Indian Journal of Science and Technology, *Vol 8(30), DOI:10.17485/ijst/2015/v8i30/85921, November 2015*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A Parallel Method for RSA Cryptosystem Utilizing Topological Architecture

Masumeh Damrudi[1]*, Kamal Jadidy Aval[1] and Norafida Ithnin[2]

[1]Department of Computer Science, Firoozkooh Branch, Islamic Azad University, Firoozkooh, Iran; m.damrudi@gmail.com
[2]Department of Computer System and Communication, Faculty of Computing, University Teknology Malaysia, Skudai - 81310, Johur Bahru, Malaysia

Abstract

Users of today's digital applications are increasing dramatically and there has been a rapid growth in the users of embedded wireless systems that usually transfer confidential information. These systems do expect robust, real time, and accurate performance, which bring Quality of Service (QoS) into mind. Parallel computing is one of the solutions to such issues to speed up the response time. On the other hand, cryptography is an important part and the first way of keeping information private. As a cryptography method, asymmetric encryption process needs massive mathematical operations, especially when a greater key is needed. This paper present a new parallel method for RSA cryptography based on mesh topology named MRSA. This method is applicable in embedded devices as a coprocessor and can act as a crypto-engine. The MRSA method is analyzed mathematically and compared to other methods. The result shows that the proposed method has fewer steps as well as multiplication operations to compute the encrypted value compared to the accepted method, which is the binary method.

Keywords: Cryptography, Mesh, Parallel, RSA, Topology

1. Introduction

Users of the digital world expect a certain behavior from applications and consumers of electronic services are subject to higher demands[1] Embedded systems use a weak CPU due to their processing needs, and intruders attack utilizing strong processing resources. Using a coprocessor to collaborate with CPU would be a suitable idea for these devices. These coprocessors may gain benefits of parallelism, which is an appropriate way of doing operations in a faster way. Cryptographic algorithms usually need massive operations; in result, speedup could be provided using a coprocessor which is benefiting from parallelism. RSA is an asymmetric cryptographic algorithm which is still being used in variety of applications, although; it has been used many years. Having a strong mathematical background, RSA is still secure using greater key lengths. Asymmetric keys solve the key distribution problem in symmetric cryptography. They are also used for non-repudiation, which helps to prove that only the sender who has the key could have sent the message.

Existing parallel approaches on cryptographic algorithms are summarized in[2]. Focusing on the RSA, there has been some pieces of research on performing it in parallel[3-17]. Recently interconnection network concept has been employed to provide a faster way of doing RSA cryptography. The TRSA and its optimization are parallel cryptographic approaches based on the RSA and tree interconnection network[18,19]. As far as we know, there is no other significant topology based cryptographic approach other than TRSA. Regardless of TRSA, the so called 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, which is the number of multiplications, are the well-known CRT[20],

*Author for correspondence

Montgomery[21], and the binary[20] where the latter is an accepted method.

In this paper, we define a new parallel cryptographic approach based on mesh interconnection network, which is a combination of parallel cryptographic algorithm and parallel architecture that is called MRSA. The order of this approach is compared to the existing approaches as well as TRSA in terms of number of multiplication. The rest of this paper is organized as follows. Primitive RSA is described in the second section. Afterwards, MRSA parallel encryption algorithm is presented, formulated and analyzed. Finally, the results are discussed and the conclusion is given.

2. Primitive RSA

The RSA details are mentioned in almost all cryptography literature. As a brief explanation, the RSA assumptions are in the following where p and q are two large prime numbers:

$$n = pq \tag{1}$$

$$\varphi = (p-1)(q-1) \tag{2}$$

$$e < n,\ gcd(e, \varphi) = 1 \tag{3}$$

$$d = e^{-1} \bmod \varphi \tag{4}$$

Considering m as plaintext, it should be divided into blocks smaller than n. $C_i = m_i^e \bmod n$ is the encryption and $m_i = C_i^d \bmod n$ is the decryption operation. The variable i is used to indicate the block numbers. As mentioned in the introduction, the greater the key is, the more secure the data transfer will be[4,8].

