Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 871091, 10 pages
doi:10.1155/2012/871091

Research Article

An Optimization of Tree Topology Based Parallel Cryptography

Masumeh Damrudi and Norafida Ithnin

Department of Computer System and Communication, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, Skudai, 81310 Johor Bahru, Malaysia

Correspondence should be addressed to Masumeh Damrudi, m.damrudi@gmail.com

Received 26 April 2012; Accepted 2 August 2012

Academic Editor: Wanquan Liu

Public key cryptography has become of vital importance regarding the rapid development of wireless technologies. The RSA is one of the most important algorithms for secure communications in public-key cryptosystems. Since the RSA is expensive in terms of computational task which is modular exponentiation, parallel processing and architecture is a reasonable solution to speedup RSA operations. In this paper, taking into account pipelining and optimization, we improve throughput and efficiency of the TRSA method, a parallel architecture solution for RSA security based on tree topology. The optimization and pipelining of the tree based architecture increases its efficiency and throughput. The experimental results demonstrate that these pipelined and optimized approaches outperform the main TRSA.

1. Introduction

Using wireless communication has made most of today's systems vulnerable. Employing appropriate measures, which provide confidentiality, integrity, authenticity, and availability for all messages in presence of adversaries, can reduce threats on applications exploiting wireless communications such as WSN. Asymmetric algorithms (called public key) can provide user authentication and be useful in key distribution and management. Public key encryption allows two parties to communicate secretly, even if all communications between them are monitored. Furthermore, it allows enormous flexibility, which in example is essential for an online merchant where an online merchant processing credit card orders from multiple purchasers. It is more convenient for the receiver to store a single private key rather than to share and manage different secret keys.

Once asymmetric cryptographies are based on number theory, they are expensive in terms of their mathematical calculations [1, 2], which need large amount of energy resources

and sufficient amount of memory for large keys [3]. Benefiting from software solutions for cryptography leads to flexibility and ease of use and upgrade whilst it do not provide enough speed and are less secure than hardware solutions. Software issues can be easily monitored and on the other hand they consume a lot of resources. Besides, it is difficult to transfer them among different operating systems whilst hardware methods need fewer computer resources and have special chips to accelerate the process [4]. As a real-world example, the security problem in sensor networks is more challenging due to the resource limited nodes. In such low resource devices, a solution based on parallel architecture becomes more appropriate due to the smaller occupied area by the architecture [5]. Basically, energy efficiency and battery life time play a major role in the lifetime of such applications [6].

The main reason of using topological interconnection of processor elements in terms of parallel architecture is to create a powerful computer or processing element for specific objectives. Parallel architecture is combined of nodes and these interconnection networks. Depending on the algorithm being deployed on the architecture, either of pipelining or optimization and in some cases both of them are applicable to the solutions. While the pipelined approach increases the throughput by processing multiple PEs simultaneously, the optimization improves efficiency by eliminating the redundant PEs.

The main contribution of this paper lies on using parallel pipelined of Tree topology for RSA cryptography and issuing an optimization to this work. The pipelining, and moreover, optimization improve the results drastically.

The rest of this paper is organized as follows. Firstly, we summarize the related works in the next section. Then, the pipelined TRSA is explained. Section 4 provides the optimization of TRSA. Afterwards, the simulation results are discussed in Section 5. Finally, the conclusion of the work is drawn.

2. Related Works

We have summarized existing parallel approaches on cryptographic algorithms [7]. In addition, a parallel cryptography method using RSA is proposed in [8], which uses tree topology as its base infrastructure. As far as the authors' knowledge, from the topology point of view, there is no significant parallel approach on RSA algorithm other than TRSA. The well-known parallel methods using software and hardware solutions on RSA are in [5, 9–21]. More details on these approaches are presented in [7]. Among these parallel approaches, the best value in terms of seconds is 3.23 ms while applying 1024 bit key length [5]. The greatest key length is 3072 [12], and the best speedup is 10.9 using 2560 key length [13]. The evaluation metric in these methods is speedup or time for most of the cases, and hardware implementations have employed Montgomery's algorithm to perform modular multiplication or exponentiation.

