Journal of the Egyptian Mathematical Society
Volume (26) - Issue (2) - 2018
ISSN:1110-256X
DOI:10.21608/joems.2018.2540.1008

SPEEDING UP MULTI- EXPONENTIATION ALGORITHM ON A MULTICORE SYSTEM

Khaled A. Fathy[1], Hazem M. Bahig[2], and Mohamed S. Farag[1]

[1] Department of Mathematics, Faculty of Science, Al-Azhar University, Cairo, Egypt.
[2]Computer Science Division, Department of Mathematics, Faculty of Science
Ain Shams University, Cairo, Egypt.
hbahig@sci.asu.edu.eg

Received 5/1/2018 Revised Accepted 8/2/2018

Abstract

A public key cryptosystem is a basic tool to protect data security. Most public key cryptosystem schemes include time consuming operations such as the modular multi exponentiation. To address this problem, a new parallel algorithm for the modular multi exponentiation is introduced. The proposed algorithm is based on parallelizing the binary method. The experimental study on a multicore system shows that the running time of the proposed algorithm is smaller than the previous parallel algorithm in the cases of large data sizes under different number of processors. The percentage of improvement is up to 55% compared with the previous algorithm.

keywords: public key cryptosystem, modular multi exponentiation, parallel algorithm.

MSC: 11Y16, 68W10 and 11T71.

1 Introduction

Given n messages as integer numbers $M_j, 0 \leq j < n$, and exponents in the binary representation $E_j = (e_{j(l-1)}e_{j(l-2)} \dots e_{j1}e_{j0})_2$, where l is the maximum length of the binary representation of the exponents. The modular multi exponentiation problem is to compute $\prod_{j=0}^{n-1} M_j^{E_j} \bmod N$, where N is a large modulus. The modular multi exponentiation is a fundamental operation in the area of the cryptography. There are several cryptographic schemes and algorithms that depend on multi exponentiation such as the RSA cryptosystem [1] and the Diffie−Hellman scheme [2] for the single exponentiation. The DSA algorithm [3], the Schnorr identification and signature scheme [4] and the BrickellMcCurley scheme [5] for double exponentiation. The ElGamal digital signature scheme [6] is an example for triple exponentiation.

There are two directions for speeding up the computation of the modular multi exponentiation. The first is to design a fast modular multiplication method. Several papers were presented to speed up the multiplication. For example, Yen and Laih [7] presented a multiplication method to speeding up the multiplication. The second direction is to design a fast exponentiation algorithm using several methods such as the binary method, 2^w-ary method and sliding window method [8–10]. Dimitrov et al. [11] presented new algorithms for the computation of modular exponentiations. These algorithms are based on complex arithmetic representations. The applicability of the proposed algorithms depends on the assumption that the inverse elements can be precomputed in advance. Moller [9, 12] presented two algorithms based on the sliding windows method. For speeding up the computation of modular multi exponentiation, Wu et al. [13] introduced a fast algorithm combining the complement recoding method and the minimal weight BSD representation technique. Sun et al. analyzed the computational efficiency

of Wu et al.'s algorithm [13] by modeling it as a Markov chain [14]. Yen et al. [15] presented a series of algorithms for the computation of multi exponentiation. These algorithms are based on the binary greatest common divisor algorithm. The proposed algorithms have the capability to evaluate double or multi exponentiation with variable base numbers and exponents. In 2016 Wu et al. [16] proposed several batch multi exponentiation algorithms for the computation of the exponentiation operations. The mechanism of these algorithms is by allowing a large number of multi-base exponentiations to be processed in batch.

Different parallel works were presented for the computation of the multi exponentiation problem. Chiou [17] proposed a parallel algorithm depending on the concurrent computation of the squaring and multiplication of the binary method. Chang and Lou [8] proposed two efficient parallel methods for the computation of the modular multi exponentiation which are based on the linear array model. Also Chang and Lou [18] generalized Chiou's parallel algorithm [17] to compute the modular multi exponentiation. Recently, Borges et al. [19] proposed a parallel algorithm to compute the modular multi exponentiation, the proposed algorithm is the generalization of Lara et al. work [20].

