An Improved RSA Algorithm for Enhanced Security

Kannan Balasubramanian, M. Arun, K. R. Sekar

Abstract: The security of the Rivest-Shamir-Adelman (RSA) public key algorithm depends on the difficulty of factoring the modulus calculated by multiplying two large prime numbers. The usefulness of the RSA public key algorithm lies in using one key for encryption and another key for decryption. However, a poor choice of the keys used in encryption and decryption can affect the security of the RSA cryptosystem. Many proposals have been made to modify the RSA cryptosystem in such a way that the attacks on the RSA cryptosystem can be overcome. In this article, we propose concealing the publicly disclosed parameters, the encryption key and the common modulus of the RSA cryptosystem by altering the values sent publicly. The values sent publicly are different from the one used in the algorithm which conceals the actual parameters to others. The implementation of this idea uses two different algorithms and randomly choosing between the algorithms. The choice of the algorithm is done using a random number generator and this choice of the algorithm has to be communicated so the decryptor uses the correct algorithm to decrypt the encrypted data. Finally we explore a faster way to implement the modular exponentiation algorithm used in the RSA encryption and decryption.

Key words: Public Key Cryptosystem, Encryption and Decryption Keys, Common Modulus, RSA cryptosystem, Factoring attack, PRNG, TRNG.

I. INTRODUCTION

The RSA algorithm is known to have several weaknesses [1]. Some of the weaknesses are due to the common modulus attack, the blinding attack, the small encryption exponent "*e*", the small decryption exponent "*d*", the forward search attack, timing attack, attack due to the multiplicative properties, the cycling attack, attack due to message concealing, the faulty encryption attack and factoring the public key attack. The following methods are adopted to enhance the security of the RSA algorithm.

1. Choosing two very large prime numbers approximately of the same size,
2. Choosing the encryption key "*e*" and the decryption key "*d*" approximately of the same size.
3. Encrypting the common modulus "*n*" and the encryption key "*e*" by a common key known to the encryptor and the decryptor.
4. Modifying the RSA algorithm to overcome the timing attacks on the algorithm.

In this paper we follow the fourth approach of modifying the RSA public key encryption algorithm so it is not vulnerable to timing attacks. We adopt the approach used in [2] and [3] to modify the RSA algorithm. Since the proposed modification to the RSA algorithm also can be known to the attacker, the attacker can try the modified algorithm as well. Hence we propose that the encryptor and the decryptor adopt more than one variation of the RSA algorithm and randomly choose the algorithm by methods known to the encryptor and the decryptor based on the parameters exchanged between them. Since the algorithm chosen cannot be easily guessed by the attacker, by randomly choosing between two variations of the RSA algorithm, the cryptosystem can be made secure. In the next section, we survey the many variations of the RSA algorithm to protect against the attacks

II. RELATED WORK

The factorization attack where attackers try to factorize the public modulus of the RSA cryposystem is the well known attack on the RSA Cryptosystem [4][5]. Other attacks are due to the choice of encryption key and decryption keys. A small encryption key or a small decryption key can be easily attacked using brute force guessing of the keys. The common modulus attack refers to recovering the plaintext when it is encrypted with two different keys using the same modulus [6]. Other attacks relate to whether the attacker can gain partial knowledge about the plaintext and timing attacks on the RSA cryptosystem. Many proposals have been made to protect the RSA cryptosystem against the attacks gaining knowledge about the decryption key and gaining knowledge about the plaintext transmitted. To overcome the attacks on the RSA algorithm, many modifications have been proposed. One way is to increase the number of key pairs used in the RSA algorithm as proposed in [7]. This algorithm uses two key pairs as against one pair as in the regular RSA algorithm. The encryption applies the public key1 first followed by public Key2 the decryption uses the private key2 and private key1 with two different moduli n_1 and n_2. The algorithm proposed in [8] modifies the RSA algorithm so the modulus is a product of four prime numbers. It remains to be studied whether the use of four prime numbers will increase the difficulty of factorization of the modulus.

Manuscript received on 22 September 2022 | Revised Manuscript received on 01 October 2022 | Manuscript Accepted on 15 November 2022 | Manuscript published on 30 November 2022.