3. MRSA Parallel Encryption Algorithm

Whenever some processing elements come together and collaborate to solve massive problems in a reasonable time, a parallel computing environment is formed. In the MRSA method, mesh topology is used as parallel processing architecture. This architecture has $\sqrt{N}$ diameter where N is the number of processor element. Each intermediate node is connected to four other nodes, each side node is connected to three nodes, and each corner node is connected to two nodes.

We have considered 10 nodes in our design which nine of them form the mesh, and one is the coordinator. From another point of view, the proposed architecture can work as a cryptographic coprocessor, which collaborates with the CPU. In this case, the coordinator can be the CPU itself. Digital embedded systems do have a CPU, and this coprocessor is used to perform the cryptographic operations faster and with higher throughput. However, the number of nodes can be more or less but not less than five. Number of nodes should follow a mesh rule which is S^2 where S is the length of mesh; in addition, a processor element is added as the coordinator of first operation. Using more nodes, the more parallelization is gained but employment of more processor elements for smaller data and key lengths should be avoided. A tradeoff between the number of processor elements, the data size and the key length exists. We have chosen a mesh with nine nodes to describe our solution.

3.1 Mesh Topology Concept

Mesh is a two-dimensional network, which is obtained by arranging N processor elements into an $S \times S$ array, where $S = \sqrt{N}$. A simple mesh is shown in Figure 1 for $S = 4$. Every mesh has S^2 nodes that are connected to their immediate neighbors. The processor in row j and column k is denoted by $P(j, k)$, where $0 \le j \le S-1$ and $0 \le k \le S-1$. Four communication lines link $P(j, k)$ to its neighbors $P(j+1, k)$, $P(j-1, k)$, $P(j, k+1)$, and $P(j, k-1)$[22]. Processor elements on the boundary rows and columns have less than four neighbors and hence less connections.

Mesh is a fixed-degree network[23]. The degrees of the interior processor elements, the four corner processor elements, and the remaining edge processor elements of a mesh are 4, 2 and 3 respectively. A mesh has S rows and S columns. Therefore, transporting a piece of data from the northwest processor to the southeast processor requires traversing $S-1$ rows and $S-1$ columns. A message originating from one corner of the mesh and traveling to the opposite corner of the mesh(diameter) requires traversing a minimum of $2S-2$ communication links[23,24], which has the order of $O\left(\sqrt{N}\right)$[25].

3.2 The MRSA Method

The MRSA method is employing mesh topology to carry out RSA encryption in a faster way. The following definitions are used for the algorithm.

$$e_d : \sum_{i=-1}^{S-3} 2^{S+i}$$

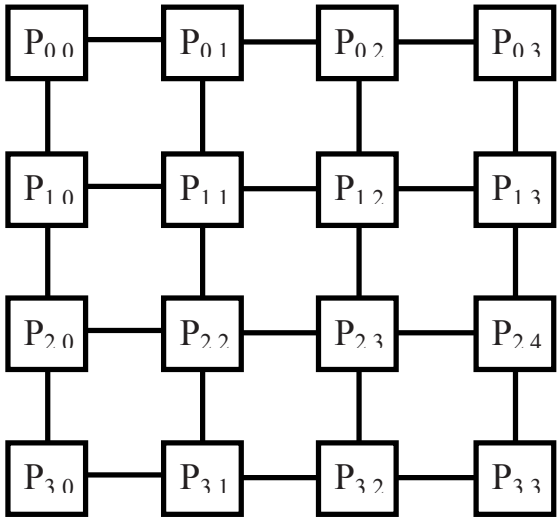

Figure 1. Mesh interconnection network.

e_o: The exponentiation for quotient of dividing e into e_d
e_r: The exponentiation for reminder of dividing e into e_d
p_i: i^{th} processor element

The values e_r and e_o are computed using the following formulas:

$$e_o = e \; div \; e_d \tag{5}$$

$$e_r = e \; mod \; e_d \tag{6}$$

The scheme of the MRSA using mesh topology is demonstrated in Figure 2(a). It should be considered that there is no need for these processor elements to be full-function processors. They just need to be able to perform multiplication and division operations. They don't need to be able to perform other operations. Receiving the data, doing the operation and sending the result are the only tasks of these processor elements. There is no need for router, routing algorithm and complex hardware devices.

PEs (Processor Elements) are simple digital circuits as shown in Figure 2(b). The modulo operation will be done based on n. The value of n will be in place, which means n will be fed into the PEs as the initial step.

