Performance of Parallel RSA on IMAN1 Supercomputer

Areej Al-Shorman
PhD Student
Department of Computer Science, King Abdullah II School for Information Technology, University of Jordan, Amman, Jordan

Mohammad Qatawneh
Professor
Department of Computer Science, King Abdullah II School for Information Technology, University of Jordan, Amman, Jordan

ABSTRACT

Numerous decryption and encryption algorithmic methods have been proposed and applied in prior research, including RSA, DES, etc. Such methods are normally assessed in their performance in accordance with the growth rates of their algorithms, based on key and input sizes. With RSA public-key security algorithms, primary operations feature modular exponentiations and reductions. As a result, sequential implementations of RSA become more computing-time, and energy-intensive. Several parallelization methods are therefore recommended in order to improve the speed of RSA algorithms. In this paper, parallel RSA algorithmic methods are assessed and then compared, based on decryption and encryption running times, speedup, and efficiency. The experimental results show that the runtime of parallel RSA algorithmic method outperform those of sequential RSA algorithmic methods.

General Terms

Computer Science, Security, Parallel Computing.

Keywords

Cryptography, RSA, MPI, Supercomputer, Public Key, Private Key, Parallel algorithm

1. INTRODUCTION

The goal of cryptographic algorithms is to securely transmit data along open channels of communication while preserving the CIA triad (confidentiality, integrity, availability) of informational resources, including information content and telecommunication data) [1]. Various methods of cryptographic algorithmic methods can be applied, namely DES, RSA, BLOWFISH and AES, towards rendering highly-sensitive information inaccessible to all but for intended recipients. Among the more important methods is RSA, which proposes to increase network transmission security through the application of key pairs. Encryption keys are termed public keys, while decryption keys are termed private keys [2]. However, the RSA algorithmic method presents numerous performance-limiting issues, including challenging mathematical problems (modular multiplications and exponentiations) and execution times [5].

Among other means of reducing RSA algorithm runtimes is the use of parallel computing [11], which involves computation wherein multiple calculations are conducted concurrently. This operates on the principle that larger problem sets can usually be partitioned into smaller sets that are then resolved simultaneously [12].

Parallel and distributed computing systems are high-performance computing systems that spread out a single application over many multi-core and multi-processor computers in order to rapidly complete the task. Parallel and distributed computing systems divide large problems into smaller sub-problems and assign each of them to different processors in a typically distributed system running concurrently in parallel [14][15] [16] [17] [18][19].

The aim of this paper is to evaluate the performances of parallel and sequential RSA algorithms. The result is conducted using IMAN1 supercomputer which is Jordan's first and fastest supercomputer. It is available for use by academia and industry in Jordan and the region and provides multiple resources and clusters to run and test High Performance Computing (HPC) codes [10].
The rest of the paper is organized as follows: Section 2 presents the related works. RSA algorithmic method is presented in Section 3, while RSA parallelization is featured in Section 4. Section 5 presents the Experimental results, and Section 6 presents the conclusion.

2. RELATED WORKS

Parallel cryptographic algorithmic methods comprise a hot area of security research, as a result of increasing requirements for speedy and efficient security-based methods that can be applied concurrently. In [4, 8] the researchers' proposal is for effective parallel RSA algorithmic methods performed on GPUs and applied in CUDA frameworks. This method further compares CPFA (CPU-based Pollard's p-1 Factorization Algorithms) and GPFA (GPU-based Pollard's p-1 Factorization Algorithms); with results demonstrating that GPFA offers faster performance than CPFA. In [5], various techniques present the standard Crypto++ Library, involving sequential RSA with the Montgomery multiplication algorithmic method, parallel RSA on many-core CPUs, and parallel RSA on many-core GPUs. Results demonstrated the speed increases of GPU implementations are superior to those of many-core CPU implementations, which in turn exceeds those of sequential CPU implementations. The researchers in [6] have applied a parallel algorithmic method for calculating d from e, wherein e is a prime number derived without resort to Euclidean algorithms. In [7], an additional RSA method is implemented in terms of multi-prime numbers p, q, and r, so as to reduce 1024-bit integers to 342-bit numbers

3. IMPLEMENTATION SEQUENTIAL RSA

RSA was initially developed in 1977 by Rivest, Shamir, and Adleman [2, 3, 5, 8, 9] and is among the most critical algorithms applied in the authentication and encryption of data for secure transmission on open networks. RSA represents the critical public-key cryptographic structure that is utilised in multiple online applications, including e-commerce and credit card processing over networks, key exchanges, and digital signatures [3, 8]. The RSA algorithmic method is considered to be slower than symmetric-key algorithmic methods for it relies on modular exponentiations.

