International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Special Issue 28 (August, 2015), PP. 7-11

CUBE BASED RSA ANALYSIS AND OPTIMIZATION

[1]Masumeh Damrudi*, [1,2]Kamal Jadidy Aval
[1]Department Department of Computer Science,
Firoozkooh Branch, Islamic Azad University, Firoozkooh, Iran
[2]Department of Computer System and Communication,
Faculty of Computing, Universiti Teknologi Malaysia,
81310, Skudai, Johur Bahru, Malaysia
*m.damrudi@gmail.com

***Abstract*— Abstract**

One of the important algorithms in public key cryptography is RSA. The RSA is expensive due to using modular exponentiation for greater keys. A parallel method is presented in previous work. The proposed algorithm (CRSA), employs hypercube interconnection network to make RSA parallel. This paper presents the optimization of CRSA along with simulation results. The results of the conducted simulations indicate that this method requires less time to carry out encryption and decryption process compared to the original RSA and the other existing parallel approaches in the literature. These results are evaluated mathematically using time complexity.

***Index Terms*— Parallel processing; cryptography; RSA; hypercube; pipeline; optimization**

I. INTRODUCTION

Security takes the main role of transferring confidential information such as the passwords of master cards through unreliable networks. The first way to overcome this issue is cryptography. The RSA is a safe algorithm for public key cryptography while using long keys [1]. The most time consuming part of RSA is modular exponentiation [2]. A new approach using hypercube interconnection network is presented in the previous work. In this paper, the optimization of newly proposed CRSA (Cube based RSA), is compared to the *Bielecki and Burak[3]*, *CRT[4]*, *Montgomery[5]*, and the binary *[4]*as an accepted method, in terms of the number of multiplications and execution time.

The rest of this article is organized as follows. Firstly, we describe the pipelining of the CRSA. Then, the optimization of this work is explained. Afterwards, mathematical analysis is discussed. The simulation results are explained in section 5. Finally, the conclusion of the work is drawn.

2. *Pipelined CRSA*

CRSA can be applied as a coprocessor for embedded systems, for example, wireless sensor nodes, which require more speed to transfer information. Enhancing CRSA, pipelining mechanism is employed to achieve higher throughput. In the pipelined variation of CRSA, Processor Elements drastically decrease the latency between data elements [6]. The CRSA for two cubes is presented in Fig. **1**.

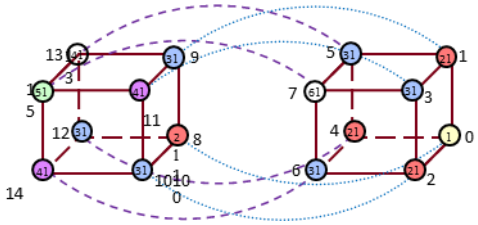


Fig. 1 CRSA algorithm on hypercube architecture

Considering the encryption as $C_i=m_i^e \bmod n$ where i is the block number, in CRSA, computing C_i should be finished before the C_{i+1} starts to be computed. In this enhanced approach, when the results of current operations are sent to the next level in hypercube, the computation for the following ciphertext are 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 cipher texts are not ready yet. In the following, the assumptions are the same as the assumption of CRSA.

The original idea illustrated in **Fig. 1** can be improved more using pipeline mechanism. The pipeline phases are shown in the following figures.