In the first step, a processor element p, calculates $m^{e_o} \; mod \; n$ and sends it as the input of processor element p_0 to send it as two inputs of processor elements p_3. Then the result of equation $m^{e_r} \; mod \; n$, which is calculated in p, is sent to processor element p_0 to send it as input of processor elements p_1. In this scheme, Some PEs even don't perform multiplication and modulo and just send the input values to the next PE. The outputs of p are A and B which are computed as following.

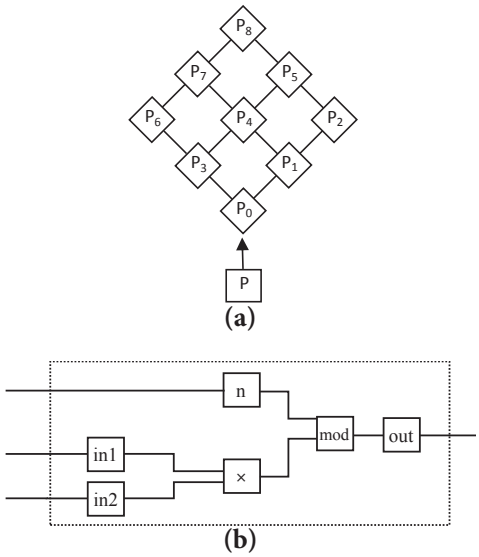

Figure 2. **(a)** Processor elements in 3×3 sample mesh architecture **(b)** Processor element scheme.

$$A = m^{e_o} \; mod \; n \tag{7}$$

$$B = m^{e_r} \; mod \; n \tag{8}$$

The processor element p_0 receives A, and sends it to p_3. Afterwards, p_0 sends B to p_1. The processor element p_3 uses A as two inputs and calculates $A^2 \; mod \; n$ and sends the results to p_4 and p_6 as its inputs. The processor element p_1 doesn't perform any calculation and sends the received value which is B to p_2. The processor elements p_4 and p_6 compute $A^4 \; mod \; n$ as their inputs is $A^2 \; mod \; n$. the processor element p_6 sends its results to p_7 but p_4 sends it to p_7 and p_5. It means that one input of p_7 is from p_6 and the other one is from p_4. Just like p_1, p_2 doesn't perform any calculation and sends the received value which is B to p_5. The processor element p_7 perform the computation of $A^8 \; mod \; n$ with its inputs $(A^4 \; mod \; n)$ and p_5 computes $A^4 \times B \; mod \; n$. The processor elements p_7 and p_5 send their values to the p_8. The output of p_8 is the encryption of m. The pseudo code of MRSA algorithm is illustrated in Figure 3.

3.3 MRSA Generalization

Apart from this simple example of a 3×3 mesh, the mesh can be of any size. The steps of the algorithm and the way processor elements should behave to do the necessary computations is formulated as following:

$\forall i, \; i \in \{i \geq 0 \; and \; i \leq S-1\}$; p_i sends its input to output (9)

MRSA Algorithm for S = 3
e_o= e div e_d
e_r= e mod e_d
A= m^{e_o} mod n (using binary method)
B= m^{e_r} mod n (using binary method)
in par; p_3: A × A (mod n), p_1:B
in par; p_6, p_4: A^2 mod n × A^2 mod n(mod n), p_2:B
in par; p_7: A^4 mod n × A^4 mod n(mod n), p_5:B × A^4 mod n
p_8: A^{12} mod n × B (mod n)

Figure 3. Pseudo code of MRSA.

$\forall j \forall l, j \in \{j \geq 1 \text{ and } j < S\}, l \in \{l \geq 0 \text{ and } l < S\}$; $p_{(j\times s)+l}$ computes MulMod (10)

$p_{(j\times s)+1}$ computes one multiplication and one modulo using the two inputs which we call them together a MulMod operation in brief. If $p_{(j\times s)+l}$ has one input, the other input value will be assumed to be the same as the received value.