Where RSA is used to decrypt ciphertext and generate signatures, additional computing capacity and time is typically needed. RSA features a very high computing burden in comparison to the numerous and very fast private key systems that are available. Therefore, multiple solutions were recommended to quicken RSA decryption, particularly when dealing with huge files. One of them is parallel computers and algorithmic methods.

RSA encryption applies in terms of the factorisation of very large numbers. In the instance of the utilisation of key lengths of 1024 bits or longer, it becomes difficult to defeat the security of this method, even with the use of high-performance computers [2]. Overall, RSA comprises the following trio of steps:

Step 1: Key Generation

The key generation step of RSA algorithm is a multi-step procedure:

1. Choose two quite large random prime integers' p and q and compute RSA modulus by multiplying them together.
2. Compute Toitent Function-$\varphi\ (\mathrm{n}) = (\mathrm{p}-1) * (\mathrm{q}-1)$
3. Choose value for public key e. e must be smaller than φ (n), and is coprime to φ (n). coprime means the two integers do not share any factors other than 1.
4. Compute the decryption key as our private key d using this equation: $\mathrm{de} = \mathbf{1\ mod}\ \varphi\ (\mathrm{n})$

The result of this e and n is termed as the public key (encryption key) and d and n as the private key (decryption key).

Step 2: RSA Encryption

For encrypting the plain text data M to the cipher text data C by this equation:

$$\mathbf{C} = (\mathbf{M})^{\mathrm{e}}\ (\mathrm{mod}\ \mathrm{n}) \ldots\ldots (1)$$

Step 3: RSA Decryption

For reversing the cipher text data C to the plain text data M using this equation:

$$\mathrm{M} = (\mathbf{C})^{\mathrm{d}}\ (\mathrm{mod}\ \mathrm{n}) \ldots\ldots (2)$$

Fig 1 shows the overall organization of RSA cryptography algorithm.

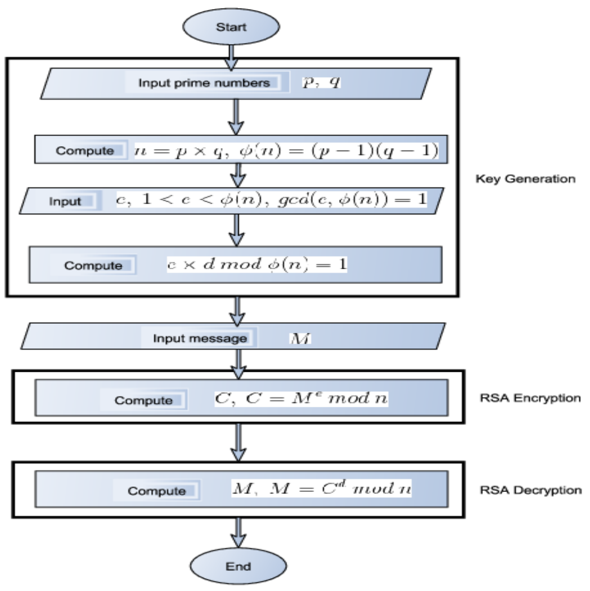
Start
Input prime numbers p, q
Compute $n = p \times q,\ \phi(n) = (p-1)(q-1)$
Input $e,\ 1 < e < \phi(n),\ gcd(e, \phi(n)) = 1$
Compute $e \times d\ mod\ \phi(n) = 1$
Key Generation
Input message M
Compute $C,\ C = M^e\ mod\ n$
RSA Encryption
Compute $M,\ M = C^d\ mod\ n$
RSA Decryption
End

Fig 1: Sequential RSA Algorithm Execution

3.1 RSA Example

In this section, we feature examples with solutions that compute both plaintext blocks (M) and ciphertext blocks (C), in order to understand better the RSA algorithmic method [8]. Example: Consider encryption and decryption using the RSA algorithm, for the following: p = 11; and q = 5.

