{(n)}$ as the sum

$$\mathcal{Q}^{(n)} = \sum_{i=1}^{n} P^{(i)} = \sum_{i=1}^{n} P^i.$$

Thus, after n iterations, the average number of transitions from state 0 to state 1 is found as

$$C_1 = \mathcal{Q}_{01}^{(n)}. \tag{2}$$

Note that the number of preprocessing multiplications is given by $(2^d - 2)/2 = 2^{d-1} - 1$, and the number of squaring operations is equal to $n - d$. Thus, the average number of multiplications required by this sliding window technique is found as

$$T_1(n, d) = 2^{d-1} - 1 + n - d + C_1 - 1. \tag{3}$$

Given the integers n and d, we can easily compute T_1 by first forming the $(d + 1) \times (d + 1)$ matrix $\mathcal{P}$, and then computing C_1 using (2). In Table 2, we tabulate the average number of multiplications for the m-ary and the CLNW sliding window methods. The column for the m-ary method contains the optimal values d^* for each n. As expected, there exists an optimal value of d for each n for the CLNW sliding window algorithm as well. These optimal values are also included in the table. The last column of the table contains the percentage difference in the average number of multiplications. The CLNW partitioning strategy reduces the average number of multiplications by 3–7% for $128 \leq n \leq 2048$. The overhead of the partitioning is negligible; the number of bit operations required to obtain the partitioning is $O(n)$.

Table 2. The m-ary versus the CLNW methods.

	m-ary		CLNW		$(T - T_1)/T$
n	d^*	T	d^*	T_1	%
128	4	168	4	156	7.14
256	4	326	5	308	5.52
384	5	483	5	458	5.18
512	5	636	5	607	4.56
640	5	789	6	756	4.18
768	5	941	6	903	4.04
896	5	1094	6	1049	4.11
1024	5	1247	6	1195	4.17
1152	6	1397	6	1341	4.01
1280	6	1546	6	1488	3.75
1408	6	1695	6	1634	3.60
1536	6	1844	6	1780	3.47
1664	6	1993	6	1927	3.31
1792	6	2142	7	2072	3.27
1920	6	2291	7	2216	3.27
2048	6	2440	7	2360	3.28

4. VARIABLE LENGTH NONZERO WINDOWS

The CLNW partitioning strategy starts a nonzero window when a 1 is encountered. Although the incoming $d - 1$ bits may all be zero, the algorithm continues to append them to the current nonzero window. For example, for $d = 3$, the exponent $E = (111001010001)$ was partitioned as

$$E = \underline{111}\ 00\ \underline{101}\ 0\ \underline{001}.$$

However, if we allow variable length nonzero windows, we can partition this number as

$$E = \underline{111}\ 00\ \underline{101}\ 000\ \underline{1}.$$

We will show that this strategy further decreases the average number of nonzero windows. The variable length nonzero window (VLNW) partitioning strategy requires that during the formation of a nonzero window (NW), we switch to ZW when the remaining bits are all zero. The VLNW partitioning strategy has two integer parameters:

d: maximum nonzero window length,
q: minimum number of zeros required to switch to ZW.

The algorithm proceeds as follows:

ZW: Check the incoming single bit: if it is zero then stay in ZW; else go to NW.

NW: Check the incoming q bits: if they are all zero then go to ZW; else stay in NW. Let $d = 1 + kq + r$ where $1 < r \leq q$. Stay in NW until $1 + kq$ bits are received. At the last step, the number of incoming bits will be equal to r. If these r bits are all zero, then go to ZW; else stay in NW. After all d bits are collected, check the incoming single bit: if it is zero, then go to ZW; else go to NW.

The VLNW partitioning produces nonzero windows which start with a 1 and end with a 1. Two nonzero windows may be adjacent; however, the one in the least significant position will necessarily have d bits. Two zero windows will not be adjacent since they will concatenated. For example, let $d = 5$ and $q = 2$, then $5 = 1 + 1 \cdot 2 + 2$, thus $k = 1$ and $r = 2$. The following illustrates the partitioning of a long exponent according to these parameters:

$$\underline{101}\ 0\ \underline{11101}\ 00\ \underline{101}\ \underline{10111}\ 000000\ \underline{1}\ 00\ \underline{111}\ 000\ \underline{1011}.$$

Also, let $d = 10$ and $q = 4$, which implies $k = 2$ and $r = 1$. A partitioning example is illustrated below:

$$\underline{1011011}\ 0000\ \underline{11}\ 0000\ \underline{1111110101}\ 00\ \underline{11110111}\ 0000\ \underline{11011}.$$

