e-ISSN: 2289-6589

AN IMPROVED RSA CRYPTOSYSTEM BASED ON THREAD AND CRT

***Saheed Yakub Kayode[1], Gbolagade Kazeem Alagbe**2

[1]Department of Physical sciences
Al-Hikmah University, Ilorin, Nigeria.

[2]Department of Computer Science
Kwara State University, Malete.Nigeria

*Corresponding author's email: yksaheed@alhikmah.edu.ng

Submission date: 15 July 2017 Accepted date: 30 Sept 2017 Published date: 30 Nov 2017

Abstract

This paper proposes an efficient approach to improve Rivest Shamir Adleman (RSA) algorithm using parallel technique. RSA public-key cryptosystem has been the most popular and interesting security technique for majority of applications such as internet protocols, secure internet acces and electronic commerce. The main bottleneck of RSA algorithm is that they are slower compared to symmetric cryptography key alternatives simply because of their foundation in modular arithmetic. Hence, how to make a more efficient and faster implementation of Rivest Shamir Adleman algorithm (RSA) is a great concern to researchers in the field of cryptography. In this paper, we propose a parallel implementation technique using Chinese Remainder Theorem and thread on encryption and decryption operation in RSA when files are to be encrypted and decrypted. Also, in our method, the key size is extended from 1024 bits to 2048 bits in length to provide a good level of security, since 1024 bits key size is no more appropraiate for protecting data. We use a parallel technique that divides RSA power process into seperate threads and employs the use of Chinese Remainder Theorem (CRT) to decrease the time required for both encryption and decryption operation. Java programming language is used to implement the algorithm. Experimental results indicate that as the thread level increases, the encryption and decryption time, which is the most time consuming operation, decreases which shows an improved speed of the RSA cryptosystem.The proposed implementation has a great potential to effectively deal with the RSA algorithm slow speed.

Keywords: RSA; thread; CRT; key size; encryption; decryption;

1.0 INTRODUCTION

The necessity for a secured communication on the internet is more profound than ever, recognizing the fact that the conduct of almost all business transactions and personal matters are carried out today through computer networks (Saheed & Gbolagade, 2016). Also, the increase in the demand for secure communications between different parties on the internet and use of electronic commerce have led to more demand of high-speed and reliable security products (Liu, Ma, Tong & Cheng, 2004). RSA stands for Ron Rivest, Shamir Adi and Adleman Leonard who are the inventors of the RSA cipher. It is a type of public key cryptography algorithm which was first made public in the year 1987 and which is the most popular, attractive and one of the widely used public key cryptography algorithm. Before the encryption process, we need to generate the keys which are public keys and the other known as private keys. A key usually uses very big prime number of 1024 bit keys which is about

300 digits long. The time it takes to factorize this number creates a high level security for this algorithm; and it takes millions of years to crash. The length of key that is common recently is 1024.However, the longer the length of the key the more guaranteed is the security provided by the RSA cryptosystem. As discussed in (Barker et al., 2007), the RSA cryptosystem with the range of key size of 1024 bits neither provides nor guarantees a sufficient and strong security level between years 2011 and 2019. Thus, the size of the key should be increased to 2048 or higher bits in length.
RSA (Rivest, Shamir & Adleman, 1978) is the most popular, widely used and deployed asymmetric public-key cryptosystem. It is used in securing web traffic and some wireless devices and email (Saheed & Gbolagade, 2017a). RSA cryptosystem, encryption operation and decryption processes are computationally expensive and heavy because their bases in modular exponentiation in very large numbers are needed. Since RSA has its foundation in arithmetic modulo operation with big numbers, its operation can be slow in a constrained environments. An example is when a web server is loaded heavily; RSA decryption process substantially decreases the number of secure socket layer (SSL) requests per second and is more than the computer server can deal with. The need for information security has become more widespread to a greater extent during these days. Parallelization of public key algorithms could be very useful for a high and reasonable level of security system and can save a lot of computation time. If combined, the public key cryptosystem will be more efficient and effective for those kinds of system. In this paper, we introduce a parallel architecture based on thread level to improve the speed of RSA cryptosystem. The RSA public key algorithm has been used for many years without any security compromise. Although, different numbers of asymmetric key cryptography algorithms have been developed after RSA cryptosystem, it is still being used (Damrudi & Ithnin, 2013; Kayode & Alagbe, 2017b) and deployed in many applications. The justification is in its security value and implementation ease. According to (Afolabi & Atanda, 2016), the RSA cryptosystem can be used for public key encryption and digital signatures.