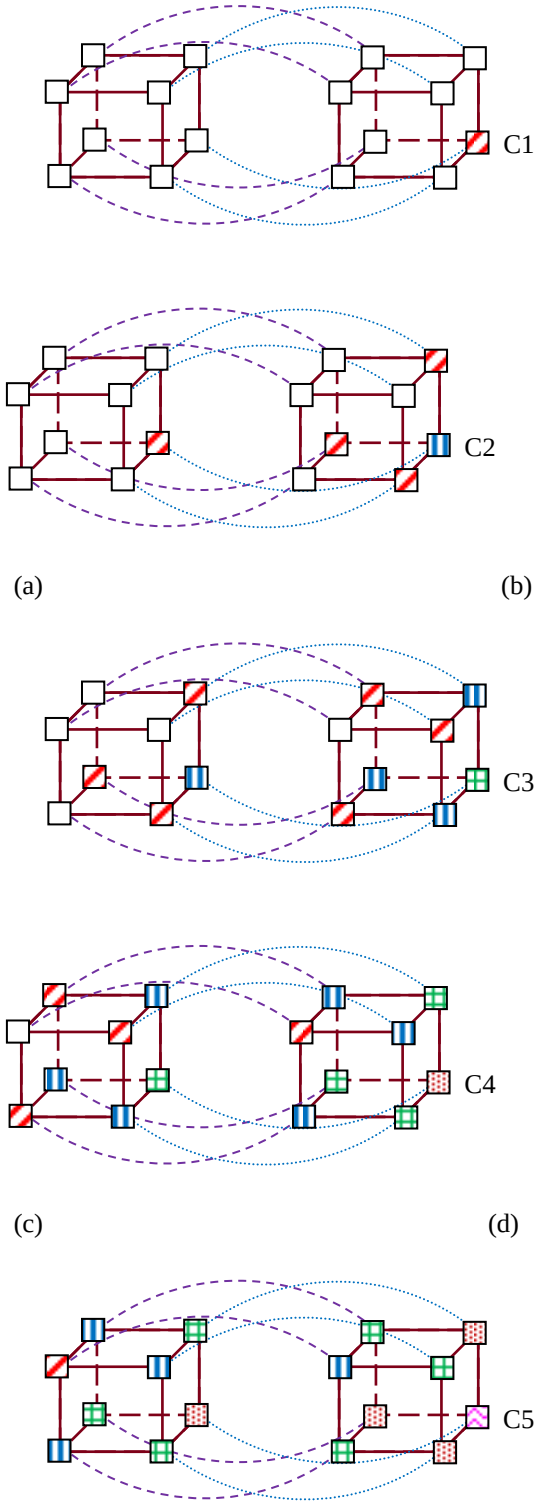


Fig. 2 Steps of pipelining in CRSA

The number of levels depends on the number of cubes in hypercube. However, the number of nodes can be more or less but not less than eight. Number of nodes should follow a hypercube rule, where the number of nodes is dividable by eight. In addition, a processor element is added as the coordinator of the first operation. Using more nodes, the more parallelization and pipeline levels will be gained. The volume of operations at the coordinator will be decreased by the increase of cubes in the hypercube, which leads to less execution time. There is always a tradeoff between the number of processor elements, the data size and key length.

In the so called example, PEs indexed 1, 3, 4, 5, 6, 8, 9, 10, and 12 perform one MulMod operation while the others do two MulMod except PEs 0 and 2 do not perform any MulMod, and PE 7 performs three multiplications. The numbers of MulMod in PEs differ according to their level and position. Although the PEs are different, the PEs in each level are equal, which means PEs of each level will terminate their operation at the same time. Based on mentioned facts, the PEs of levels two and three perform one MulMod, which means $T_2 = T_3$.

The first level of cube and the coordinator are considered first pipeline level together, and their execution time depends on the e_o. This architecture is heterogeneous that means the processor elements of the first and second level are different from the other process elements as described in below. The processor element known as coordinator must be more advanced than the others and the processor elements, which need more than one MulMod are more advanced than the other PEs. Knowing these facts, the difference between the execution time of these processor elements and the others is very small. As mentioned above, the coordinator can be the CPU of the embedded system, and the hypercube part can be used as a coprocessor. Therefore, for the execution time on all levels, we have:

$$T_1 \cong T_2 = T_3 \cong T_4 \cong T_5$$

Using pipelining, as its stages are clarified in the Fig. **2**, the throughput of parallel approach with pipelining will be about five times of the throughput of parallel approach for a hypercube with 16 nodes. When C_1 is computed, calculation of C_6 will be started as a new block; whilst in the CRSA, when C_1 is computed, calculation of C_2 will be started as the next block. *Th* is defined as throughput in the subsequent equations.