1. Calculate $n = p * q, n = 11 * 5 = 55$
2. Calculate$\varphi\ (\mathrm{n}) = (\mathrm{p}-1) * (\mathrm{q}-1) = 10 * 4 = 40$.
3. Select e, e < φ (n) and coprime to 40. e may be 3, 7, 9,11,13,17, 19…….
4. Calculate $\mathrm{de} = \mathbf{1\ mod}\ \varphi\ (\mathrm{n})$, suppose we choose e=7.
   i. d(7)=1 mod 40, then applying Euclidean Algorithm
   ii. 40= 5(7) +5
   iii. 7=1(5) +2
   iv. 5=2(2) +1, Next extended Euclidean
   v. 1=5- 2(2)
   vi. 1=5-2(7-1(5))
   vii. 1=5-2(7) +2(5)
   viii. 1=3(5)-2(7)
   ix. 1=3(40-5(7))-2(7)
   x. 1=3(40)-15(7)-2(7)
   xi. 1=3(40)-`17(7)

   In this example, -17 is in front of our encryption key 7. This number is negative, so apply
   $\mathrm{d} = -\mathbf{17\ mod}\ 4$
   $d = 23$
5. Encrypt each plaintext letter by applying equation #1. Suppose we want to encrypt "HI". First, we encrypt H, where the numerical representation for H is 7 (m=7).So, c= 77 mod 55, then c =28 mod 55. The ciphertext of H is 28. Second, we encrypt I, where the numerical representation for I is 8 (m=8). So, c= 87 mod 55, then c =2 mod 55. The ciphertext of I is 2. The ciphertext: 28, 2.

6. Decrypt each cipher letter by applying equation # 2. .Suppose we want to decrypt "28, 2". First, we decrypt 28, so, m= 28 23 mod 55, then m =7 mod 55. The plaintext of 7 is H. Second, we decrypt 2,

so, m = 2 23 mod 55, then m =8 mod 55. The plaintext of 2 is I. The plaintext: HI.

4. IMPLEMENTATION OF PARALLEL RSA

Cryptographic algorithms are considered to be compute-intensive algorithmic methods. The key concept is to therefore instrument more efficient parallel RSA algorithms of RSA, for execution on the IMAN Zaina supercomputing cluster. To provide for parallel RSA implementations, it is desirable that the data features no dependencies. Original data or files are split into varied blocks, with each assigned to separate processors. The steps of the encryption method are stated in Table 1 and Fig2 and those of the decrypted method are stated in Table 2.

Table 1. Steps of Parallel RSA Encryption Method

Steps numbe r	Parallel RSA Encryption Method
1	The master processor Splits the text file to N blocks; where N is number of processor
2	Sent N-1 block to separated processor.
3	Each processor will encrypt the block by using encryption key e and sent it back
4	The master processor gets all ciphertext blocks from all worker processors and put them in one text file and store it in the output file

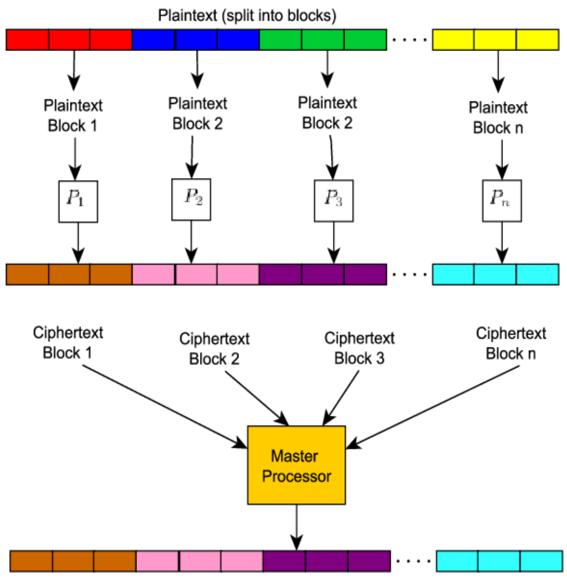

Fig 2: Block Diagram for Parallel Fig Implementation of RSA Algorithm

Table 2. Steps of Parallel RSA Decryption Method

Steps number	Parallel RSA Decryption Method
1	The master processor will send the ciphertext blocks to another or the same process that encrypt it by using decryption key (d) to be decrypted r
2	Sent decrypted block back again to master processor
3	The master processor gets all decrypted blocks from all worker processors and put them in one text file and store it in the output file

The proposed parallel RSA is implemented using C++ programming language and MPI library.

5. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, the findings are discussed and assessed based on decryption and encryption times. The IMAN1 Zaina cluster is utilized to run the experiments, with the MPI library applied in our implementations of parallel RSA algorithms. This algorithmic method is assessed based on varying input sizes and varying numbers of processors. Averages across multiple runs are then considered in the recording of outcomes. Software and hardware specifications and also the application parameters utilized are presented in Table 3.

Table 3. The Hardware and Software Specifications

Specification Type	Description
Hardware specification	Intel(R) Core (TM) I5-6200U CPU @ 2.30GHz 2.40 GHz, 8.00 GB RAM
Software Specification	Scientific Linux 6.4 with open MPI 1.5.4, C and C++ compiler
File size	128 KB, 265 KB, 512KB,1MB, 2MB, 3MB
Number of Processors	1,2,4,8,16,32,64

5.1 Encryption and Decryption Time Evaluation

Experiments were carried out so as to enhance the performances of parallel RSA, which demonstrated certain encouraging outcomes. We applied MPI in combination with the GCC infrastructure in order to apply parallel RSA operations that decrease runtimes and enhance performances. The duration of execution of various test cases is recorded though the timing utility of UNIX.

To enhance the performance of decryption and encryption implemented by parallel RSA across the sequential RSA, operations were run on 1, 2, 4, 16, 32, and 64-core processes. Every experiment was carried out 5 times and duration averages were recorded as final values for each test case. Duration is measured according to two operational classes, encryption times (durations recorded for encryption), and decryption times (durations recorded for decryption).

For these experiments, we utilized test cases to validate and confirm the robustness of our proposed version of parallel RSA. This is implemented by specifying fixed key sizes and also file sizes, which vary from 128KB up to 3MB, as depicted in Table 3. All cases are implemented repeatedly with 1, 2, 4, 8, 16, 32, and 64 processor runs over 5 cycles, with the averages of all readings recorded as final times. Results are presented in Table 4.

Table 4 presents improvement and performance comparisons of parallel runtimes on 2, 4, 8, and 16-core configurations, against sequential implementations on single processors for various test cases, which demonstrated superior load distribution between numerous processors. Then again, improvements and performances decrease when the numbers of processors increase with particular file sizes. This results from increases in communication overheads; where the advantages of parallelism are consequently diminished - particularly when elevating processor counts from 16 to 32 cores. As a result of this emergent behavior, we disabled processor counts greater than 32 cores in the experiments.

5.2 Speed up evaluation

Speed up represents the ratios between sequential and parallel runtimes. Table 4 and Fig 4 show the increasing speed ups using parallel RSA algorithms corresponding to 2, 4, 8, 16, 32, and 64-core computation, on files that are 128KB up to 3MB in size. Findings demonstrate that the parallel RSA achieves the best speed ups values, up to 30%, particularly with large processor counts.

5.3 Parallel Efficiency Evaluation

Parallel efficiency denotes ratios between speed up values and processor counts. Fig 5 demonstrates parallel efficiencies for RSA algorithms, based on various numbers of processors and on files ranging from 128KB up to 3MB in size. With dual processors, parallel RSA attains up to 99% efficiency with 3MB files sizes. Furthermore, parallel RSA attains its highest efficiencies with large processor counts involving 16 and 32 processor cores, as RSA generates superior speed up values once implemented on 16 or more cores.