These recent approaches have not discussed the time complexity or the order of the algorithms which are inseparable from parallel processing. To the best of the authors' knowledge, the only existing discussion on time complexity is the known CRT [22], Montgomery [23], and the binary [22] which is an accepted method, that are based on the number of multiplications. Herewith, to analyze the proposed approach, we deal with the time complexity of optimization and pipelining of TRSA method.

3. Pipelined TRSA

TRSA is a new parallel approach on RSA using the tree interconnection network [8]. TRSA can be applied as a coprocessor for embedded systems such as wireless sensor nodes, which

require more speed in transferring confidential information. Enhancing TRSA, pipelining mechanism is employed to achieve higher throughput. Applying pipelining to the main solution, the latency between data that are being encrypted drastically decreases [5]. It is assumed that the base operation is one multiplication and one modulus, which is called MulMod in brief. Considering the encryption as $C_i = m_i^e \bmod n$ where i is the block number, in TRSA, computing C_i should be finished before the C_{i+1} starts to be computed. In this enhancement to the main approach, when the results of current operations are sent to the next level of tree, the computation for next ciphertext can be started in this level. In principle, a new operation can be initiated with this frequency. Even though previous ones are still in pipeline, and the ciphertext is not ready for prior plaintext so far.

The following definitions are used in the original algorithm.

n_p: The number of PEs of tree + 1

e_l: The exponentiation for the last leaf node's input

e_o: The exponentiation for other leaf nodes' input

p_i: ith processor element.

The RSA consists of two great prime numbers p and q, which compute modulus $n = pq$. The value $\varphi = (p-1)(q-1)$ is employed to determine e, where $e < n$ and $gcd(e, \varphi) = 0$. Consequently, d has the following relation:

$$d = e - 1 \bmod \varphi. \tag{3.1}$$

Let e_o, e_l, A, and B be computed as following:

$$e_o = e \operatorname{div} n_p, \tag{3.2}$$

$$e_l = e_o + e \bmod n_p, \tag{3.3}$$

$$A = m^{e_o} \bmod n, \tag{3.4}$$

$$B = m^{e_l} \bmod n. \tag{3.5}$$

Figure 1(a) presents the schematic structure of a tree-based architecture using seven PEs. Figure 1(b) demonstrates inner structure of each PE.

The encryption/decryption solution in TRSA using seven PEs as tree topology is indicated in the following. Processor elements from p_0 to p_2 receive the value of A as their inputs, and p_3 will receive both A and B. These PEs will perform the following operations:

$$\begin{gathered} \forall i, \in \{i \geq 0, i \leq 2\}; \quad p_i \text{ computes } (A \times A) \bmod n, \\ p_3 \text{ computes } (A \times B) \bmod n. \end{gathered} \tag{3.6}$$

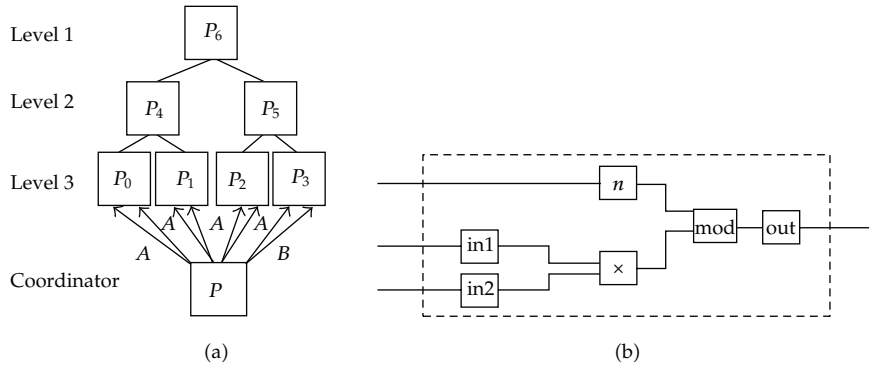

Figure 1: (a) Processor elements in tree architecture, (b) interior structure of each PE.