***** Correspondence Author (s)

Dr. Kannan Balasubramanian*, School of Computing, SASTRA University, Thanjavur, India. Email: kannanb@cse.sastra.edu

M. Arun, Associate Professor, Department of Computer Science and Engineering, Veltech Rangarajan Dr. Sagunthala R& D Institute of Science and Technology (Deemed to be University), Chennai. (Tamil Nadu), India.

Dr. K. R. Sekar, School of Computing, SASTRA University, Thanjavur, India. Email: sekar_kr@cse.sastra.edu

Another variant uses three prime numbers instead of four but substantially modifies the encryption and decryption procedures of the RSA algorithm [9]. The algorithm proposed in [10] uses three prime numbers in but the encryption and decryption procedures are not modified. The algorithm in [11] also uses four prime numbers but in addition, it modifies the encryption key by applying the algorithms used in DES symmetric encryption algorithm. The modification proposed in [12] discusses choosing the prime numbers large enough, so factorization is made difficult. The modification proposed in [13] proposes to prevent factorization attack on the modulus *n* by choosing another value X, using which encryption key and decryption keys are calculated. Attacker will not be able to guess the decryption key by factorizing X since X is not the product of *p* and *q*. The proposed algorithm in [14] modifies the encryption by combining RSA encryption and Elgamal public key encryption. Moreover to prevent factorization attack three prime numbers are used in the calculation of *n*.
The implementation proposed in [15] focusses on computing using very large prime numbers in software. The algorithm proposed in [16] suggests choosing an alternate encryption key using the nearest neighbour approach and using distance 'r' among the the possible keys.

III. THE PROPOSED ALGORITHM

The two RSA variations we use in our implementation are described below.
Algorithm 1 (the public key *e* is hidden):

1. Generate two large random prime numbers (*p*,*q*)
2. Calculate the modulus $n=p*q$
3. Calculate $\varphi(n)) = (p-1)(q-1)$
4. Select e with the following condition
5. Select $f = e*2 + 1$
6. Select *d* such that $ed \equiv 1 \; mod \; n$
7. Make (*f*,*n*) as the Public Key
8. Make the decryption key as (*d*,*n*)
9. Encryption C= $M^{(f-1)/2} \; mod \; n$
10. Decryption D= $C^{d} \; mod \; n$

The Algorithm1 hides the encryption key "*e*" but sends the common modulus in the clear

The Algorithm 2 hides the common modulus "*n*" but sends the encryption key in clear. The proposed modification to RSA algorithm depends on randomly choosing between the two algorithms. We assume only two parties are involved and the initiator of the encryption algorithm randomly chooses between one of the above algorithms and sends the parameters of the algorithm. Assume that User A is planning to use the RSA cryptosystem with User B. User A initiates a Request message with User B with a nonce n_A. User A and User B can agree on Algorithm 1 or Algorithm 2 based on the nonce generated if this is nonce is truly random . In practice, the random numbers generates are based on a starting value (i.e. seed) which has to be chosen by the User A. To overcome this problem, we can allow User B to send a random number n_B to A. User A now generates another random number n_C using the nonce received from User B and sends to B. Based on the random number generated User A and User B agree on Algorithm 1 if the random number n_C is odd and vice versa. This exchange of messages is shown in Fig 1.

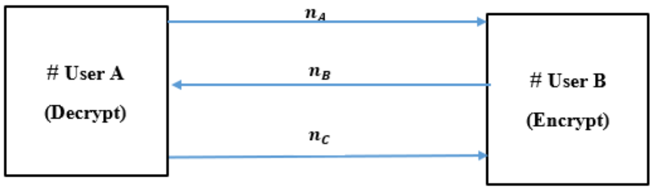

Figure. 1 Protocol using Three Message Exchanges for Agreement on the Algorithm to be used.

The strength of the above algorithm is that the attacker now has to guess both User A's starting value and User B's starting value to know which algorithm is being used. The chance of the user guessing both User A and User B is relatively less if both users change the starting value often. This algorithm can further be improved if the random nonces chosen are based on True random number generators instead of Pseudo random number generators. Most of the Programming Languages including Java and C include a system call to obtain the current time which can be used as the seed value. The use of current time improves the random value obtained as a result the choice of the algorithm based on the nonce n_C can be made be truly random. The protocol can be simplified if the current time is used as the seed value for the random number generator. Instead of three message exchanges, only one message is required which is the random number n_A generated by User A. This modification is shown in Fig.2. The implementation of the algorithm uses two different moduli n_1 and n_2 and the encryption key uses n_1 and the decryption key uses the modulus n_2.