In this work, we present a fast parallel algorithm to compute the modular multi exponentiation based on the binary method and the parallel multiplication algorithm. Then we study the efficiency of the proposed algorithm on a multicore system.

The rest of the paper is organized as an introduction and four sections. In Section 2, we discuss the parallel algorithm presented by Borges et al. [19]. The introduced parallel algorithm for the computation of the modular multi exponentiation is given in Section 3. In Section 4, we study the proposed algorithm practically on a multi-core system to measure the running time and the speedup complexity. Also the comparison and discussion with the recent parallel algorithm is given in the same section. The conclusion of the presented work is given Section 5.

2 Borges et al. Algorithm

In this section, we will describe and analyze the recent parallel algorithm for the modular multi exponentiation presented by Borges et al. [19] which will be named BLP algorithm. The section will be organized in two subsections. In the first subsection, we will describe the main idea and the complete steps of the algorithm. In the second subsection, we will describe our observations on this algorithm .

2.1 The Algorithm

Borges et al. [19] introduced two parallel algorithms to compute the modular multi exponentiation. In the first algorithm, the binary representation of the exponents, as a two dimensional array, is split among the available processors into equal length partitions. Let $\{r_0 = 0, r_1, r_2, \ldots, r_k\}$, denote the points of partition. Using the points of partition, each processor computes $a_i = (\prod_{j=0}^{n-1} M_j^{E_{j,i}})^{2^{r_i - 1}} \mod N$, where $E_{j,i} = (e_{jr_i-1}\ e_{jr_i-2} \ldots e_{jr_{i-1}})$ is the ith partition of E_j. Finally, the algorithm computes the multiplication of the results of each processor. The second algorithm (BLP) was designed to make the load balance between the processors equally to maximize the speedup. The algorithm is based on computing the optimal number of processors, k, and the optimal points of partition, $\{r_0 r_2, \ldots, r_k\}$, to distribute the work equally among the k processors. The value of k is computed as follows [19].

$$k = [\gamma + \frac{1}{2}] \tag{1}$$

where [ ] represents the nearest integer, $\gamma = -\log_{x_2}(\frac{l(1-x_2)}{d} + 1)$, $x_2 = \frac{\frac{n}{2}M}{1+\frac{n}{2}M}$, M is the cost of the modular multiplication operation and equal to 1.14 from practical view, and d is a chosen value (greater than or equal 1) to avoid the collision of r_i and r_{i+1} after truncation.

To compute the points of partitions, they defined the value $\alpha = \frac{l}{1-x_2^k}$, so that the optimal points of partitions will be defined as follows [19]

$$r_i = \alpha(1 - x_2^i) \text{ for } 1 \leq i \leq k \text{ and } r_0 = 0 \quad (2)$$

The complete pseudocode for the algorithm is as follows.

Algorithm: BLP

Input: Messages as integer numbers $M_j, 0 \leq j < n$, exponents in the binary representation $E_j = (e_{j(l-1)}e_{j(l-2)} \dots e_{j1}e_{j0})_2$ and modulus N, where l is the maximum length.

Output: An integer $C = \prod_{j=0}^{n-1} M_j^{E_j} \bmod N$.

Begin

1. Compute the optimal number of processors k as in Eq(1).
2. Compute the positions of partition $r_i \; \forall \; 0 \leq i \leq k$ as in Eq(2).
3. Compute $a_i = (\prod_{j=0}^{n-1} M_j^{E_{j,i}})^{2^{r_i-1}} \bmod N$, by processor $p_i, \forall \; 1 \leq i \leq k$.
4. Multiply the k values of a_i, and assign it to $C = \prod_{i=1}^{k} a_i \bmod N$