Recently, Saheed and Gbolagade (2017a) proposed an improved RSA algorithm based on residue number system (RNS). In this paper, two optimizations are investigated to make the RSA cryptosystem operation easy. Younis, Fadhil and Jawad (2016) proposed a review on the parallelization of the RSA algorithm and Chinese Remainder Theorem to improve the decryption process. Also, Asaduzzaman, Gummadi, and Waichal (2015) presented an approach to deal with the RSA Decryption complexity. In this work, the effect of compute unified device architecture (CUDA) and pthread on decryption in RSA is explored by homomorphic encryption. Saxena and Kapoor (2014) suggested a novel Parallel RSA algorithm based on repeated square-and multiply method. Other researchers who worked lately on RSA cryptosystem are Fan et al., (2010) Li et al., (2010), and Liu et al., (2010).

The rest of this paper is organized as follows; section two discusses the methodology. Section three highlights the system architecture and implementation, while in section four, the results are presented.

2.0 Material and methods

2.1 Proposed methodology

The RSA encryption/decryption process is applied into a parallelism form in order to reveal the efficient way to implement the RSA cryptosystem algorithm. As for both the encryption and decryption processes, the number of threads to be used in the process is selected in order to enhance its speed. The Software application for the project is implemented using java programming language;thereby, taking advantage of java's extensive library of security classes under the "java.security" package and the Cipher class under the "javax.crypto" package. The application is implemented using four main classes, namely, PairGenerator, RSA, Worker and RSAAPP classes.

2.2 Parallel RSA implementation

The details of the parallel implementation are given in Figure 3.1, which include public class that implements the algorithm.

2.3 Algorithm Procedure

Before the message, data or file is encoded or encrypted, the public key and private key should be generated. This process is done between the user and the internet service provider.

Key generation steps:

2.3.1 Firstly, select two distinct prime numbers p and q. For the purpose of security, the integers p and q have to be chosen randomly with the length in bit.

2.3.2. Secondly, compute n = p * q.

2.3.3. Then, find the Euler's totient function, Ø(n) = (p-1) * (q-1).

2.3.4. Choose a random integer e, $1 < e < \theta$, such that the gcd $(e, \theta) = 1$. Now, e is used as Public-Key exponent (e).

2.3.5. Now, use the extended Euclidean algorithm to find d as follows: d = e-1(mod Ø(n)) that is, d is the multiplicative inverse of e, such that $ed \equiv 1 \pmod{\theta}$.

2.3.6. d is the Private-key exponent, such that dx e = 1 mod Ø(n).

2.3.7. The Public Key is made up of modulus (n) and the public exponent (e) which is public. That is, (e,n).

2.3.8. The Private Key is made up of modulus (n) and the private exponent (d), that is private exponent, which must be kept secret that is, (d, n).

2.4 Encryption

Encryption means enciphering message, data, file or plain text into cipher text (file). The process step is as follow:

2.4.1 Send the Public-Key to the receiver who wants to store the message, data or file with him or her.

2.4.2 The user message, data or file is now matched to an integer value by using an agreed protocol known as padding scheme.

2.4.3 The message, file or data is encrypted and the result of encryption is the cipher text (data) C which is given as $C = m^e \pmod{n}$.

2.4.4 This result of encryption which is known as the encrypted data is now stored.

2.5 Decryption

Decryption means deciphering the encrypted data, file or message to the original meassage or file. The key size of the RSA cryptosystem decryption exponent (d) and modulus (n) is very significant; simply because the complexity is dependent on it.