PEs p_0 to p_3 will send their results to their parents which are p_4 and p_5. The values D and E are the inputs of p_4 and p_5 that are computed as follows:

$$\begin{aligned} D &= (A \times A) \bmod n, \\ E &= (A \times B) \bmod n. \end{aligned} \tag{3.7}$$

In the second step, processor elements p_4 and p_5 will carry out the following operations to compute F and G:

$$\begin{aligned} &p_4 \text{ computes } F = (D \times D) \bmod n, \\ &p_5 \text{ computes } G = (D \times E) \bmod n. \end{aligned} \tag{3.8}$$

Then p_4 and p_5 will send their results to the root. The root will execute the following operation to generate the output of p_6 which is the encryption of m using RSA:

$$p_6 \text{ computes } (F \times G) \bmod n \tag{3.9}$$

The details of the operations are discussed in [8].

The pipeline phases are shown in Figure 2. It is shown that while the computation for C_1 is still in process, the computations for other messages have already been started.

The number of levels depends on the number of nodes in the tree. In this example, number of nodes is eight in which seven of them are forming the tree, and the other one is the coordinator. However, the number of nodes can be more or less but not less than four. Number of PEs should follow the tree rule which is $2^{\text{No. of levels}} - 1$, in addition a processor element is also added as the coordinator of first operation. Using more nodes, the more parallelization will be gained, nevertheless, overloading of more processor elements for smaller data and key length should be prevented. The coordinator operations will be decreased due to the increase in the number of PEs for the tree which leads to less execution time. In this approach, there is always a tradeoff between the number of PEs, the data size, and the key length.

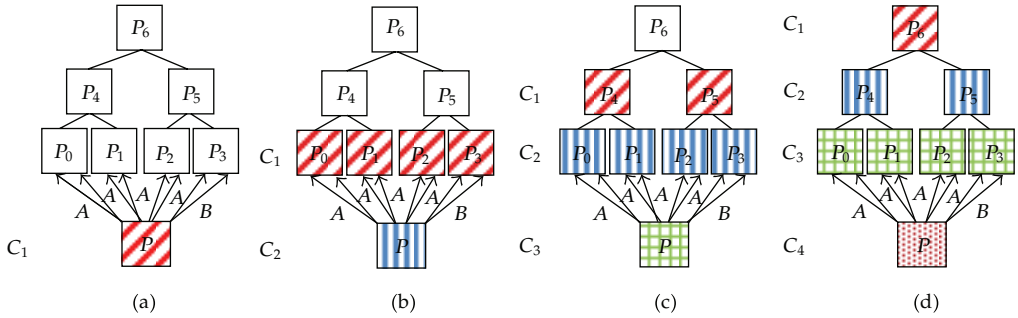

Figure 2: Steps of pipelining in TRSA.

3.1. Analysis of Pipelining

Once levels one, two, and three do just one MulMod operation, their execution times indicated by T_1, T_2, and T_3 are equal and we have

$$T_1 = T_2 = T_3. \tag{3.10}$$

The execution time of the operation in the coordinator depends on the computational power of the coordinator and the e_l. Although tree architecture used here is a homogenous architecture itself, this architecture is heterogeneous as a whole, which means that the coordinator processor element is different from other process elements. The PE known as coordinator must be more advanced than others. Assuming this and knowing that the computation in this PE is more complicated than others, the difference between the execution time of this processor element and others should be very small. As an applied example, the coordinator can be the CPU of the embedded system, and the tree part can be used as a coprocessor. In this case, the coprocessor is homogeneous. According to the above discussion, and using T_c as the execution time of coordinator, we can have

$$T_1 = T_2 = T_3 \cong T_c. \tag{3.11}$$

Using pipelining, as it is clarified in Figure 2, the throughput of TRSA with pipelining is about four-times of the throughput of original TRSA for a tree with seven nodes. In the pipelined version, when computation of C_1 is completed, calculation of C_5 will be started as a new block; whilst in the original TRSA, when C_1 is computed, calculation of C_2 will be started as the next block. Th is representing throughput in the subsequent equations:

$$\begin{aligned} T_1 = T_2 = T_3 &\cong T_c, \\ T_{\text{pipelined}} &= \frac{1}{4} T_{\text{original}}, \\ \text{Th}_{\text{pipelined}} &= 4\text{Th}_{\text{original}}. \end{aligned} \tag{3.12}$$

It should be considered that if the number of processor elements increase, the throughput of pipelining will be increased too. The relation of the throughput for