End

2.2 Analysis and Observations

In this subsection, we will give two observations on Eq(1) and Eq(2) which are used to compute the optimal number of processors suggested by Borges et al. and to determine the positions of partition. In order to give our observations, we first calculate the optimal number of processors suggested by Borges et al. on different values of n and l where $n = 4,\ 8,\ 12,\ 16$ and 20, and $l = 2^{10},\ 2^{11},\ 2^{12},\ 2^{13},\ 2^{14},\ 2^{15},\ 2^{16}$ and 2^{17}. Table 1 shows the optimal number of processors at which the maximum speed will occur according to Borges et al.

n \ l	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
4	14	16	18	20	22	24	26	28
8	23	27	30	34	37	41	44	48
12	31	36	41	46	51	57	62	67
16	38	45	52	58	65	72	78	85
20	45	53	61	70	78	86	94	102

Table 1: The optimal number of processors suggested by Borges et al.

From Table 1 we observed the following:

1. The optimal number of processors increases with the increasing of the values of n and l. In the cases of large values of n and l, the optimal number of processors may not be available in real machines.
2. The formulas depend on the value of M which is the cost of the modular multiplication operation and this cost depends on the machine used. Also the reduction of the formulas depends on assuming that: the number of $1's$ in each column of the array of the binary representation of the exponents is $\frac{n}{2}$.

The question now is whether an algorithm can be designed to achieve the following goals:

1. The proposed algorithm does not depend on any parameters and assumptions.

2. The running time of the proposed algorithm is smaller than the running time of BLP algorithm at the optimal number of processors k.

3. The running time of the proposed algorithm is smaller than the running time of BLP algorithm at a fixed number of processors depending on the computing power of the machine. Because a large number of machines does not have the required number of processors k.

4. The behavior of the running time of the proposed algorithm is better than the behavior of the running time of BLP algorithm at different numbers of processors.

3 The Proposed Multi- Exponentiation Algorithm

In this section, we will introduce a parallel algorithm to perform the modular multi exponentiation that achieves the goals of the question asked in the previous section.

Given the messages as an array of integers $M_j, 0 \leq j < n$, and the binary representation of the exponents as a two dimensional array E. A row E_j represents the binary representation of the $(j+1)^{th}$ exponents, $0 \leq j < n$. The maximum length of the exponents will be denoted by l. First we will define an array G of l elements as follows: $g_i = \prod_{j=0}^{n-1} M_j \bmod N$ if $e_{ji} \neq 0$ for $0 \leq i < l$. The value of g_i for $0 \leq i < l$ can be considered as the value of the $(i+1)^{th}$ column of the array E. Now the modular multi exponentiation can be computed as $\prod_{i=0}^{l-1} g_i^{2^i} \bmod N$.

The main idea to design the proposed algorithm for the modular multi exponentiation in parallelism consists of two stages. The main idea of the first stage is an assignment of a processor for each column of the array of the binary representation of the exponents E to compute $g_i^{2^i} \bmod N$, $0 \leq i \leq l-1$. This goal can be achieved as follows. First, by initializing the value of g_i with $\prod_{j=0}^{n-1} M_j \bmod N$ if $e_{ji} \neq 0$ and then by squaring g_i $i-$times. In the second stage, we apply the parallel multiplication algorithm [21] to multiply g_i, $0 \leq i \leq l-1$ and store the result in g_0.

The main differences between the proposed algorithm and BLP algorithm are as follows:

1. In the proposed algorithm, we do not compute the optimal number of processors but we use l processors theoretically. In the experimental study we simulate the algorithm on p (available number of processors).

2. In the proposed algorithm, each processor is assigned to a column in the binary representation array to compute the value of g_i where $0 \leq i < l$. In BLP algorithm, each processor is assigned to a two dimensional block of the array of the binary representation to compute the value of a_i where $0 \leq i < k$.

3. The final multiplication step in BLP algorithm is executed sequentially while in the proposed algorithm the multiplication step is executed in parallel.

The complete pseudocode for the fast parallel modular multi exponentiation algorithm, FMME, is as follows: step 1 represents the first stage of the proposed algorithm while steps 2-5 represent the second stage.