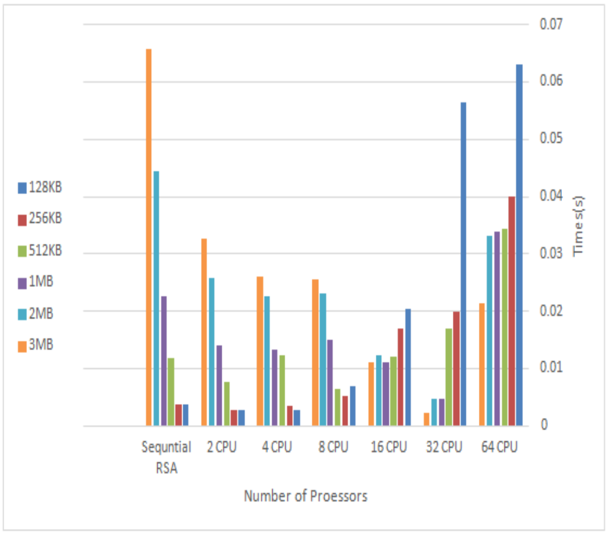

Fig 3: Execution Time for parallel RSA with varied file sizes

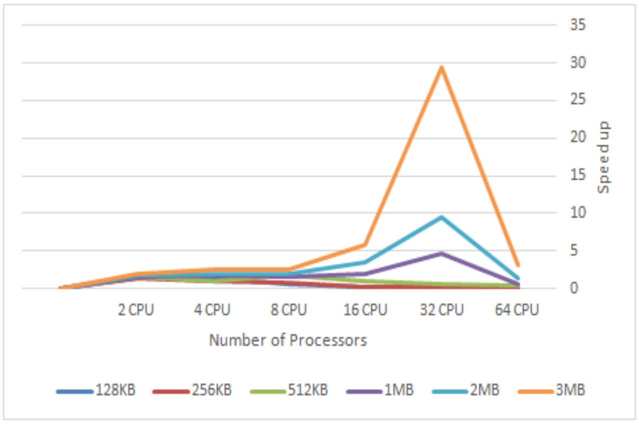

Fig 4: Speedup for parallel RSA with varied file sizes

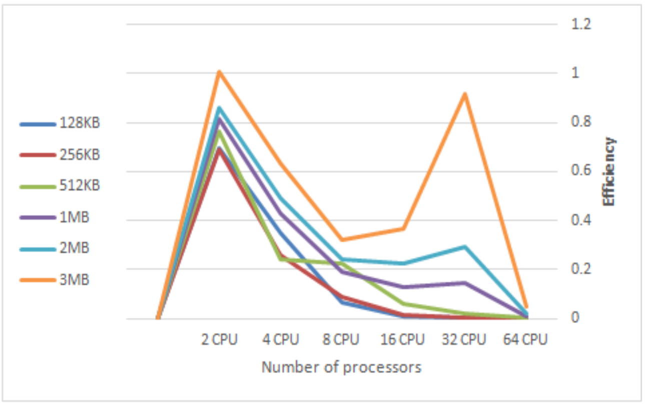

Fig 5: The efficiency for parallel RSA with varied file sizes

Table 4. Test Case: Comparative Results Obtained using Fixed Key Size with Different File Sizes

S.No	File Size	Program Segment Type	Time (in second) taken the serial and parallel execution of the same code on Different number of processor						
			Serial code (1 CPU)	2 CPU	4 CPU	8 CPU	16 CPU	32 CPU	64 CPU
1	128 KB	Encryption & Decryption Time	0.00376	0.0027	0.00268	0.00697	0.02028	0.05641	0.06304
2	256 KB	Encryption & Decryption Time	0.00369	0.00268	0.00354	0.00508	0.0169	0.019881	0.03989
3	512 KB	Encryption & Decryption Time	0.01178	0.00773	0.0122	0.00652	0.01208	0.017	0.03431
4	1MB	Encryption & Decryption Time	0.02269	0.01394	0.01329	0.01502	0.01115	0.00478	0.03394
5	2MB	Encryption & Decryption Time	0.0443	0.02569	0.02258	0.022985	0.01230	0.004681	0.03319
6	3MB	Encryption & Decryption Time	0.06577	0.03268	0.026	0.0256	0.01118	0.00224	0.02142

Table 5. Speedup for Parallel RSA with Different File sizes

S.No	File Size	Speed up of the parallel execution of the same code on different number of processors					
		2CPU	4CPU	8CPU	16CPU	32CPU	64CPU
1	128 KB	1.39259	1.40298	0.53945	0.185404	0.06665	0.05964
2	256 KB	1.37686	1.04237	0.72637	0.218343	0.18560	0.09250
3	512 KB	1.52393	0.96557	1.80674	0.975165	0.69294	0.34334
4	1MB	1.62769	1.70729	1.51065	2.034977	4.74686	0.66853
5	2MB	1.72440	1.96130	1.92734	3.601333	9.46378	1.33445
6	3MB	2.01255	2.52961	2.56914	5.882826	29.3616	3.07049

6. CONCLUSION

In this research, we present performance evaluations of the parallel RSA algorithmic method according to decryption and encryption. This method is applied with MPI library and experimental runs are carried out on the IMAN1 supercomputing cluster. Assessments of parallel RSA are always in terms of different numbers of processors in relation to input file sizes.