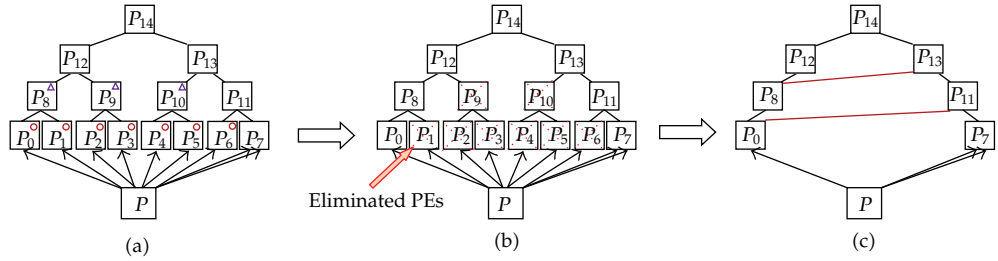

Figure 3: Optimization of TRSA using tree with 15 PEs.

the pipelined TRSA and the throughput for original TRSA where l represents the levels of the tree is

$$\mathrm{Th}_{\mathrm{pipelined}} = (l+1)\mathrm{Th}_{\mathrm{original}}. \tag{3.13}$$

In this approach, all PEs are busy all the time computing cipher texts while in the original TRSA, just one level of the processor elements was busy computing, and the others were idle. The pipelining mechanism provides an appropriate load balancing. In this example, having just four pipeline stages has led to a higher throughput, and more pipelining by employing a bigger tree architecture will achieve higher performance and more efficient encryption implementation.

4. Optimization of TRSA

The original TRSA method can be improved and optimized to increase efficiency as well as decrease area, which is one of the most important factors in energy and size limited systems like sensor nodes. In the original TRSA there are some nodes, which perform the same operation during operation and generate the same result. Knowing this, the results of some PEs can be employed instead of the results of some other PEs. These redundant PEs can be eliminated. Figure 3 illustrates the transition from original TRSA to the optimized TRSA. Considering four levels in a tree, the PEs which are doing the same operation are marked with the same shape in Figure 3(a). The nodes that can be eliminated from the tree to optimize the algorithm are omitted in Figure 3(c) and new connections are created.

Eliminating redundant PEs leads to a smaller area usage. By the increase in levels of the tree, the redundant PEs will be increased. If the levels of parallelizing are more than four, the number of eliminated PEs increases exponentially.

4.1. Analysis of Optimization

Efficiency of the parallel approaches [24–26] is measured using efficiency factor, which is $E = S/P$, where P and S are the number of processors and speedup, respectively. Assuming the number of PEs to be 15 as demonstrated in Figure 3, the efficiency of the algorithm will increase from $S/15$ to $S/7$ employing optimization. Taking into the general form,

the efficiency of original TRSA is $E = S/(2^l - 1)$ where l is the number of tree levels. Optimizing the TRSA, the efficiency of optimized TRSA becomes

$$E_{\text{op}} = \frac{S}{(2l-1)}. \tag{4.1}$$

The improvement of efficiency is obtained by

$$\frac{E_{\text{op}}}{E} = \frac{(2^l - 1)}{(2l-1)}. \tag{4.2}$$

The above equation shows that the efficiency is boosted exponentially, especially, when the number of levels increases.

The number of multiplications for accepted method, which is binary in the best case, is $k-1$ where k is the number of bits of the exponent [14, 22, 27]. The number of multiplications in the binary method for the worst and average cases are $2(k-1)$ and $1.5(k-1)$. Let l be the number of levels, the power of A and B in (3.4) and (3.5) will be divided to the number of PEs which is 2^l. Hence, $2(k-1-\log(2^l))$ multiplications will be done to compute A and B consequently. Thus the number of multiplications of coordinator for the worst case is

$$K' = 2\left(k - 1 - \log\left(2^l\right)\right) = 2(k-l-1). \tag{4.3}$$

The total number of multiplications for optimized TRSA is $2l-1$, thus the total multiplication count is $2(k-l-1)+2l-1$. While there are some operations which execute simultaneously in the parallel architectures, the actual number of multiplications is much less than the total number of multiplications. In this case, the total number of PEs in optimized TRSA is $2l-1$ where l is the level of tree and the number of multiplications is l due to the concurrency of PEs. Therefore, the total number of multiplications for encryption is decreased to $2(k-l-1)+l$.