Algorithm: FMME

Input: Messages as an array of integers M of n elements, exponents in the binary representation as a two dimensional array E of n rows and l columns, and modulus N.

Output: Modular multi exponentiation $\prod_{j=0}^{n-1} M_j^{E_j} \bmod N$

Begin

1. for $i = 0$ to $l-1$ do parallel

$g_i = 1$

for $j = 0$ to $n - 1$ do

if $e_{ji} \neq 0$ then

$g_i = (g_i * M_j) \bmod N$

for $j = 1$ to i do

$g_i = {g_i}^2 \bmod N$

2. $m = \lceil \frac{l}{\log l} \rceil$

3. while $m > \lceil \log l \rceil$ do

$h = \lceil \log m \rceil$

for $i = 0$ to $\frac{m}{h} - 1$ do parallel

if $i \neq \frac{m}{h} - 1$ then

for $j = 1$ to $h - 1$ do

$g_{ih} = (g_{ih} * g_{ih+j}) \bmod N$

else

for $j = 1$ to $m - 1 - ih$ do

$g_{ih} = (g_{ih} * g_{ih+j}) \bmod N$

$g_i = g_{ih}$

$m = \lceil \frac{m}{h} \rceil$

4. for $i = 1$ to $m - 1$ do

$g_0 = (g_0 * g_i) \bmod N$

5. return g_0

End

The running time complexity of the proposed algorithm can be analyzed as follows. In the first step there are $l-1$ squares and n multiplications will be performed. In the steps $2 - 5$, there are approximately t_{FPM} [21] multiplications. The time of

the proposed algorithm is $T = (l-1) \times S + (n + t_{\text{FPM}}) \times M$ where S and M is the running time required for the modular squaring and multiplication respectively. The storage complexity of the proposed algorithm is $O(l)$.

4 Experimental Results

In this section, we present an implementation for FMME and BLP algorithms and then we compare between them according to running time and speedup.

The implementation is based on a multicore system from Google Cloud systems. The name of machine is n1-highmem-64 and consists of 64 cores and 416 GB memory. The algorithms were implemented using used C++ programming language and we use the library OpenMP to allow the parallelization. Also we used GNU Multiple Precision (GMP) library for arbitrary precision arithmetic.

In the experiment, we implemented the algorithms on different sizes and numbers of the exponent E. N in our experiments is equal to $2^l + 1$. E and M were chosen as random numbers of length l. The sizes of the exponents, l, are from 2^{10} to 2^{17}. The numbers of the exponents, n, are 4, 8, 12, 16 and 20. We implemented the algorithms on a sample of 20 random numbers.

The analysis of the experimental results will be described in three subsections. In the first subsection, we will compare FMME and BLP algorithms using the number of processors suggested by Borges et al [19]. In the second subsection, we will compare FMME and BLP algorithms using a fixed number of processors. The speed up of FMME and BLP algorithms will be studied in the third subsection.

4.1 The Comparison at k Processors

In order to analyze the performance of the FMME algorithm, we compared FMME and BLP algorithms using the optimal number of processors suggested by Borges et al. and shown in Table 1. In the cases of $n = 4$, 8 and 12, we give the results using the required number of processors. Because the required number of processors in the cases of $n = 16$ and 20 is greater than 64 processors so we give the results using a simulation on a machine with 64 cores. An important note is about the running time of the multiplication of g_i, $0 \leq i < l$ generated from the first step of FMME algorithm. In some cases this running time using the sequential algorithm for the multiplication is smaller than the time using the parallel multiplication algorithm [21]. The reason is that the modular squaring performed in the first step sometimes gives a lot of small values for some g_i. The results of the running time of the two algorithms using different inputs are shown in Table 2, where the running time is measured in seconds.