$$T_{pipelined\ parallel} \cong 1/5\ T_{mainCRSA}$$

$$Th_{pipelined\ parallel} \cong 5 Th_{mainCRSA}$$

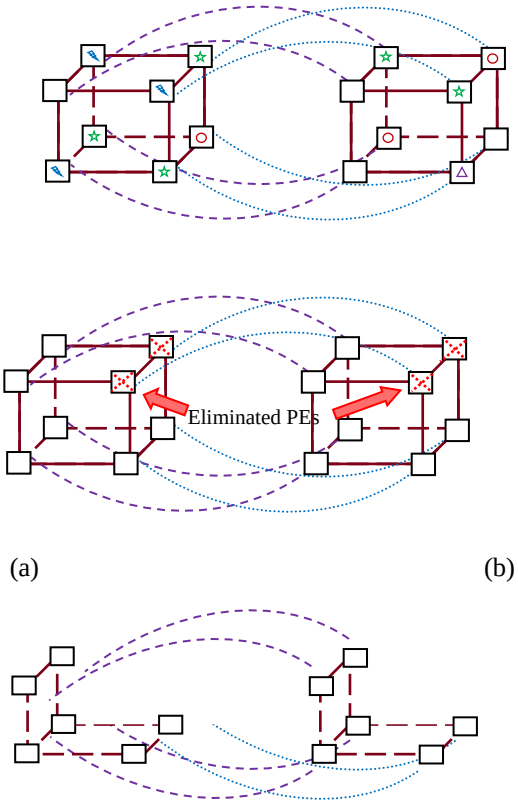
It should be considered that if the number of processor elements increases, the throughput of pipelining will increase too. Assuming N_c as the number of cubes in hypercube, the relation of the throughput for the pipelined parallel approach and the throughput for original CRSA parallel approach is as following:

$$Th_{pipelined\ parallel} = (N_c + 3)\ Th_{mainCRSA}$$

Applying this enhancement, all processor elements are in use all the time computing cipher texts. In contrast, in the main CRSA, at each time slice, just one level of the processor elements was in use doing the computation, and the others were idle. Pipelining mechanism provides appropriate distribution of the work load among resources and improves throughput.

3. *Optimization of CRSA*

Although the main method of CRSA with pipelining has admissible results, we have improved and optimized it to increase efficiency as well as decreasing the area, which is one of the most important factors in energy, and size limited systems like battery-powered sensor nodes. The main CRSA has some nodes, which are performing the same task and producing the same results during execution of the algorithm. Knowing this, the results of some PEs can be employed instead of the others with the same results. These redundant PEs can be eliminated in the architecture. By the increase in the number of cubes, the redundant PEs will be increased where applying optimization reduces the growth rate of the architecture. Considering two cubes in a hypercube, the PEs which are doing the same operation are marked with the same shape in Fig. **3** (a). Fig. **3** (b) illustrates the nodes, which can be eliminated and finally, the nodes that can be eliminated from the hypercube to optimize the algorithm are omitted in Fig. **3** (c). It should be considered that the way processor elements are connected is not changed and only the connections to the eliminated nodes are removed from the architecture. This elimination leads to a simpler architecture and smaller area usage. The eliminated PEs do not include the ones playing the role of connecting two cubes. If the level of parallelizing is more than five, the number of eliminated PEs will increase.



(a) (b)

(c)

Fig. 3 Optimization of CRSA using two cubes with 16 PEs

The elimination decreases the size of the hypercube in Fig. **3** from 16 to 12 PEs. The efficiency of the parallel approaches is measured using efficiency factor, which is given as $E=S/P$ where P and S are the number of processor elements and speedup respectively [7-9]. Assuming the number of PEs as 16 like the example in Fig. **3**, employing optimization the efficiency of the algorithm will be increased from $S/16$ to $S/12$. Taking into the general form, the efficiency of main CRSA is $E=S/8N_c$ where N_c is the number of cubes in the hypercube. Optimizing the CRSA, the efficiency of optimized CRSA is $E_{op}=S/6N_c$.