In order to compute the average number of multiplications, we model the partitioning process as a Markov chain. The state variable S takes the values $0, 1, 2, \ldots, k, k+1, k+2$. When $S = 0$, we are forming a zero window. The formation of a nonzero window starts with $S = 1$. Assuming the exponent bits are produced with equal probability, we conclude that

$$\mathcal{P}_{00} = \mathcal{P}_{01} = \frac{1}{2}.$$

In state 1, we check the incoming q bits. If they are all zero (with probability 2^{-q}), then the next state is 0; otherwise (with probability $1 - 2^{-q}$), the next state is 2. We proceed in this fashion until we reach state $k+1$. Thus,

$$\left.\begin{aligned} \mathcal{P}_{i,0} &= 2^{-q} \\ \mathcal{P}_{i,i+1} &= 1 - 2^{-q} \end{aligned}\right\} \qquad \text{for } i = 1, 2, \ldots, k.$$

In step $k+1$, we check the remaining r bits. If they are all zero (with probability 2^{-r}), the next state is 0; otherwise (with probability $1 - 2^{-r}$), the next state is $k+2$. This implies

$$\begin{aligned} \mathcal{P}_{k+1,0} &= 2^{-r}, \\ \mathcal{P}_{k+1,k+2} &= 1 - 2^{-r}. \end{aligned}$$

Finally, after all d bits are collected, we check the incoming single bit. If it is zero, the next state is 0; otherwise the next state is 1. Thus, we have

$$\mathcal{P}_{k+2,0} = \mathcal{P}_{k+1,1} = \frac{1}{2}.$$

The state table of the Markov chain is given in Table 3.

Table 3. The VLNW state table.

S	S^+	$\mathcal{P}$	S^+	$\mathcal{P}$
0	0	1/2	1	1/2
1	0	2^{-q}	2	$1-2^{-q}$
2	0	2^{-q}	3	$1-2^{-q}$
3	0	2^{-q}	4	$1-2^{-q}$
⋮	⋮	⋮	⋮	⋮
k	0	2^{-q}	$k+1$	$1-2^{-q}$
$k+1$	0	2^{-r}	$k+2$	$1-2^{-r}$
$k+2$	0	1/2	1	1/2

For example, the transition probability matrix for $d = 12$ and $q = 3$ (thus, $k = 3$ and $r = 2$) is given as

$$\mathcal{P} = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 1/8 & 0 & 7/8 & 0 & 0 & 0 \\ 1/8 & 0 & 0 & 7/8 & 0 & 0 \\ 1/8 & 0 & 0 & 0 & 7/8 & 0 \\ 1/4 & 0 & 0 & 0 & 0 & 3/4 \\ 1/2 & 1/2 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The Markov chain for the VLNW partitioning is slightly different from that of the CLNW partitioning in the sense that, after a single iteration of the chain, we may receive 1, q, or r bits. Thus,

if we allow the chain to iterate n times, then we may collect more than n bits. The number of bits collected is determined by the average behavior of the Markov chain. Let $\mathcal{P}^{(s)}$ be the s-step transition probability matrix, and $\mathcal{Q}^{(s)}$ be the sum of the i-step transition probability matrices for $i = 1, 2, \ldots, s$. After s iterations, the number of bits collected is equal to

$$n' = \mathcal{Q}_{00}^{(s)} + \mathcal{Q}_{01}^{(s)} + \mathcal{Q}_{k+2,0}^{(s)} + \mathcal{Q}_{k+2,1}^{(s)} + q\left[\sum_{i=1}^{k} \mathcal{Q}_{i,i+1}^{(s)} + \mathcal{Q}_{i,0}^{(s)}\right] + r\left[\mathcal{Q}_{k+1,k+2}^{(s)} + \mathcal{Q}_{k+1,0}^{(s)}\right]. \quad (4)$$

The average number of nonzero windows is given by

$$C_2 = \mathcal{Q}_{01}^{(s)} + \mathcal{Q}_{k+2,1}^{(s)}, \quad (5)$$

and the average number of bits (zero or one) within the nonzero windows is found as

$$C_3 = \mathcal{Q}_{01}^{(s)} + \mathcal{Q}_{k+2,1}^{(s)} + q\left[\sum_{i=1}^{k} \mathcal{Q}_{i,i+1}^{(s)}\right] + r\mathcal{Q}_{k+1,k+2}^{(s)}, \quad (6)$$

which gives the average nonzero window length as C_3/C_2. Thus, we find the average number of multiplications as

$$T_2(n', d, q) = 2^{d-1} - 1 + n' - \frac{C_3}{C_2} + C_2 - 1. \quad (7)$$

In order to compute the average values of T_2, we first pick the parameters d and q. Then, using the expression (4), we find the smallest s such that $n' \geq n$. This value of s is used in expressions (5) and (6) to obtain an average value for T_2. Our experiments have indicated that the best values of q are between 1 and 3 for $128 \leq n' \leq 2048$ and $4 \leq d \leq 8$. In Table 4, we tabulate the minimal values of T/n (the m-ary), and T_2/n' (VLNW) together with the optimal values of d for $n = 128, 256, \ldots, 2048$ and $q = 1, 2, 3$. The VLNW algorithm requires 5–8% fewer multiplications than the m-ary method. In Figure 1, we plot the average number of multiplications for the m-ary and the sliding window algorithms as a function of $n = 128, 256, \ldots, 2048$.