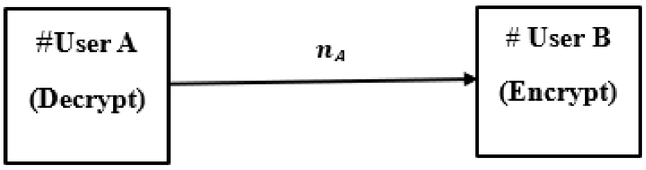

Figure. 2. Protocol using Only One Message for Algorithm Selection.

IV. IMPLEMENTATION OF THE PROPOSED ALGORITHM

The following steps are followed in the improved algorithm.

1. Let "*m*" is the message to be transmitted.
2. User A generates the true random number n_A based on the algorithms for generating true random numbers specified in [21].
3. User A sends the random number n_A to User B over a secure channel.
4. User A and User B execute Algorithm 1 or Algorithm 2 based on whether the random number is even or odd.

The above algorithm is easy to implement. The only drawback in the algorithm is that the random number selected by A is sent unencrypted to User B which introduces the possibility of a third party listening or modifying the random number over the channel.

To avoid the manipulation of the random number, the random number can be encrypted using a pre-shared key between User A and User B. Including these steps, the above algorithm can be modified as shown below.

1. The protocol involves User A and User B calculating the secret-key "K" through the Diffie-Hellman Key Exchange method.
2. User A generates the true random number n_A based on the algorithms for generating true random numbers specified in [17][20][21].
3. User A sends the random number n_A to User B encrypted by key "K" channel.
4. User A and User B execute Algorithm 1 or Algorithm 2 based on whether the random number is even or odd

A. Implementation of the modular exponentiation algorithm

The RSA algorithm described above involves calculating m^a mod n where a is the exponent as described in Algorithm 1 and Algorithm 2. The modular exponentiation is usually implemented by the square and multiply algorithm by reading the bits of the exponent from left to right or from right to left. This implementation of the square-and-multiply algorithm is shown in Fig.3.

```
int binpow ( int a, int b, int m) {
   a %= m;
   int  res = 1;
   while (b > 0) {
      if (b & 1)
         res = res * a % m;
      a = a * a % m;
      b >>= 1;
   }
   return res;
}
```

Figure 3. The square-and-multiply algorithm for Modular exponentiation in C

As opposed to the square-and-multiply algorithm, a faster method for calculating m^a mod n is by calculating

$$m^{2^{i-1}} m^{2^{i-2}} \dots \dots m^{2^0}$$

where i is the number of bits in the binary representation of the number a. We can note that all the above values may not be required since only the values with the corresponding binary bit set to 1 will be required. For example to calculate 347^{135} mod 547 we can see that

$347^{2^7} 347^{2^2} 347^{2^1} 347^{2^0}$ mod 547 which is obtained from the binary representation of 135 as $(10000111)_2$ This also can be represented as (7_{347} 2_{347} 1_{347} 0_{347}) mod 547 using a short notation[22].

We observe that

$0_{347} = 347$

$1_{347} = (347 \times 347) \bmod 547 = 69$

$2_{347} = (69 \times 69) \bmod 547 = 385$

$3_{347} = (385 \times 385) \bmod 547 = 535$

$4_{347} = (535 \times 535) \bmod 547 = 144$

$5_{347} = (144 \times 144) \bmod 547 = 497$

$6_{347} = (497 \times 497) \bmod 547 = 312$

$7_{347} = (312 \times 312) \bmod 547 = 525$

Finally the required value is computed as 347 x 69 x 385 x 525 = 305 What we have done is effectively squared the base value and stored them and the final required value is found by multiplying the necessary values. The required number of multiplications equals the number of bits in the binary representation of the exponent and the number of ones in the binary representation of the exponent. After every multiplication the reduction to the modulus value is done. Reduction can be efficiently done by repeatedly subtracting the modulus value until the value is less than the modulus[18][19][20].