Using the calculation method from [11], the number of multiplications is presented with $\eta(k,l)$ and in result the number of multiplications for the optimized TRSA in the worst case is

$$\eta(k,l) = 2(k-1) - l. \tag{4.4}$$

Table 1 compares the number of multiplications in the best, average, and worst case for binary, CRT, Montgomery, and optimized TRSA for both general form and a tree with 60 levels.

The number of multiplications of primitive RSA will be reduced using the optimized TRSA method compared to the CRT and Montgomery method. Although, the best case in the TRSA and the binary methods are the same, the average and worst cases of TRSA are improved. The level of this improvement depends on the number of MulMod blocks in the tree topology and the length for RSA cryptographic key in bits.

Table 1: Number of multiplications in binary, CRT, montgomery, and optimized TRSA.

Method	Best case		Average case		Worst case	
Binary	General form	$k = 1024$ $l = 60$	General form	$k = 1024$ $l = 10$	General form	$k = 1024$ $l = 60$
	$k - 1$	1023	$1.5(k - 1)$	1535	$2(k - 1)$	2046
TRSA	$k - 1$	1023	$1.5(k - 1) - 0.5l$	1505	$2(k - 1) - l$	1986
CRT			$3k^2/4 + k$	787456		
Montgomery			$2k^2 + k$	2098176		

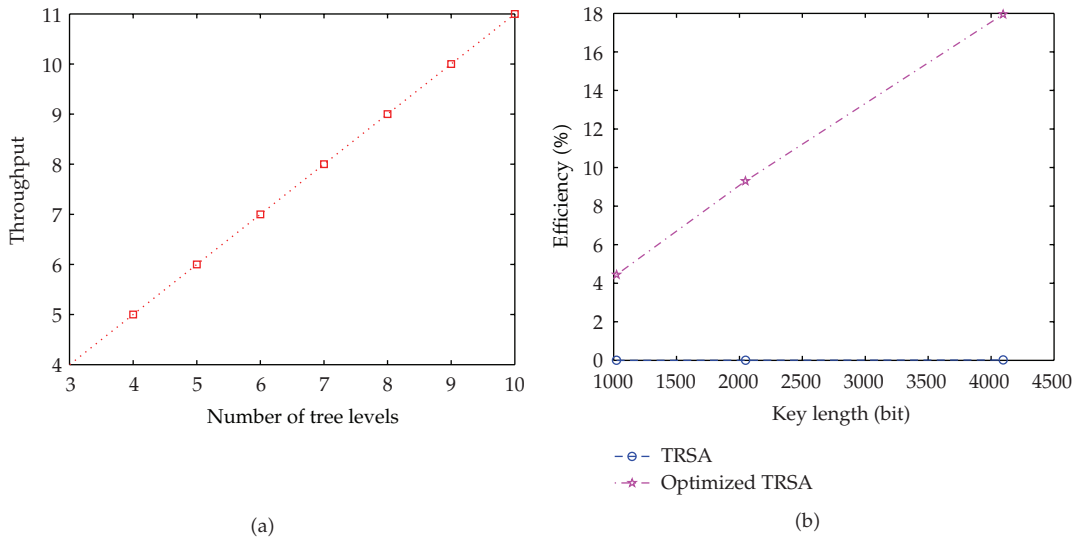

Figure 4: Optimization and pipelining of TRSA. (a) Throughput of original TRSA versus pipelined TRSA. (b) Efficiency of original TRSA versus Optimized TRSA.

5. Simulations, Results, and Discussion

Benefiting from pipelining technique and the optimization method, the variation of TRSA presented in this paper outperforms original TRSA. According to [8] and the results of time complexity that is based on the number of multiplications, original TRSA also outperforms the well-known existing approaches from the literature which are CRT, Montgomery, Binary, and the sequential approaches.

The advantages of pipelining in terms of throughput are presented in Figure 4(a). The throughput is $\mathrm{Th}_{\mathrm{pipelined}} = (l + 1)\mathrm{Th}_{\mathrm{original\,TRSA}}$. The advantages of optimization in terms of efficiency are presented in Figure 4(b).