4. *Mathematical analysis*

Along with the throughput and efficiency, which are common parameters to analyze the architectures, the number of multiplications of an RSA method is known as a useful parameter to compare the results [10, 11]. The number of multiplications for binary as an accepted method is $k-1$, $3/2(k-1)$ and $2(k-1)$ in the best, average, and worst case respectively where k is the number of bits of the exponent [4, 11, 12].

Let N_c be the number of cubes in hypercube, so the power of A and B in equations 17 and 18, will be divided into the number of PEs, which is $10+3\times 2^2\sum_{i=3,j=4}^{i=2^{N_c}-1} i\,(i=2j-1, j=2j)$. Thus the number of multiplications of the coordinator for the best case is:

$$k'=(k-1)-\log\left(10+3\times 2^2\sum_{i=3,j=4}^{i=2^{N_c}-1} i\,(i=2j-1, j=2j)\right)$$

From mathematical point of view, we have:

$$\sum_{i=3,j=4}^{i=2^{N_c}-1} i\,(i=2j-1, j=2j)=2^{N_c+1}-(N_c+3)$$

Therefore,

$$k'=(k-1)-\log\left(10+3\times 2^2\times(2^{N_c+1}-(N_c+3))\right)$$

$$k'=(k-1)-\log\left(24\times 2^{N_c}-12N_c-26\right)$$

The calculation of the total number of multiplications for optimized hypercube is described below. Considering two cubes in the architecture, PEs indexed 1, 3, 4, 5, 6, 8, 9, 10, and 12 perform one MulMod while the others do two MulMod except PEs indexed 0 and 2, which do not perform any MulMod and PE_7 that performs three multiplications. It must be considered that in the optimized form, PEs indexed 1, 3, 9, and 11 are eliminated. The construction of the first cube includes PEs indexed 0, 2, 4, 5, 6, and 7 whilst PEs indexed 4, 5, and 6 perform one MulMod and PE_7 does three MulMod operations, which totally becomes 6 MulMod operations. The second and last cube includes processor elements indexed 8, 10, 12, 13, 14 and 15, whilst 8, 10, and 12 perform one MulMod and PEs indexed 13, 14, and 15 do two MulMods, which totally is nine MulMods. As a generalization, if more than two cubes are employed, the middle cubes will have a total number of 10 MulMod operations due to the need for one more MulMod operation while getting the result of the next cube to be passed to the first cube. It is reminded that the result is always generated in PE_7, which is located in the first cube. Summarizing all of these, the number of MulMods is $6+9+10(N_c-2)=10N_c-5$. Thus the total number of

multiplications in the best case is *(k-1)-log*$(24\times 2^{N_c}-12N_c-26)$*+10Nc-5*.

Although the number of multiplications is a significant parameter, the concurrency of the multiplications mentioned above can also be discussed further more. In the proposed approach, not only the number of multiplications is decreased, but also there are some multiplications carried out simultaneously in different PEs, which decreases the execution time. Although, the total number of multiplication in the optimized hypercube is *10Nc-5*, the total time for performing the multiplications is *5Nc-1* due to the concurrency of PEs.

Using the calculation method from[13], the number of multiplication is illustrated by *η(k, Nc)* and in result, the number of multiplications for the optimized CRSA in the best case is:

$\eta(k,N_c) = (k-1)\text{-}log(24\times 2^{N_c}-12N_c-26)+5N_c-1$

Therefore:

$\eta(k,N_c) = k\text{-}log(24\times 2^{N_c}-12N_c-26)+5N_c-2$

Table 1 compares the number of multiplications in the best, average and worst case for the binary algorithm that is the best known approach [4], and optimized CRSA with two cubes as well as the number of multiplications for *CRT[4]*, *Montgomery[5]*, *Bielecki and Burak[3]*.