n / l	4		8		12		16		20	
	FMME	BLP	FMME	BLP	FMME	BLP	FMME	BLP	FMME	BLP
2^{10}	0.006	0.001	0.008	0.002	0.008	0.001	0.011	0.003	0.019	0.005
2^{11}	0.017	0.004	0.018	0.006	0.023	0.007	0.03	0.008	0.034	0.011
2^{12}	0.053	0.023	0.045	0.02	0.053	0.037	0.058	0.033	0.056	0.032
2^{13}	0.18	0.17	0.12	0.19	0.13	0.20	0.13	0.15	0.13	0.23
2^{14}	0.90	0.87	0.76	1.03	0.79	1.04	0.58	1.18	0.61	1.22
2^{15}	4.47	5	3.52	5.85	3.55	5.06	2.97	6.59	3.18	6.65
2^{16}	19.93	25.69	15.31	26.71	18.74	24.83	17.15	37.96	17.81	35.89
2^{17}	93.94	130.20	85.26	126.36	101.40	215.01	97.91	216.40	103.46	204.77

Table 2: The time comparison between FMME and BLP algorithms

From Table 2, we observed the following:

1. BLP algorithm is better than FMME algorithm in the cases of $l = 2^{10},\ 2^{11}$ and 2^{12} for all values of n. The percentage of the improvement of BLP algorithm is $30\% - 83\%$ but the values of running times in these cases are very small. Also BLP algorithm is better than FMME algorithm in the cases of $n = 4$ for $l = 2^{13}$ and 2^{14}. In this cases, the percentage of the improvement of BLP algorithm is approximately 6%.

2. Our proposed algorithm FMME is better than BLP algorithm in the cases of $n = 4$ for $l = 2^{15},\ 2^{16}$ and 2^{17}. Also our proposed algorithm FMME is better than BLP algorithm in all cases of $l = 2^{13},\ 2^{14},\ 2^{15},\ 2^{16}$ and 2^{17} for all values of n except $n = 4$. For example, in the case of $n = 8$ and $l = 2^{16}$ the given results are using 44 processors and the running time of FMME algorithm is 15.31 seconds while the running time of BLP algorithm is 26.71. The percentage of the improvement of the FMME algorithm is $10\% - 55\%$.

4.2 The Comparison at a Fixed Number of Processors

In the cases of a large number of exponents n and a large size of the exponents l, the required number of processors k suggested by Borges et al. will be a large number of processors. Therefore the comparison between FMME and BLP algorithms using a fixed number of processors, independent of n and l, is important to give an impression of the behavior of the running time of FMME and BLP algorithms.

An important note is that we ignored the running time in the cases of $l = 2^{10},\ 2^{11},\ 2^{12}$ and 2^{13}. Because all the values of the running time in these cases are smaller than 1 second. Table 3 represents the running time of both algorithms at $p = 16$ and 32.

		l							
		2^{14}		2^{15}		2^{16}		2^{17}	
n	p	FMME	BLP	FMME	BLP	FMME	BLP	FMME	BLP
4	16	1.06	1.02	5.24	6.30	28.62	34.63	148.43	189.35
	32	0.63	0.62	3.70	3.70	14.72	20.32	75.78	110.49
8	16	1.08	1.49	5.92	9.33	31.93	43.54	181.86	229.44
	32	0.80	1.10	3.92	6.68	16.04	31.12	89.96	155.56
12	16	1.50	2.29	7.78	12.79	43.16	73.82	233.28	421.06
	32	1.04	1.34	4.17	7.90	21.82	44.22	117.79	307.46
16	16	1.36	2.48	6.87	12.74	38.68	76.39	218.09	518.57
	32	0.82	1.63	3.65	9.71	19.34	48.79	108.49	283.50
20	16	1.44	2.37	7.40	15.51	40.56	88.61	229.14	565.21
	32	0.82	1.59	3.80	9.02	20.39	50.53	114.20	337.69

Table 3: The time comparison between FMME and BLP algorithms at $p = 16$ and 32

From Table 3, we observed the following:

1. BLP algorithm is better than FMME algorithm in the case of $n = 4$ and $l = 2^{14}$ at $p = 16$ and $p = 32$. The percentage of the improvement of BLP algorithm is 4%.