Considering the number of levels for TRSA to be three, the throughput of pipelined TRSA is four-times of main TRSA. The more levels employed, the better throughput obtained.

The simulation results of optimization for the average of 100 iterations are utilized to draw Figure 4(b). The results are achieved using C++ language of Microsoft Visual Studio 2008, OpenMP and OpenSSL. The efficiency is $E = S/P$ where S is representing the execution time for TRSA and optimized TRSA, and the efficiency is multiplied by 100 to be in percent unit. When the number of PEs for original TRSA is seven, the number of PEs in Optimized TRSA becomes five. Just like the case in pipelining, the more levels of tree employed, the more efficiency obtained.

Figure 4(b) shows the efficiency of the original TRSA versus optimized TRSA for 1024, 2048, and 4096 bit key lengths using 64, 128, and 256 byte input file and 61, 29, and 15 levels, respectively. The optimized TRSA has preferred efficiency than original TRSA, which means that less area is required.

Selecting number of levels for a tree among others depends on the area and the speed which is desired for the security needs of the target system. The speed and number of PEs are always related to each other.

The optimized TRSA has more efficiency than original TRSA, which means that it requires less area than TRSA that leads to less hardware complexity. It is necessary to clarify that the two variations of the original TRSA represented in this paper are not in conflict with each other. Taking pipelining into account along with the optimization, the outcome will be optimum in terms of speed, throughput, and efficiency all at the same time as three significant parameters.

6. Conclusions

This paper has come up with a pipelining and optimization approach for original TRSA to achieve better results in RSA encryption using TRSA method. The efficiency of the original TRSA approach is improved using the optimization whilst the throughput is increased by conducting pipelining. The increase in the efficiency resulted from optimization of TRSA depends on the bits of key and the tree levels. Just like efficiency, the throughput also increases by the increase in the tree levels. The simulation results confirmed that having a greater key indicates the efficiency of optimized TRSA which becomes better than original TRSA. Applying the optimization, the number of PEs decreases, which results in less area usage that is extremely important in size and energy limited embedded systems.

Acknowledgment

The authors do not have any conflict of interests with the content of the paper.

References

[1] A. G. Marco, A. S. Martinez, and O. M. Bruno, "Fast, parallel and secure cryptography algorithm using Lorenz's attractor," *International Journal of Modern Physics C*, vol. 21, no. 3, pp. 365–382, 2010.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the Association for Computing Machinery*, vol. 21, no. 2, pp. 120–126, 1978.

[3] M. H. Ahmed et al., "Security for WSN based on elliptic curve cryptography," in *Proceedings of the 1st International Conference on Computer Networks and Information Technology (ICCNIT '11)*, Piscataway, NJ, USA, 2011.

[4] W. Zhang, W. Chen, J. Tang, P. Xu, Y. Li, and S. Li, "The development of a portable hard disk encryption/decryption system with a MEMS coded lock," *Sensors*, vol. 9, no. 11, pp. 9300–9331, 2009.

[5] G. Perin, D. G. Mesquita, F. L. Herrmann, and J. B. Martins, "Montgomery modular multiplication on reconfigurable hardware: fully systolic array vs parallel implementation," in *Proceedings of the 6th Southern Programmable Logic Conference (SPL '10)*, pp. 61–66, Ipojuca, Brazil, March 2010.

[6] R. Kayalvizhi, M. Vijayalakshmi, and V. Vaidehi, "Energy analysis of RSA and ELGAMAL algorithms for wireless sensor networks," in *Proceedings of the 8th WSEAS International Conference on Applied Electromagnetics, Wireless and Optical Communications (ELECTRO '10)*, N. Mastorakis et al., Ed., pp. 20–24, World Scientific and Engineering Academy and Society, Athens, Greece, March 2010.