Table 1. Number of multiplications in Binary[4], Bielecki and Burak [3], CRT[4], Montgomery [5] and CRSA

Method		General form		k=1024 Nc=2
Bielecki and Burak [3]		2^{k-2}		4.4942e+307
CRT[4]		$3k^2/4+k$		787456
Montgomery [5]		$2k^2+k$		2098176
Competitive Methods	**Best**	**Average**	**Worst**	**Worst**
Binary[4]	k-1	1.5(k-1)	2(k-1)	2046
Optimized CRSA	$k-log(24\times 2^{N_c}-12N_c-26)+5N_c-2$	$1.5k-1.5log(24\times 2^{N_c}-12N_c-26)+5N_c-2.5$	$2k-2log(24\times 2^{N_c}-12N_c-26)+5N_c-3$	2044

While $N_c=2$, the number of PEs for optimized CRSA is 12, and the number of multiplications is 2044 in the worst case. The number of multiplication operations for average and worst cases of RSA will be reduced using the optimized CRSA method compared to the binary method [4] and the methods in Bielecki and Burak [3], CRT [4], and Montgomery [5]. This reduction depends on the number of MulMod blocks in the hypercube topology and the bit length for RSA cryptographic key.

5 *Simulation Results*

CRSA, pipeline technique and the optimization method of CRSA outperform the well known existing approaches from the literature, which are *CRT*, *Montgomery*, Binary, and *Bielecki and Burak*. The example of $N_c=2$ in mathematical analysis confirmed this result.

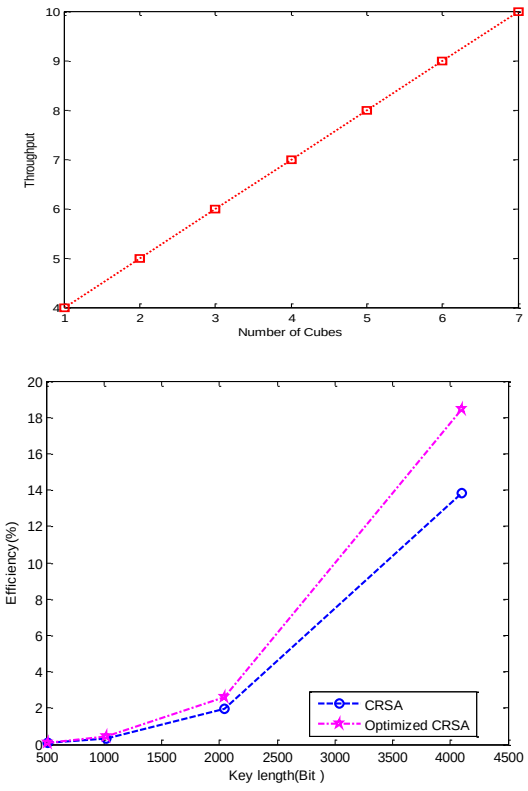
The execution time of these approaches are achieved using C++ language of Microsoft® Visual Studio 2008™, OpenMP© and OpenSSL. The Bielecki and Burak' approach is exempted from the results due to its long time of execution. The hardware platform is a quad core, which represents four PEs can perform their operation at the same time.

Table 2. Execution time of competitive methods

Method	Operation length(bit)		
	1024	2048	4096
CRT	0.443906724	4.841289807	60.31472437
Montgomery	0.026109282	0.133479549	0.813800137
Binary	0.309712094	3.867729003	51.71194862
CRSA	0.004906768	0.025235907	0.156326272

The execution values are obtained by average of 100 iterations for each method. The messages of these encryptions are 64, 128 and 256 bytes for 1024, 2048 and 4096 bit operations respectively. In this example the number of cubes is considered to be one.

The advantages of applying pipelining to the main CRSA in terms of *throughput* are presented in Fig. **4**(a). As it is explained before, the *throughput* is $Th_{pipelined\ parallelCRSA}=(N_c+3)\ Th_{mainCRSA}$. Using two cubes the throughput is five times of the situation of using one cube. The advantages of optimization in terms of *efficiency* are presented in Fig. **4**(b). The simulation results of optimization for the average of ten iterations are utilized to draw this figure. The speedup is considered by the average execution of 100 iterations of CRSA and optimized of CRSA, and the efficiency is multiplied by 100 to be in percent unit.