Overall, our findings demonstrate that parallel RSA features shorter running times for various file sizes and with limited numbers of processors, which is due to the observation that the processor counts increase with particular file sizes.

Future research will involve efficient parallel implementation of key generation in RSA algorithmic methods and will be carried out on the IMAN1 supercomputing cluster in order to reduce the execution times.

7. ACKNOWLEDGMENTS

Our thanks to Eng. Zaid Abudayyeh Who have contributed towards produce this work

8. REFERENCES

[1] Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of applied cryptography. CRC press.

[2] Saxena, S., & Kapoor, B. (2014, February). An efficient parallel algorithm for secured data communications using RSA public key cryptography method. In Advance Computing Conference (IACC), 2014 IEEE International (pp. 850-854). IEEE.

[3] Lakkadwala, M., & Valiveti, S. (2017, January). Parallel generation of RSA keys—A review. In Cloud Computing, Data Science & Engineering-Confluence, 2017 7th International Conference on (pp. 350-355). IEEE.

[4] Lin, Y. S., Lin, C. Y., & Lou, D. C. (2012, May). Efficient parallel RSA decryption algorithm for many-core GPUs with CUDA. In International Conference on *Telecommunication Systems, Modeling and Analysis (ICTSM2012).*

[5] Fadhil, H. M., & Younis, M. I. (2014). Parallelizing RSA algorithm on multicore CPU and GPU. International Journal of Computer Applications, 87(6).

[6] Chang, C. C., & Hwang, M. S. (1996). Parallel computation of the generating keys for RSA cryptosystems. Electronics Letters, 32(15), 1365-1366.

[7] Rao, B. S., & Ramesh, M. An Efficient Parallel Algorithm for Secure Data Communication Using RSA Algorithm. red, 100111(11101000), 11001001.

[8] Asaduzzaman, A., Gummadi, D., & Waichal, P. (2015, April). A promising parallel algorithm to manage the RSA decryption complexity. In SoutheastCon 2015 (pp. 1-5). IEEE.

[9] Tan, X., & Li, Y. (2012, March). Parallel analysis of an improved RSA algorithm. In Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on (Vol. 1, pp. 318-320). IEEE.

[10] Saadeh, M., Saadeh, H., & Qatawneh, M. (2016). Performance Evaluation of Parallel Sorting Algorithms on IMAN1 Supercomputer. International Journal of Advanced Science and Technology, 95, pp. 57-72.

[11] Bruce, S. (1995). Applied cryptography: protocols, algorithms, and source code in C, Second Edition. John Wiley & Sons.

[12] Ananth, G., Anshul, G., George, K,. & Vipin, K. (2003). Introduction to Parallel Computing, Second Edition, Addison Wesley.

[13] Kasahara, H., & Narita, S. (1984). Practical multiprocessor scheduling algorithms for efficient parallel processing. IEEE Transactions on Computers, 33(11), pp. 1023-1029.

[14] Qatawneh Mohammad. (2005). Embedding Linear Array network into the Tree-Hypercube Network. European Journal of Scientific research, 10(2), pp. 72-77.

[15] Mohammad Qatawneh, Ahmad Alamoush, Ja’far Alqatawneh. (2015). Section Based Hex-Cell Routing Algorithm (SBHCR). International Journal of Computer Networks & Communications (IJCNC), 7(1), pp. 167-177.

[16] Azzam Sleit, Wesam AlMobaideen, Mohammad Qatawneh, Heba Saadeh. (2008). Efficient processing for binary submatrix matching. American Journal of Applied Sciences, 6(1), pp. 78-88.

[17] Mohammad Qatawneh (2011). Embedding Binary Tree and Bus into Hex-Cell Interconnection Network. Journal of American Science. 7(12), pp.367-370.

[18] Qatawneh Mohammad, Hebatallah Khattab 2015. New Routing Algorithm for Hex-Cell Network. International Journal of Future Generation Communication and Networking. 8(2), pp. 295-306.

[19] Mohammad Qatawneh 2016. New Efficient Algorithm for Mapping Linear Array into Hex-Cell Network. International Journal of Advanced Science and Technology. 90, pp. 9-14.