3.0 System Architecture and Implementation

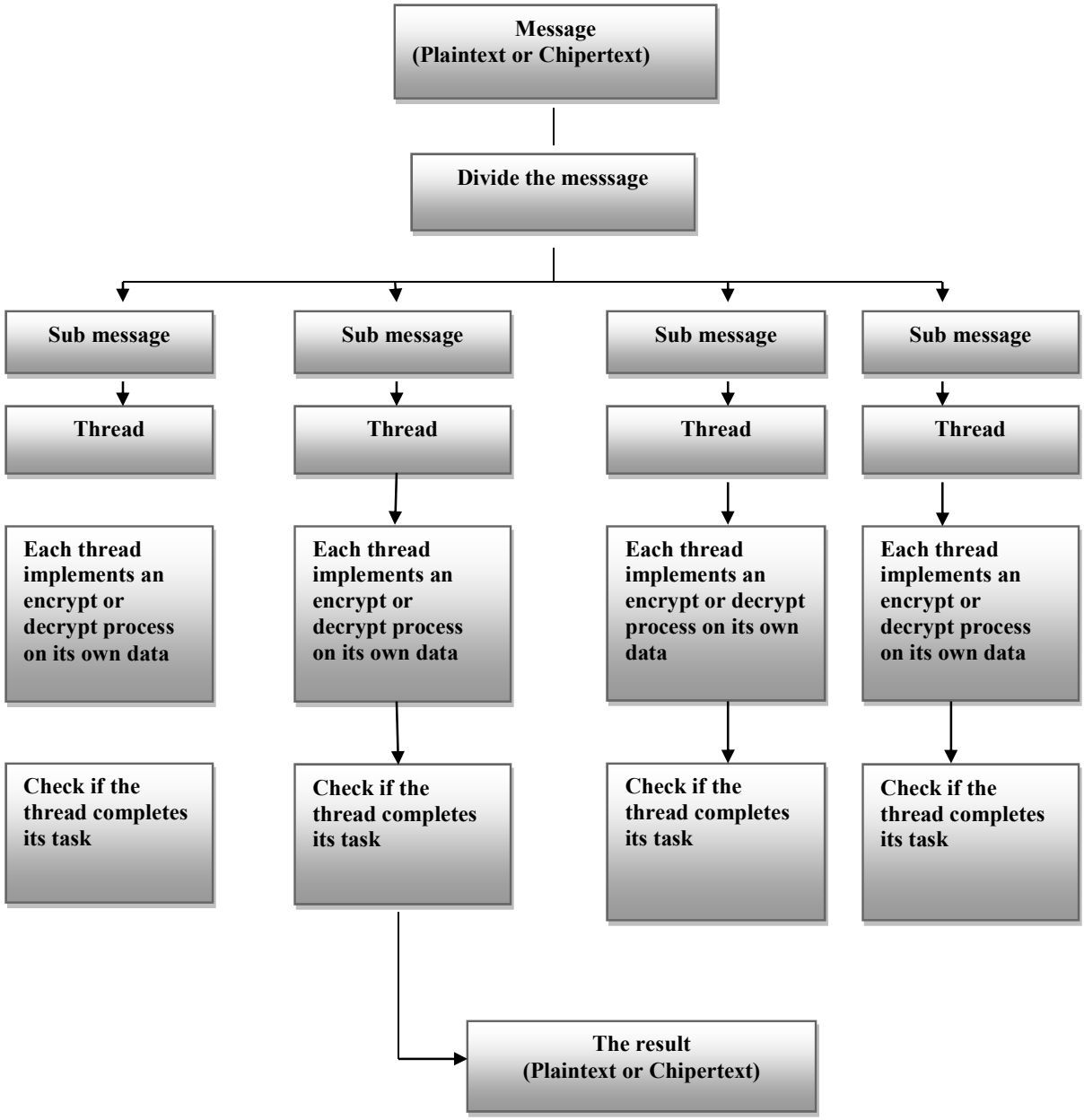

Figure 3.1 System Architecture

4.0 Results and discussion

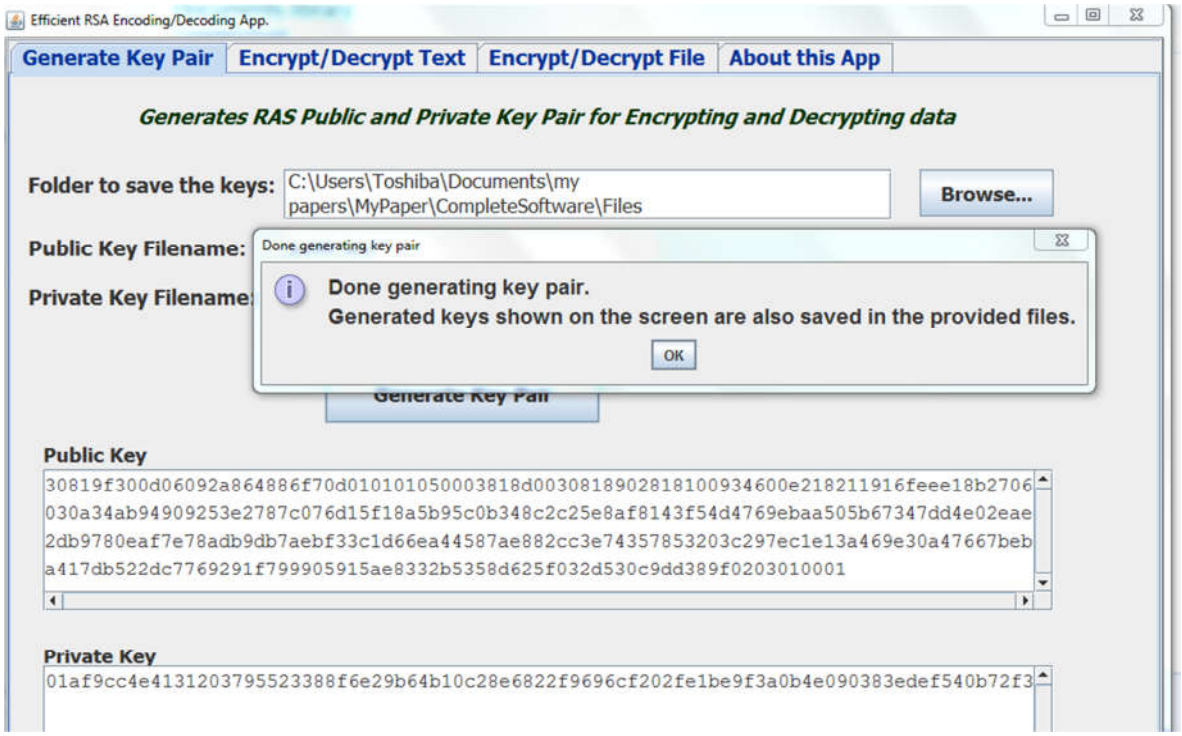

Figure 4.1 Generate key tab

The generate key in figure 4.1 provides the interface to generate both the public-key (n, e) and private key (n, d). The public key (n, e) is to be transmitted to the user before the actual encryption can take place and the private key (n, d) is to be used by the receiver to decrypt the enciphered message, data or file.

4.1 Encrypt /Decrypt file Tab

As shown in the following figures 4.2 and 4.3, this tab allows the user to select a file and have it either encrypted or decrypted depending on which radio button is selected. For encryption, the user is expected to first browse and locate the file containing the public key which should have been created from the "Generate Key Pairs" tab earlier.

For the decryption, the user is expected to provide the private key. The default filenames and folder for the keys are automatically suggested; but the user can provide different ones.

The decryption is done on a previously encrypted file. For example, the encrypted file from figure 4.2 is in the input file field for decryption. Upon clicking the "Decrypt file" button, the original file is decrypted back.