(a) (b)

Fig. 4 (a) The throughput of pipelined CRSA to CRSA

A. Efficiency of CRSA vs. Optimization of CRSA for three levels

Considering the number of cubes for CRSA to be two, the throughput of pipelined CRSA is five times of main CRSA. The more cubes employed, the more throughput will be obtained.

When the number of PEs for CRSA is eight, the number of PEs in Optimized CRSA becomes six. The more PEs employed, the more efficiency will be obtained considering the aforementioned tradeoff.Fig. **4**(b) shows the efficiency of CRSA and optimized CRSA for 512, 1024, 2048, and 4096 bit key length using 128, 256, 512, 1024 byte plaintext respectively. The optimized CRSA has preferred efficiency compared to the CRSA, which means the need to less area.

Selecting a hypercube among others is dependent upon the area and the speed and throughput, which are desired for the security needs of the target system. There is always a trade-off between speed and number of PEs. It must be considered that the number of PEs must be increased to a value that the overhead of multiple PEs does not lead to the reduction in the speed.

II. Conclusion

The main drawback of asymmetric algorithms like RSA is the high execution time of modular exponentiation execution time. The given algorithm, the optimized CRSA, is employed to decrease this execution time. The pipelined and optimized of this method are used to improve the throughput and efficiency of the encryption process. In addition, the results show that this method reduces the number of multiplications, which contributes in a higher speed. In addition, the pipelining method makes all of the PEs busy doing computations all the time and no PE is idle, which results in optimum throughput. Eventually, the optimized method performs the same operations exploiting fewer PEs, which leads to less area usage that is another important factor in the world with widely use of embedded systems. Moreover, other public key algorithms can be used instead of RSA. In addition, other interconnection networks are applicable to the algorithms.

References

[1] S. de Souza Raposo, *et al.*, "M-ary parallel modular exponentiation: Software vs. hardware," in *15th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*, Iran, Tehran, 2010, pp. 19-24.

[2] W. Fan, *et al.*, "Parallelization of RSA Algorithm Based on Compute Unified Device Architecture," in *9th International Conference on Grid and Cooperative Computing (GCC)* Nanjing, China, 2010, pp. 174-178.

[3] W. Bielecki and D. Burak, *Parallelization Method of Encryption Algorithms*. New York, USA: Springer, 2007.

[4] C. K. Koc, "High-Speed RSA Implementation," RSA Data Security, Inc., Redwood City, CA, USA1994.

[5] C.-L. Wu, "Fast Parallel Montgomery Binary Exponentiation Algorithm Using Canonical- Signed-Digit Recoding Technique," presented at the Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing, Taipei, Taiwan, 2009.

[6] G. Perin, *et al.*, "Montgomery modular multiplication on reconfigurable hardware: Fully systolic array vs parallel implementation," in *VI Southern Programmable Logic Conference (SPL)*, Ipojuca 2010, pp. 61-66.

[7] H. El Rewini and M. Abd El Barr, *Advanced Computer Architecture and Parallel Processing*. USA: John Wiley & Sons Publishing, 2005.

[8] V. Kumar and V. N. Rao, "Parallel depth first search. Part II. analysis," *Int. J. Parallel Program.*, vol. 16, pp. 501-519, 1987.

[9] L. R. Scott, *et al.*, *Scientific Parallel Computing*. New jersey, USA: Princeton University Press, 2005.

[10] A. Cilardo, *et al.*, "Carry-Save Montgomery Modular Exponentiation on Reconfigurable Hardware," presented at the Proceedings of the conference on Design, automation and test in Europe - Volume 3, 2004.

[11] S. Sepahvandi, *et al.*, "An Improved Exponentiation Algorithm for RSA Cryptosystem," in *International Conference on Research Challenges in Computer Science, ICRCCS '09.* , Shanghai, 2009, pp. 128-132.

[12] D.-Z. Sun, *et al.*, "How to compute modular exponentiation with large operators based on the right-to-left binary algorithm," *Applied Mathematics and Computation*, vol. 176, pp. 280-292, 2006.

[13] P. Lara, *et al.*, "Parallel modular exponentiation using load balancing without precomputation," *Journal of Computer and System Sciences*, vol. 78, pp. 575-582, 2012.