Based on the Equation (9), the processor elements p_1 and p_2 only send their results to their successor node and don't perform any computation. Therefore, we can have the following formulas for a 3×3 mesh.

$$p_1: p_2 : \text{B} \tag{11}$$

$$p_3: A^2 \bmod n \tag{12}$$

$$p_4: p_6 : A^4 \bmod n \tag{13}$$

$$p_5: A^4 \times B \bmod n \tag{14}$$

$$p_7: A^8 \bmod n \tag{15}$$

$$p_8: A^{12} \times B \bmod n \tag{16}$$

In this example, $S=3$, therefore:

$$e_d = \sum_{i=-1}^{S-3} 2^{S+i} = 2^2 + 2^3 = 12 \tag{17}$$

A known fact in number theory is:

$$(a\times b) \bmod n = ((a \bmod n) \times (b \bmod n)) \bmod n \tag{18}$$

Based on this fact, the following computations are used to prove that the output of the p_8 is the encrypted message. Output of p_8 is the result of $A^{12} \times B \bmod n$. Therefore:

$$C = (A^{12} \times B) \bmod n = ((A^{12}) \bmod n \times B \bmod n) \bmod n$$

$$= ((A)^{12} \bmod n \times B \bmod n) \bmod n$$

It is derived from Equations (7) and (8) that:

$$= ((m^{e_o} \bmod n)^{12} \bmod n \times (m^{e_r} \bmod n) \bmod n) \bmod n$$

Utilizing Equation (17):

$$= ((m^{e_o} \bmod n)^{e_d} \times (m^{e_r} \bmod n)) \bmod n$$

$$= \left(\left(m^{e_o}\right)^{e_d} \times m^{e_r} \bmod n\right)$$

$$= \left(\left(m^{e_o}\right)^{e_d}\right)^{+e_r} \bmod n$$

And using Equations (5) and (6):

$$= m^e \bmod n$$

The ciphertext resulting from MRSA is the same as original RSA.

4. Performance and Analysis

The MRSA method has improved the modular exponentiation process as following to do faster computations:

$$r = e \bmod e_d$$

$$m^e = m^{e/ed} \times \ldots \times m^{e/ed+r}$$

$$m^e = m^r \prod_{i=1}^{e_d} m^{e/e_d} \tag{19}$$

In the MRSA algorithm only p should be enough powerful to compute modular exponentiation, and the other ones should do at most one MulMod operation. Some processor elements are even simpler and just pass the received value. There is no need to raise m to the power of e, and p just needs to raise m to the power of e/e_d (in this example $e/12$) and this reduction makes the computation faster. Using binary exponentiation in p, we can achieve a more appropriate execution time. The performance of the algorithm is explained briefly as following.

The MRSA is a new method to calculate RSA, which its results are highly depending on the number of MulMod operations. The more blocks of MulMod are used, the more speedup will be gained, and fewer steps will be applied to calculate the total value.

As described in the literature from[8,26-28], the number of multiplications in binary method for the worst case is $2(k-1)$ and for the best case is $(k-1)$ where k is the bit length of the exponent, which is e. The MRSA method is mainly divided into two parts. The first part of MRSA is based on the binary method. Let x be the mesh diameter,

A and B in Equations (7) and (8), as the powers, will be divided to the number of PEs, which is $\sum_{i=-1}^{x-3} 2^{x+i}$. Hence, $2\left(k-1-\log\left(\sum_{i=-1}^{x-3} 2^{x+i}\right)\right)$ multiplications in the worst case will be applied to compute A and B consequently. Thus, the number of multiplications of the first part in the worst case in the coordinator is:

$$k' = 2\left(k-1-\log\left(\sum_{i=-1}^{x-3} 2^{x+i}\right)\right) \quad (20)$$

The total number of multiplications for the mesh as the second part is x^2-x since x PEs do not carry out multiplication, thus the total number of multiplications for the best case will be $2\left(k-1-\log\left(\sum_{i=-1}^{x-3} 2^{x+i}\right)\right)+x^2-x.$

The other subject that should be discussed is the concurrency of the multiplications. Not only the number of multiplications is decreased in this approach, but also there are some multiplications performing at the same time in different PEs, which decreases the time complexity. Although, the total number of PEs carrying out multiplication mesh is x^2-x, where x is the diameter of mesh, the total time for performing the multiplications is $2x$-1, due to the concurrency of PEs. In result, the number of multiplications for the MRSA is $2\left(k-1-\log\left(\sum_{i=-1}^{x-3} 2^{x+i}\right)\right)+2x-1.$