2. FMME algorithm is better than BLP algorithm in all cases of $n = 8,\ 12,\ 16$ and 20 for all values of l at $p = 16$ and 32 and in the case of $n = 4$ and $l = 2^{15},\ 2^{16}$ and 2^{17} at $p = 16$ and $p = 32$. We also observed that the percentage of improvement of FMME algorithm increases with the increasing of the length of the exponents i.e with the increasing

of l. For example, the percentage of improvement of FMME algorithm in the case of $n = 12$ and $l = 2^{14}$ is 34% for $p = 16$ while the percentage of improvement of FMME algorithm in the case of $n = 12$ and $l = 2^{17}$ is 44% for $p = 16$.

4.3 The Speed up Measurement

The speedup of a parallel algorithm is defined to be the ratio of the running time of the sequential version to the running time using p processors [22].

n \ l	2^{14}	2^{15}	2^{16}	2^{17}
4	2.64	15.43	70.26	414.63
8	3.46	25.41	135.82	746.79
12	4.87	28.95	192.27	864.77
16	6.37	35.95	247.04	1095.55
20	7.66	43.55	248.92	1382.91

Table 4: The sequential time

Table 4 gives the running time of the sequential algorithm [19] which will be used to compute the speed up of FMME and BLP algorithms. The implementation of the algorithm is based on the same platform and input data.

To measure the speed up of FMME and BLP algorithms we computed the speed up of both algorithms at different number of processors and different data sizes. Table 5 gives the speed up comparison between FMME and BLP algorithms at $p = 16,\ 32$ and at the optimal number of processors k suggested by Borges et al.

		l							
		2^{14}		2^{15}		2^{16}		2^{17}	
n	p	FMME	BLP	FMME	BLP	FMME	BLP	FMME	BLP
4	16	2.49	2.59	2.94	2.45	2.45	2.03	2.79	2.19
	32	4.19	4.26	4.17	4.17	4.77	3.46	5.47	3.75
	k	2.93	3.03	3.45	3.09	3.53	2.73	4.41	3.18
8	16	3.20	2.32	4.29	2.72	4.25	3.12	4.11	3.25
	32	4.33	3.15	6.48	3.80	8.47	4.36	8.30	4.80
	k	4.55	3.36	7.22	4.34	8.87	5.08	8.76	5.91
12	16	3.25	2.13	3.72	2.26	4.45	2.60	3.71	2.05
	32	4.68	3.63	6.94	3.66	8.81	4.35	7.34	2.81
	k	6.16	4.68	8.15	5.72	10.26	7.74	8.53	4.02
16	16	4.68	2.57	5.23	2.82	6.39	3.23	5.02	2.11
	32	7.77	3.91	9.85	3.70	12.77	5.06	10.10	3.86
	k	10.98	5.40	12.10	5.46	14.40	6.51	11.19	5.06
20	16	5.32	3.23	5.89	2.81	6.14	2.81	6.04	2.45
	32	9.34	4.82	11.46	4.83	12.21	4.93	12.11	4.10
	k	12.56	6.28	13.69	6.55	13.98	6.94	13.37	6.75

Table 5: The speed up comparison between FMME and BLP algorithms at $p = 16,\ 32$ and $p = k$

From Table 5, we observed that the speed up of FMME algorithm is better than the speed up of BLP algorithm except

in the case at $n = 4$ and $l = 2^{14}$. For example, when $n = 16,\ l = 2^{16}$ and $p = k$, the speedup of FMME algorithm is 14.40 while the speedup of BLP algorithm is 6.51.

The minimum value of speedup for BLP algorithm is 2.03 while the maximum value is 7.74 using different number of processors $p = 16, 32$ and k. On the other side, the minimum value of speedup for FMME algorithm is 2.45 while the maximum value is 14.4.

5 Conclusions

Modular multi exponentiation is a fundamental operation in many public key cryptosystems. In this paper, we presented a new parallel algorithm for the modular multi exponentiation which is based on parallelizing the binary method and the multiplication of large numbers.