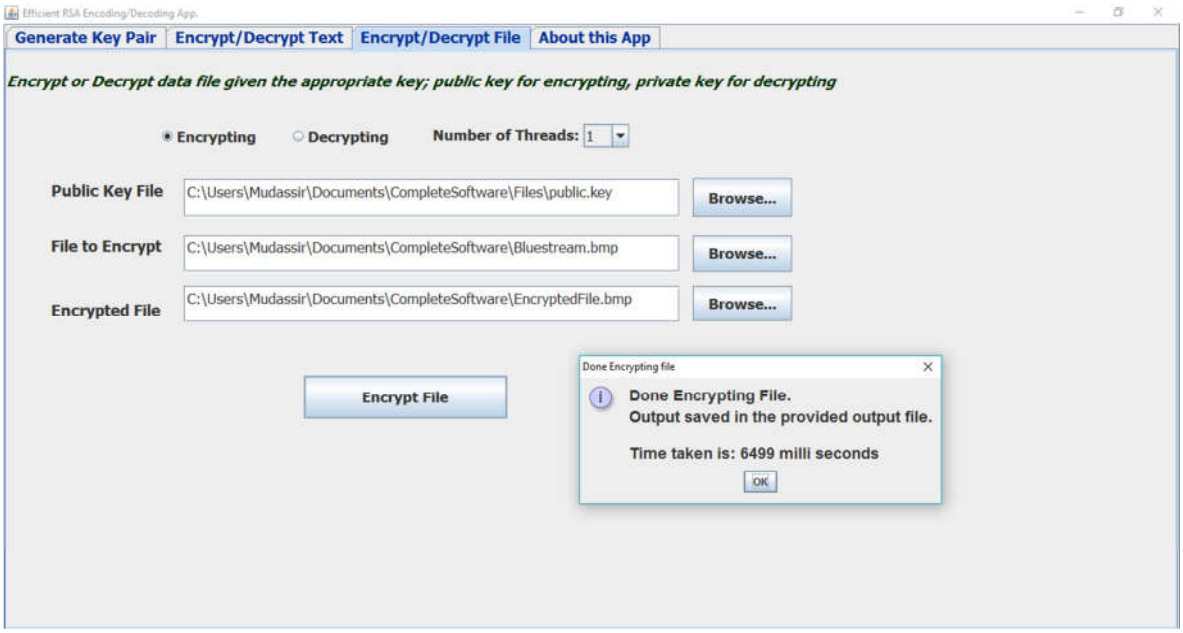

Figure 4.2 Encryption time of file size 2.25MB

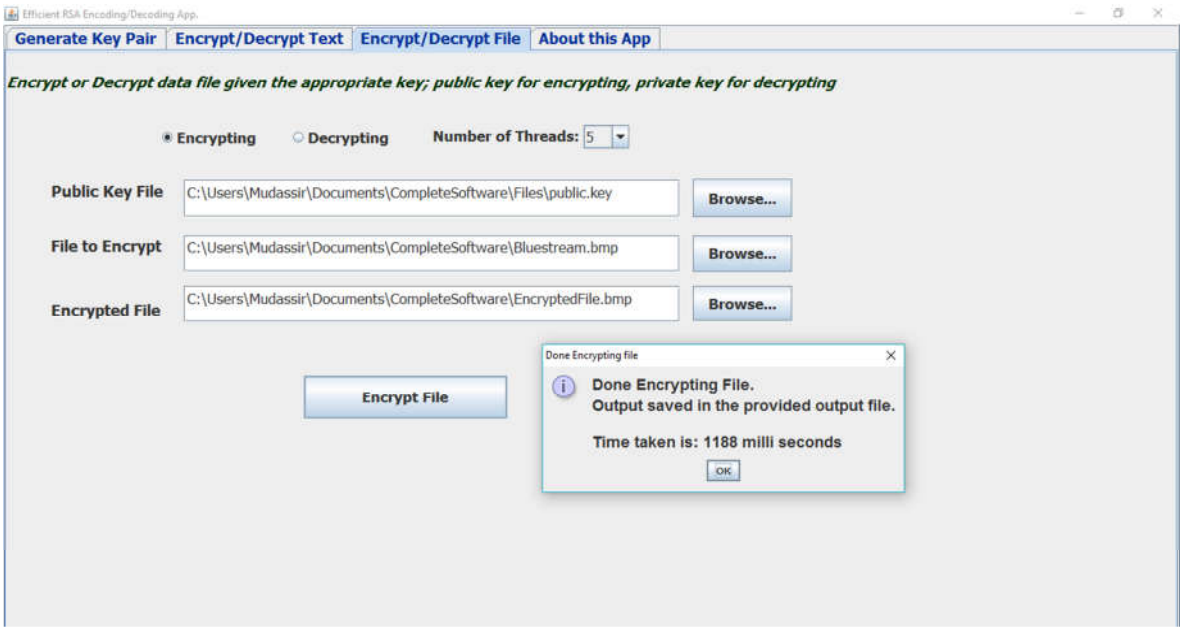

Figure 4.3 Encryption time of file size 2.25MB

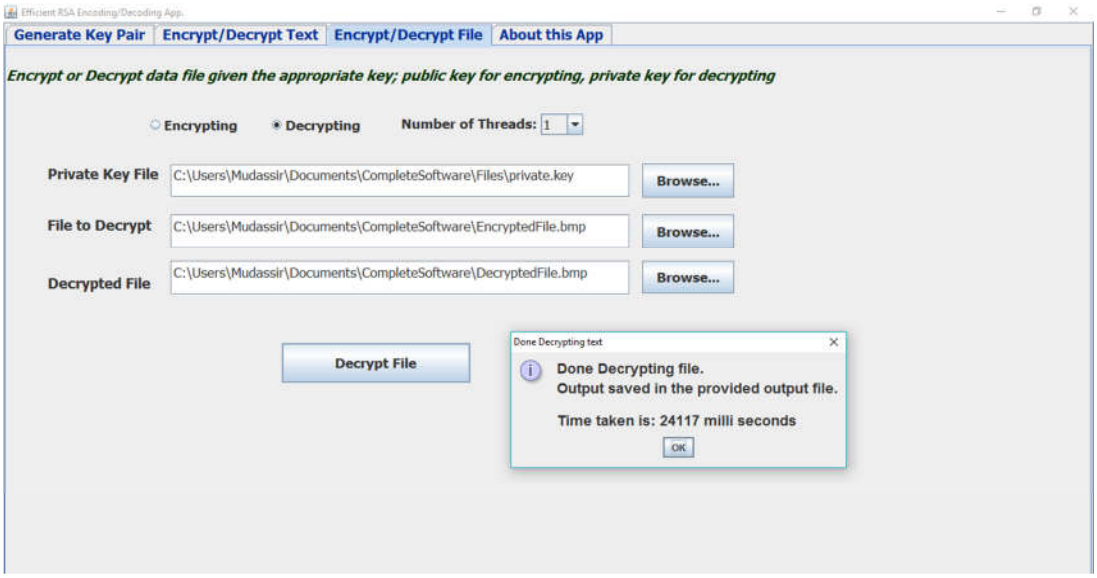

Figure 4.4 Decryption time of file size 2.25MB

Table 4.1 illustrates the time taken for both encryption stage and decryption process. It was observed that as the thread level increases, the encryption and decryption, which is the most time consuming operation, decreases.

Table 4.1 Encryption/decryption time of file size = 2.25MB

Thread Levels	Encryption time(milliseconds)	Decryption time (milliseconds)
Thread 1	6499	24117
Thread 2	3068	11825
Thread 3	2080	9905
Thread 4	1424	9081
Thread 5	1188	8797