Having the convenient formula for *geometric series*:

$$\sum_{k=0}^{n} ar^k = \frac{a(1-\mathrm{r}^{n+1})}{1-r} \quad (21)$$

Therefore,

$$\sum_{i=-1}^{x-3} 2^{x+i} = \sum_{i=0}^{x-2} 2^{x+i-1} = \sum_{i=0}^{x-2} 2^{x-1}2^{i} = \frac{2^{x-1}(1-2^{x-2+1})}{1-2}$$

$$\frac{2^{x-1}(1-2^{x-2+1})}{1-2} = \frac{2^{x-1}-2^{2x-2}}{-1} = 2^{2x-2}-2^{x-1} \quad (22)$$

The reduction of the number of multiplications is reasonable. Using the calculation method from(5), the number of multiplication is illustrated by $\eta(k,x)$. Eventually, the total number of multiplications in the worst case is:

$$\eta(k,x) = 2(k-1-\log(2^{x-1})(2^{x-1}-1))+2x-1$$
$$\eta(k,x) = 2(k-1-(\log(2^{x-1})+\log(2^{x-1}-1)))+2x-1$$
$$\eta(k,x) = 2(k-1-(x-1+\log(2^{x-1}-1)))+2x-1$$
$$\eta(k,x) = 2(k-1-x+1-\log(2^{x-1}-1))+2x-1$$
$$\eta(k,x) = 2(k-x-\log(2^{x-1}-1))+2x-1$$
$$\eta(k,x) = 2(k-\log(2^{x-1}-1))-1 \quad (23)$$

The above Table 1 shows that the number of multiplication operations of primitive RSA will be reduced using the MRSA method compared to the other methods as well as the TRSA. This improvement depends on the number of MulMod blocks in the mesh topology and the length of RSA cryptographic key.

The speedup of MRSA to the binary method in the worst case is calculated using the following formula:

$$Speedup = \frac{2(k-1)}{2(k-\log(2^{x-1}-1))-1} = \frac{(k-1)}{(k-\log(2^{x-1}-1)-0.5)} \quad (24)$$

Figure 4(a) shows the MRSA method using multiple bit key lengths and mesh diameters based on the formula of the worst case. Figure 4(b) presents the same figure from another point of view to figure out the speedup for each configuration.

Table 1. compares the number of multiplications in the average, and worst case for *binary*, *CRT*, *Montgomery*, *TRSA*(19), and *MRSA*

Method	Average Case		Worst Case	
Binary	*General form*	*Criteria*	*General form*	*Criteria*
	1.5(k–1)	*1535*	*2(k–1)*	*2046*
TRSA	*1.5(k–1)–0.5l*	*1505*	*2(k–1)–l*	*1986*
MRSA	*1.5(k–log(2^{x-1}–1))+0.5x–1*	*1495*	*2(k–log(2^{x-1}–1))–1*	*1967*
CRT	*3k^2/4+k*			*787456*
Montgomery	*2k^2+k*			*2098176*