The experimental results show that the efficiency of the proposed algorithm is better than the recent previous parallel algorithm on a multicore system in the cases of large data sizes. The percentage of improvement of the running time of the proposed algorithm is up to 55% compared with the previous algorithm. Moreover, the maximum value of the speedup for the recent previous algorithm is 7.74 while the proposed algorithm has maximum speedup is 14.4.

References

[1] R. Rivest, A. Shamir and I. Adleman, A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, **21**, (1978), 120-126.

[2] W. Diffie and M.E. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory **IT-22 (6)**, (1976), 644-654.

[3] National Institute of Standards and Technology, A proposed federal information processing standard for digital signature standard (DSS), Federal Register **56 (169)**, (1991), 42980-42982.

[4] C.P. Schnorr, Efficient identification and signature for smart cards, in: Advances in Cryptology-proceedings of Crypto89, SpringerVerlag, New York, (1990), 239-252.

[5] E.F. Brickell and K.S. McCurley, Interactive identification and digital signature, AT&T Technical Journal (Nov.), (1991), 73-86.

[6] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inf. Theory **31 (4)**, (1985), 469-472.

[7] S.M. Yen and C.S. Laih, Common-multiplicated multiplication and its applications to public key cryptography, Elect. Letters, **29(17)**, (1993), 1583-1584.

[8] C. Chang and D. Lou, Parallel computation of the multi exponentiation for cryptosystems, Int. J. Comp. Math., **63:1-2**, (1997), 6-29.

[9] B. Moller, Improved techniques for fast exponentiation, LNCS, **2587**, (2003), 298-312.

[10] S.M. Yen, C.S. Laih and A.K. Lenstra, Multi exponentiation, IEE Proc. Computers and Digital Techiques, **141**, (1994), 325-326.

[11] V.S. Dimitrov, G.A. Jullien and W.C. Miller, Complexity and fast algorithms for multiexponentiations, IEEE Trans. Comput., **49(2)**, (2000), 141-147.

[12] B. Moller, Algorithms for multi exponentiation, Selected Areas in Cryptography, LNCS, **2259**, (2001), 165-180.

[13] C. Wu, D. Lou, J. Lai and T. Chang, Fast modular multi exponentiation using modified complex arithmetic, App. Math. and Comp., **186**, (2007), 1065-1074.

[14] D. Sun, J. Huai, J. Sun and J. Zhang, Computational efficiency analysis of Wu et al.'s fast modular multi exponentiation algorithm, App. Math. and Comp., **190**, (2007), 1848-1854.

[15] S.M. Yen, C. Chen and S. Moon, Multi exponentiation algorithm based on binary GCD computation and its application to side-channel countermeasure, J. Cryptogr. Eng., **2**, (2012), 99-110.

[16] Q. Wu, Y. Sun, B. Qin, J. Hu, W. Liu, J. Liu and Y. Ding, Batch Public Key Cryptosystem with batch multi-exponentiation, Fut. Gen. Comp. Sys., **62**, (2016), 196-204.

[17] C.W. Chiou, Parallel implementation of the RSA public-key cryptosystem, Int. J. Comp. Math., **48**, (1993), 153-155.

[18] C. Chang and D. Lou, Fast parallel computation of multi exponentiation for public key cryptosystems, Proc. of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, (2003), 955-958.

[19] F. Borges, P. Lara, and R. Portugal, Parallel algorithms for modular multi-exponentiation, Journal of App. Math. and Comp., **292**, (2017), 406-416.

[20] P. Lara, F. Borges, R. Portugal and N. Nadia Parallel modular exponentiation using load balancing without precomputation. J. Comp. and Sys. Sci., **78**, (2012), 575-582.

[21] Khaled A. Fathy, Hazem M. Bahig and A. A. Ragab, A Fast Parallel Modular Exponentiation Algorithm, Arab. J. Sci. Eng., In press.

[22] S. Akl, Parallel Computation: Models and Methods. Prentice Hall, Upper Saddle River, New Jersey, 1997.