Another method for exponentiation [22] is by noting that a^b mod n can be calculated as

$a^b = (a^2)^{b/2}$ if b is an even number. If b is an odd number we can calculate $a^b = a^{b-1}\, a$ where b-1 is even. After every exponentiation, the power is halved and in log n steps will reach 1 if b is a power of 2. The program code for implementing this is shown in Fig. 4.

```
int mod_expo ( int base, int  exponent, int modulus)
{int result = base;
   while (exponent > 0)
   {
      result = (result * result) % modulus;
         exponent = exponent >> 1;
   }
   return result;
}
```

Figure. 4 Program to perform modular exponentiation when the exponent is a power of two in C

One can use the above method instead of the square-and-multiply algorithm to calculate the modular exponentiation. Although the number of exponentiations will be large this method will be efficient when the exponent is large and has few ones and remaining zeroes.

B. True random number generation

In Linux, the /dev/random and the/dev/urandom device can be used to generate true random numbers. The Fig. 5 shows an implementation of the true random number generation in Linux.

```
#include <stdio.h>
int main(){
   int randomvalue;
   FILE *fpointer;
   fpointer = fopen("/dev/urandom", "rb");
  fread(&randomvalue,sizeof(int),1,fpointer);
   fclose(fpointer);
   printf("%d ",randomvalue);
   return 0;  }
```

Figure. 5. True random number generation in Linux using the /dev/urandom device in C Programming Language

V. ANALYSIS OF THE PROPOSED ALGORITHM

Since the parties involved in the public key cryptosystem reveal either the encryption key or the common modulus, an attacker capturing the public key for factoring the modulus "n" has 50% chance of capturing the correct modulus over the channel.

Hence the attacker spends twice the amount of time in the factorization attack on the common modulus. The attacker capturing the encryption key "*e*" to launch a chosen plaintext attack has 50% chance of succeeding since the encryption key is hidden 50% of the time by the algorithm. The attacks on RSA can be overcome by the above algorithm with the use of the true random number generator. One way to generate such random number is by using the /dev/urandom device in the Linux random which relies on the entropy provided by the hard disk reads and writes to generate the random number [21].

VI. CONCLUSION

This research article tried to overcome the vulnerability in the RSA encryption of having the send the key "*e*" and the modulus "*n*" in the clear by using a novel method of hiding the key "*e*" and the modulus "*n*". It uses a method to choose from two different algorithms Algorithm 1 which hides the encryption key and Algorithm 2 which hides the modulus. The choice between the two algorithms is made using a true random number generator provided in the Linux /dev/random device. By using this protocol, the time taken for the attacker to crack the cryptosystem will be increased and hence this method effectively the security of the RSA algorithm. The implementation of modular exponentiation introduced another procedure which will perform better when the exponent has few ones.