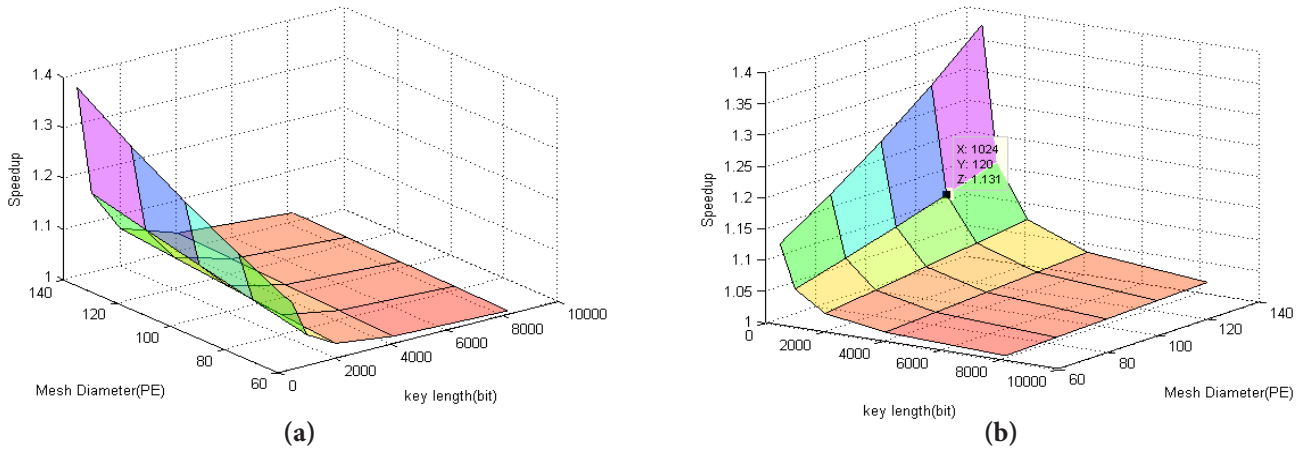

Figure 4. Comparison of speedup for multiple key lengths and mesh diameters.

Figure 4 (a) shows that exploiting more PEs in MRSA will result in better performance for greater key lengths. The increase of speedup in Figure 4 (b) is depicted with a red arrow.

Selecting the number of processor elements for a mesh is dependent to the area and the speed, which is desired for the security needs of the target system. There is a tradeoff between speed and number of PEs. It must be considered that the number of PEs must be increased to a place that the overhead of multiple PEs does not increase the multiplications which leads to reduction in speed.

5. Conclusion

The MRSA is carrying out RSA cryptography employing parallel mesh topology. Using more secure bit key lengths such as 1024, 2048, and 4096 the MRSA has more suitable speed than the well-known methods. This method uses a coprocessor, which is made up of MulMod blocks that collaborates with CPU. Considering *p* as the main CPU of an embedded device, the MulMod blocks in mesh topology represent the coprocessor.

The speedup of MRSA method to the previous methods depends on the number of MulMod blocks. As it is seen from the Table 1 and Figure 4, MRSA method is better in terms of the number of multiplications. This improvement is increased using more MulMod blocks. As the number of MulMod operations decreases, the steps as well as the execution time of the MRSA will be decreased.

The architecture of this method is designed to be scalable and can be resized to suit the needed performance. A hardware solution outperforms a software solution as it is almost the case[29,30]. It's a cheap way of achieving high performance.

6. References

1. Goossens K, Dielissen J, Meerbergen Jv, Poplavko P, Edwin Rijpkema A, Waterlander E, et al. Guaranteeing the quality of services in networks on chip. In: Jantsch A, Tenhunen H, editors. Networks on Chip. United States of America: Springer Science + Business Media, Inc.; 2004.
2. Damrudi M, Ithnin N. State of the Art Practical Parallel Cryptographic Approaches. Australian Journal of Basic and Applied Sciences. 2011; 5(7): 660–77.
3. Bielecki W, Burak D. Parallelization Method of Encryption Algorithms. In: Pejas J, Saeed K, editors. New York, USA: Springer; 2007.
4. Fan W, Chen X, Li X. Parallelization of RSA Algorithm Based on Compute Unified Device Architecture. 9th International Conference on Grid and Cooperative Computing (GCC); IEEE Computer Society;Nanjing, China. Washington, DC: USA. 2010 Nov. p. 1–5.
5. Lara P, Borges F, Portugal R, Nedjah N. Parallel modular exponentiation using load balancing without precomputation. Journal of Computer and System Sciences. 2012; 78(2):575–82.
6. Li Y, Liu Q, Li T. Design and implementation of an improved RSA algorithm. International Conference on E-Health Networking, Digital Ecosystems and Technologies (EDT); Shenzhen: China. IEEE Computer Society; 2010 Apr. p. 17–18.
7. Qing L, Yunfei L, Lin H. On the design and implementation of an efficient RSA variant. 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE);