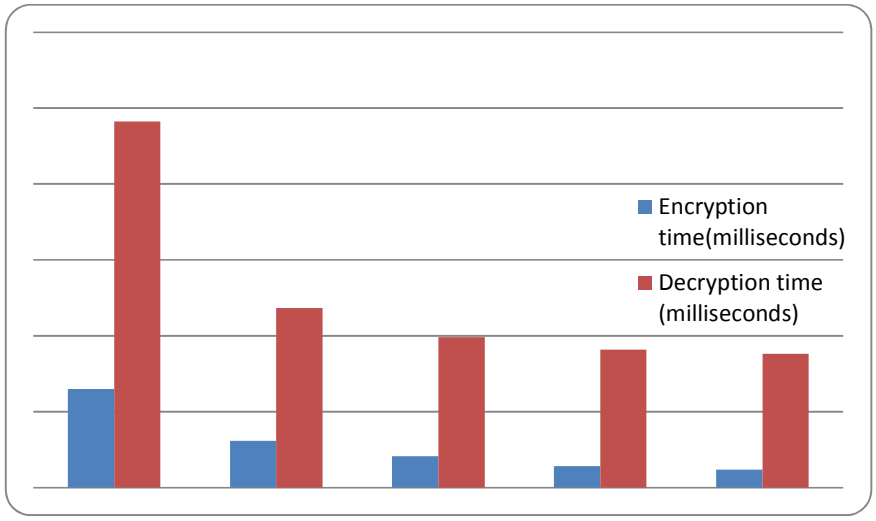

Figure 4.5 Encryption and decryption time for the thread levels

5.0 Conclusion

This paper proposes a parallel implementation of RSA algorithm using thread architecture. Sequential implementation has been reported in the literature but RSA algorithm implemented sequentially does not provide sufficient speed. We propose a faster implementation of RSA algorithm in this paper based on parallelism concept. The parallelism concept used is thread architecture which is implemented using java programming language on a different number of threads. As the thread level increases, the encryption and decryption times decrease and this shows an improved speed of the RSA algorithm cryptosystem. RSA algorithm has been one of the successful asymmetric cryptographies. It can be used to provide and gurantee security in web traffic, cloud and security products. Its limitation is that it is very slow in a constrained environment. Therefore, in this paper, we provide a technique using thread and Chinese Remainder theorem (CRT) to improve the speed of the RSA cryptosystem. The future work will be to extend the thread level and observe if there would be a significant difference in both encryption and decryption processes of the RSA cryptosystem.

References

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.

Afolabi, A.O. & Atanda, O.G**.** (2016). Comparative Analysis of Some Selected Cryptographic Algorithms. Computing, Information Systems, Development Informatics & Allied Research Journal. (Vol 7, No 2, pp 41-52).

Barker, E. B., Barker, W. C., Burr, W. E., Polk, W. T., & Smid, M. E. (2007). Sp 800-57. recommendation for key management, part 1: General (revised).

Fan, W., Chen, X., & Li, X. (2010, November). Parallelization of RSA algorithm based on compute unified device architecture. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on* (pp. 174-178). IEEE.

Li, Y., Liu, Q., & Li, T. (2010, April). Design and implementation of an improved RSA algorithm. In *E-Health Networking, Digital Ecosystems and Technologies (EDT), 2010 International Conference on* (Vol. 1, pp. 390-393). IEEE.

Damrudi, M., & Ithnin, N. (2013). Parallel RSA encryption based on tree architecture. *Journal of the Chinese Institute of Engineers*, *36*(5), 658-666.

Younis, M. I., Fadhil, H. M., & Jawad, Z. N. (2016). Acceleration of the RSA Processes based on Parallel Decomposition and Chinese RemainderTheorem. International Journal of Application or Innovation in Engineering & Management. (Vol. 3, Issue 1, pp. 12-23).

Liu, Q., Ma, F., Tong, D., & Cheng, X. (2004, July). A regular parallel RSA processor. In *Circuits and Systems, 2004. MWSCAS'04. The 2004 47th Midwest Symposium on* (Vol. 3, pp. iii-467). IEEE.

Liu, Q., Li, Y., & Hao, L. (2010, August). On the design and implementation of an efficient RSA variant. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on* (Vol. 3, pp. V3-533). IEEE.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120-126.

Saheed, Y.K., and Gbolagade K.A., (2016). "Efficient Image Encryption based on the moduli set $\{2^{n-1}, 2^n, 2^{n+1}\}$"**.**Al-Hikmah Journal of Pure & Applied Sciences Vol.3 (2016): 15-21.

Saheed Y. K., and Gbolagade K. A**.** (2017a). An Improved RSA algorithm based on Residue Number System. In: Proceedings of the 13th Nigeria Computer Society International Conference, 2017.Vol. 28. Pp 86-91.

Kayode, S.Y., & Alagbe, G.K. (2017b). Efficient RSA Cryptosystem Decryption Based on Chinese Remainder Theorem and Strong Prime. Anale.SeriaInformatică. (Vol. XV, fasc. 2 – 2017, pp.43-47).

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