REFERENCES

1. Bakhtiari,M., & Maarof,M.A, Serious Security Weakness in RSA Cryptosystem, *Int. J. of Computer Sci. Issues*, Vol.9, Issue 1, No.3, January 2012.
2. Intila,C, Gerardo,B & Medina, R, A study of public Key "*e*" in RSA algorithm,https://iopscience.iop.org/article/10.1088/1757-899X/482/1/012016/pdf
3. Intila,C, Gerardo B & Medina, R., Modified RSA algorithm based on Key Generation, *Proceedings of The IIER International Conf*., Manila, Philippines, 27th -28th June 2018.
4. Boneh,D., Twenty years of attacks on the RSA Cryptosystem, *Notices of the AMS*, 203-213, 1999.
5. Mumtaz,M. & Ping,L., Forty years of attacks on the RSA Cryptosystem: A Survey, *J. of Discrete Mathematical Sciences & Cryptography*, 22(1), 9-29, 2019 [CrossRef]
6. Landau,S., A Brief Summary of attacks on RSA, https://www.rose-hulman.edu/class/ma/holden/Archived_Courses/Math479-0304/resources/attacks-rsa/
7. Abudin,J., Keot,S.K., Malakar,G., Borah,N.M., & Rahman.,M., Modified RSA Public Key Cryptosystem Using Two Key Pairs, *Int. J. of Computer Sci. and Information Technologies*, Vol. 5.(3), 2014. 3548-3550.
8. Nivetha,A., Preethy Mary S. & Santosh Kumar S., Modified RSA Encryption Keys using Four Keys, *Int. J. of Engineering Research and Tech*., 3(7). 2015.
9. Zaid,M.M.A. & Hassan,S., Lightweight RSA Algorithm Using Three Prime Numbers, *Int. J. of Engineering and Tech*., 7(4.36) 293-2956, 2018. [CrossRef]
10. Chowhan,S.S., & Jaju S.A., A Modified RSA algorithm to enhance Security for Digital Signature. *In Proceedings of Int. Conf. and workshop on Computing and Communication*, IEEE, Vancouver BC, Canada, 2015.
11. Khanum,S., Sharma,B., & Beniwal,G., Hybrid Public Key Cryptosystem combining RSA and DES algorithm, *Int. J.of Innovations in Engineering and Tech.*, 7(3), 466-471,2016
12. Sarjiyus,O, Enhancing RSA Security Capability using Public Key Modification, *Int. J. of Research and Scientific Innovation* , 7(9), September 2020.
13. Sahu,J, Singh,V,Sahu,V,& Chopra,A, An enhanced Version of RSA to increase the Security, *J. of Network Communication and Emerging Technologies*, 7(4),April 2017
14. Goyal,P., & Kumar,D., Implementation for Enhancement of Computation Technique By Combining Enhanced RSA and El-Gamal Public Key Cryptosystems, *Int. J. of Computer Sci. and Tech*., 5(2), April-June 2014.
15. Obaid,T.A.S., Study A Public Key in RSA Algorithm, European J. of Engineering Research and Sci., 5(4), 2020. [CrossRef]
16. Abu-Dawas,M.A, & Hussain,A.K., Enhancement of RSA Scheme using Agreement Secure Information for Nearest Parameters, *Int. J. of Computer and Information Technology*, 4(2), March 2015.
17. Patidar,R.,& Bhartiya,R, Implementation of Modified RSA Cryptosystem Based on Offline Storage and Prime Number, *Int. J. of Computing and Tech*., 1(2), March 2014. [CrossRef]
18. Patel,S.R., & Shah,K., Security Enhancement and Speed Monitoring of RSA Algorithm, *Int. J. of Eng. Development and Research*, 2(2), 2014.
19. Al-Kaabi,Engr S.S, & Belhaouari, Methods Toward Enhancing RSA Algorithm: A Survey, *Int, J, of Network Security and its Applications*, 11 (3) May 2019. [CrossRef]
20. Thiziers,A.H., Theodore, H.C., Zoueu, J.T. & Michel, B., Enhanced, Modified and Secured RSA Cryptosystem based on *n* Prime Numbers and Offline Storage for Medical Data Transmission via Mobile Phone, *Int. J. of Advanced Computer Sci. and Applications*, Vol.10., No. 19, 2019[CrossRef]
21. Alzhrani,K.and Aljaedi,A., Windows & Linux Random Number Generation Process: A Comparative Analysis, *Int. J. of Computer Applications*, 113(8), 2015. [CrossRef]
22. Sepahvandi,S., Hosseinzadeh,M., K.Navi & A.Jalili, An improved exponentiation algorithm for RSA Cryptosystem, *IEEE Int, Conf, on Research Challenges in Computer Sci*., 2009. [CrossRef]

AUTHORS PROFILE

Dr. Kannan Balasubramanian, is currently working as Professor in the Department of Computer Science and Engineering, School of Computing, Thanjavur. He received his M.Sc.(Tech) degree in Computer Science from BITS Pilani in 1989 and M.Tech degree in Computer Science and Engineering from IIT Bombay in 1991 and Ph.D degree in Computer Science from UCLA in 1999. He has 15 years of teaching experience at Mepco Schlenk Engineering College, Sivakasi. He worked on Virtual Private Networks after finishing his doctorate. He has published two books with IGI-Global and has published in many International Journals and Conferences. His areas of Interest are Computer Networks,Cryptography and Network Security, Cyber Security and Machine Learning for Cryptography.

M. Arun, is currently working as Associate Professor in the Department of Computer Science and Engineering at Veltech Rangarajan Dr. Sagunthala R& D Institute of Science and Technology (Deemed to be University), Chennai. He has 10 years of teaching experience at Mepco Schlenk Engineering College, Sivakasi and three years of teaching experience at Amrita College of Engineering and Technology, Nagercoil. He has published in many international journals. His areas of interests are Networking, Network Security and Machine Learning.

Dr. K. R. Sekar, is working as a Senior Assistant professor (SAP) in the department of computer science and engineering, School of Computing, SASTRA Deemed University, India for the past 15 years. His area of research is towards hesitant fuzzy, Big Data Analytics, Machine learning and Data Science.