China. Piscataway, NJ, USA: IEEE Computer Society. 2010 Aug 20–22.
8. Sepahvandi S, Hosseinzadeh M, Navi K, Jalali A. An Improved Exponentiation Algorithm for RSA Cryptosystem. International Conference on Research Challenges in Computer Science, ICRCCS '09, Shanghai. Piscataway, NJ: USA; IEEE Computer Society; 2009 Dec. p. 28–29.
9. Teerakanok T, Kamolphiwong K. Accelerating asymmetric-key cryptography using Parallel-key Cryptographic Algorithm (PCA). 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Pattaya, Chonburi, Thailand: NY; IEEE; 2009 May 6–9.
10. Ciet M, Neve M, Peeters E, Quisquater J-J. Parallel FPGA implementation of RSA with residue number systems - can side-channel threats be avoided? IEEE 46th Midwest Symposium on Circuits and Systems; Cairo: Egypt. 2004 Dec 30.
11. Fournier JJA, Moore S. Hardware-Software Codesign of a Vector Co-processor for Public Key Cryptography. 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, DSD; Dubrovnik Washington, DC: USA. 2006 Aug 30–Sep 1.
12. Jiang H, Yang G. Resistant against power analysis for a fast parallel high-radix RSA algorithm. International Conference on Electric Information and Control Engineering (ICEICE); Wuhan. 2011.
13. Nedjah N, Mourelle LD. High-performance SoC-based implementation of modular exponentiation using evolutionary addition chains for efficient cryptography. Appl Soft Comput [Article]. 2011 Oct; 11(7):4302–11.
14. Nibouche O, Nibouche M, Bouridane A, Belatreche A. Fast architectures for FPGA-based implementation of RSA encryption algorithm. IEEE International Conference on Field-Programmable Technology; Brisbane: Australia. 2004 Dec. p. 6–8.
15. Perin G, Mesquita DG, Herrmann FL, Martins JB. Montgomery modular multiplication on reconfigurable hardware: Fully systolic array vs parallel implementation. VI Southern Programmable Logic Conference (SPL), Ipojuca. 2010 Mar. p. 24–6.
16. Qiang L, Fangzhen M, Dong T, Xu C. A regular parallel RSA processor. The 47th Midwest Symposium on Circuits and Systems MWSCAS '04, Hiroshima: Japan. 2004 Jul. p. 25–8.
17. Alkar AZ, Sonmez R. A hardware version of the RSA using the Montgomery's algorithm with systolic arrays. Integration, the VLSI Journal. 2004; 38(2):299–307.
18. Damrudi M, Ithnin N. Parallel RSA Encryption based on Tree Architecture. Journal of the Chinese Institute of Engineers. Taylor and Francis. 2012.
19. Damrudi M, Ithnin N. An Optimization of Tree Topology Based Parallel Cryptography Mathematical Problems in Engineering. 2012. p. 10.
20. Koc CK. High-Speed RSA Implementation. Redwood City: CA. 1994.
21. Wu C-L. Fast Parallel Montgomery Binary Exponentiation Algorithm Using Canonical- Signed-Digit Recoding Technique. Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing; Taipei, Taiwan. Berlin, Germany: Springer-Verlag. 2009. p. 428–38.
22. Akl SG. The Design and Analysis of Parallel Algorithms. USA: Prentice-Hall; 1989.
23. Miller R, Boxer L. Algorithms Sequential and Parallel: A Unified Approach. Second Edition ed. Hingham, Massachusetts; 2005.
24. Parhami B. Introduction to Parallel Processing Algorithms and Architectures. Melhem RG, editor. Santa Barbara, California: Kluwer Academic Publishers; 2002.
25. El Rewini H, Abd El Barr M. Advanced Computer Architecture and Parallel Processing. USA: John Wiley and Sons Publishing; 2005.
26. Brickell EF. A Survey of Hardware Implementation of RSA. Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology. 705060: Springer-Verlag. 1990. p. 368–70.
27. Sun D-Z, Cao Z-F, Sun Y. How to compute modular exponentiation with large operators based on the right-to-left binary algorithm. Applied Mathematics and Computation. 2006; 176(1):280–92. doi: 10.1016/j.amc.2005.09.062.
28. Zhang CN. An improved binary algorithm for RSA. Computers and Mathematics with Applications. 1993; 25(6):15–24. doi: 10.1016/0898-1221(93)90295-7.
29. Thapliyal H, Srinivas MB. VLSI Implementation of RSA Encryption System using Ancient Indian Vedic Mathematics. Proceedings of the Micro technologies of The New Millennium (VLSI Circuits and Systems). 2005 May.
30. Celikel E, Davidson J, Kern C. Parallel performance of DES in ECB mode. IEEE International Symposium on Computer Networks ISCN' 06. Istanbul, 2006.