[7] M. Damrudi and N. Ithnin, "State of the art practical parallel cryptographic approaches," *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 7, pp. 660–677, 2011.
[8] M. Damrudi and N. Ithnin, "Parallel RSA encryption based on tree architecture," *Journal of the Chinese Institute of Engineers*. In press
[9] W. Bielecki and D. Burak, *Parallelization Method of Encryption Algorithms*, Springer, New York, NY, USA, 2007.
[10] W. Fan, X. Chen, and X. Li, "Parallelization of RSA algorithm based on compute unified device architecture," in *Proceedings of the 9th International Conference on Grid and Cloud Computing (GCC '10)*, pp. 174–178, Nanjing, China, November 2010.
[11] P. Lara, F. Borges, R. Portugal, and N. Nedjah, "Parallel modular exponentiation using load balancing without precomputation," *Journal of Computer and System Sciences*, vol. 78, no. 2, pp. 575–582, 2012.
[12] Y. Li, Q. Liu, and T. Li, "Design and implementation of an improved RSA algorithm," in *Proceedings of the International Conference on E-Health Networking, Digital Ecosystems and Technologies (EDT '10)*, pp. 390–393, Kunming, China, April 2010.
[13] L. Qing, L. Yunfei, and H. Lin, "On the design and implementation of an efficient RSA variant," in *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE '10)*, pp. V3533–V3536, Kunming, China, August 2010.
[14] S. Sepahvandi, M. Hosseinzadeh, K. Navi, and A. Jalali, "An improved exponentiation algorithm for RSA cryptosystem," in *Proceedings of the International Conference on Research Challenges in Computer Science (ICRCCS '09)*, pp. 128–131, Shanghai, China, December 2009.
[15] T. Teerakanok and K. Kamolphiwong, "Accelerating asymmetric-key cryptography using Parallel-key Cryptographic Algorithm (PCA)," in *Proceedings of the 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON '09)*, pp. 812–815.
[16] M. Ciet et al., "Parallel FPGA implementation of RSA with residue number systems—can side-channel threats be avoided?" in *Proceedings of the IEEE International Symposium on Micro-NanoMechatronics and Human Science*, vol. 2, pp. 806–810, 2004.
[17] J. J. A. Fournier and S. Moore, "Hardware-software codesign of a vector co-processor for public key cryptography," in *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD '06)*, pp. 439–446, Dubrovnik, Croatia, September 2006.
[18] H. Jiang and G. Yang, "Resistant against power analysis for a fast parallel high-radix RSA algorithm," in *Proceedings of the International Conference on Electric Information and Control Engineering (ICEICE '11)*, pp. 1668–1671, Wuhan, China, April 2011.
[19] N. Nedjah and L. De MacEdo Mourelle, "High-performance SoC-based implementation of modular exponentiation using evolutionary addition chains for efficient cryptography," *Applied Soft Computing Journal*, vol. 11, no. 7, pp. 4302–4311, 2011.
[20] O. Nibouche, M. Nibouche, A. Bouridane, and A. Belatreche, "Fast architectures for FPGA-based implementation of RSA encryption algorithm," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 271–278, December 2004.
[21] Q. Liu, F. Ma, D. Tong, and X. Cheng, "A regular parallel RSA processor," in *Proceedings of the 47th Midwest Symposium on Circuits and Systems (MWSCAS '04)*, pp. III467–III470, July 2004.
[22] C. K. Koc, *High-Speed RSA Implementation*, RSA Data Security, Redwood, Ore, USA, 1994.
[23] C.-L. Wu, "Fast parallel montgomery binary exponentiation algorithm using canonical- signed-digit recoding Ttechnique," in *Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing*, Taipei, Taiwan, 2009.
[24] H. El-Rewini and M. Abd-El-Barr, *Advanced Computer Architecture and Parallel Processing*, John Wiley & Sons, New York, NY, USA, 2005.
[25] V. Kumar and V. N. Rao, "Parallel depth first search. II. Analysis," *International Journal of Parallel Programming*, vol. 16, no. 6, pp. 501–519, 1987.
[26] L. R. Scott, T. Clark, and B. Bagheri, *Scientific Parallel Computing*, Princeton University Press, Princeton, NJ, USA, 2005.
[27] D.-Z. Sun, Z.-F. Cao, and Y. Sun, "How to compute modular exponentiation with large operators based on the right-to-left binary algorithm," *Applied Mathematics and Computation*, vol. 176, no. 1, pp. 280–292, 2006.