Table 4. The m-ary versus the VLNW methods.

	m-ary		VLNW				$(T_2 - T)/T_2$
				T_2/n'			for q^*
n	d^*	T/n	d^*	$q=1$	$q=2$	$q=3$	%
128	4	1.305	4	1.204	1.203	1.228	7.82
256	4	1.270	4	1.184	1.185	1.212	6.77
384	5	1.256	5	1.172	1.183	1.170	6.85
512	5	1.241	5	1.163	1.175	1.162	6.37
640	5	1.231	5	1.158	1.170	1.157	6.01
768	5	1.225	5	1.155	1.167	1.154	5.80
896	5	1.221	6	1.152	1.150	1.161	5.81
1024	5	1.217	6	1.148	1.146	1.157	5.83
1152	6	1.212	6	1.145	1.143	1.154	5.69
1280	6	1.207	6	1.142	1.140	1.152	5.55
1408	6	1.203	6	1.140	1.138	1.150	5.40
1536	6	1.200	6	1.138	1.136	1.148	5.33
1664	6	1.197	6	1.137	1.135	1.147	5.18
1792	6	1.195	6	1.136	1.134	1.146	5.10
1920	6	1.193	6	1.135	1.133	1.145	5.03
2048	6	1.191	6	1.134	1.132	1.144	4.95

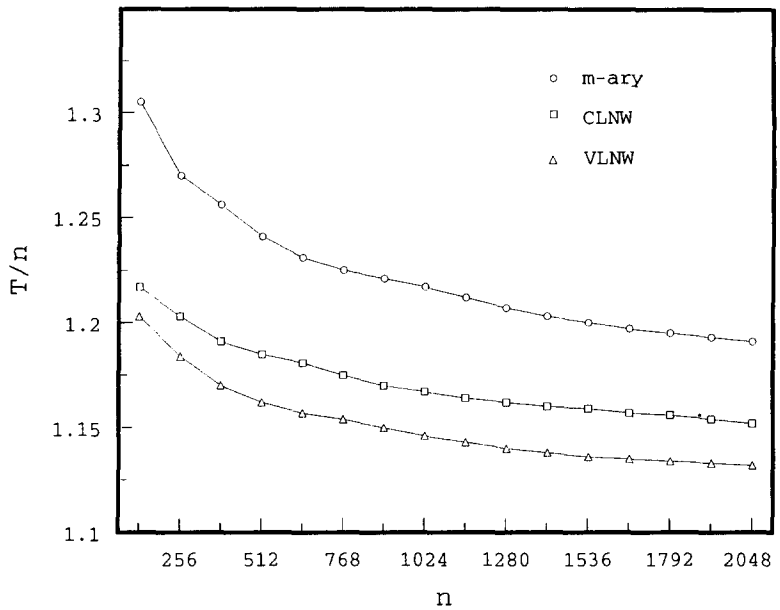

Figure 1. The m-ary versus the sliding window algorithms.

5. CONCLUSIONS

The sliding window algorithms are easy to program, introducing negligible overhead. The reduction in terms of the number of multiplications is notable, for example, for $n = 512$, the m-ary method requires 636 multiplications whereas the CLNW and VLNW sliding window algorithms require 607 and 595 multiplications, respectively. The reduction in total time can be significant when the multiplication operation requires a considerable amount of time. Such applications are found in cryptography where one needs to compute $x^E \pmod{M}$ for very large values of E and M. The sliding window algorithms have been proposed in this context, especially for speeding up the RSA encryption and decryption operations. Other related exponentiation heuristics can be found in [7–9].

REFERENCES

1. D.E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Volume 2, Second edition, Addison-Wesley, Reading, MA, (1981).
2. R.L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* **21** (2), 120–126 (February 1978).
3. T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* **31** (4), 469–472 (July 1985).
4. National Institute for Standards and Technology, Digital signature standard (DSS), *Federal Register* **56**, 169 (August 1991).
5. Ç.K. Koç, High-radix and bit recoding techniques for modular exponentiation, *International Journal of Computer Mathematics* **40** (3/4), 139–156 (1991).
6. J. Bos and M. Coster, Addition chain heuristics, In *Advances in Cryptology—CRYPTO 89, Proceedings*, Lecture Notes in Computer Science, No. 435, (Edited by G. Brassard), pp. 400–407, Springer-Verlag, New York, (1989).
7. Y. Yacobi, Exponentiating faster with addition chains, *Advances in Cryptology—EUROCRYPT 90*, Lecture Notes in Computer Science, No. 473, (Edited by I.B. Damgård), pp. 222–229, Springer-Verlag, New York, (1990).
8. E.F. Brickell, D.M. Gordon, K.S. McCurley and D.B. Wilson, Fast exponentiation with precomputation, *Advances in Cryptology—EUROCRYPT 92*, Lecture Notes in Computer Science, No. 658, (Edited by R.A. Rueppel), pp. 200–207, Springer-Verlag, New York, (1992).
9. Ö. Eğecioğlu and Ç.K. Koç, Exponentiation using canonical recoding, *Theoretical Computer Science* **129** (2), 407–417 (1994).