Journal of Discrete Mathematical Sciences & Cryptography
ISSN 0972-0529 (Print), ISSN 2169-0065 (Online)
Vol. 26 (2023), No. 4, pp. 1005–1026
DOI : 10.1080/09720529.2021.1962026

The alternative method to speed up RSA's decryption process

Kritsanapong Somsuk
Department of Computer and Communication Engineering
Faculty of Technology
Udon Thani Rajabhat University
UDRU, Udon Thani
Thailand

Abstract

The aim of this paper is to propose the special methods to speed up RSA' s decryption process. In fact, this method gives the option of selecting the best parameters to recover the plaintext. Assuming that the new integer has a lower Hamming Weight than the private key and is mathematically related to the private key, it will be used as the exponent in the decryption process instead of the private key. Furthermore, this method can be used in conjunction with the Chinese Remainder Theorem (CRT) to determine the best exponents. In addition, a method derived from the Square and Multiplication Algorithm is proposed to compute modular exponentiation in order to reduce computation time. However, this method can be performed when number 1 of the exponent's binary value is not adjacent to each other. Assuming that the new integers for CRT have the lowest Hamming Weight, the experimental results show that if the proposed integers meet the condition, the proposed method is the best algorithm for completing the task. Furthermore, when 2048 bits of modulus is chosen and compared to CRT, the average time is reduced about 8%. Moreover, three-fourths of the cases require the proposed method with CRT in order to save the most computation costs on the decryption side. As a result, the proposed method is very likely to be chosen to complete the decrypting task.

Subject Classification: *94A60 (Cryptography), 00A69 (General applied mathematics), 68M25 (Computer security).*

Keywords: *RSA, Decryption process, Modular multiplication, Modular exponentiation, Modular square.*

1. Introduction

RSA [1] is one of the most effective methods for encrypting and decrypting secret information sent over an insecure channel. To complete

E-mail: kritsanapong@udru.ac.th

the task, it employs a pair of keys that are mathematically related to one another. The public key is the one that has been revealed to the group. The other key, which is kept secretly by the owner, is known as the private key. This system can be used for two different tasks. The first task is to use RSA for securing data sent via the network. The public key is used as the exponent in the encryption equation to transform the original message, known as the plaintext, into an unreadable message, known as the ciphertext, before sending it via the communication channel to the receiver. After receiving the ciphertext, the receiver can recover the plaintext by using a decryption process with the private key. At the moment, it is very popular to use RSA to secure a wide range of sensitive data, including images [2], [3], [4], and voice [5]. The second task is to perform digital signatures. It differs from the first task in which the private key is used for signing the message, while the public key is used for verifying the signed text. Furthermore, using RSA with a digital signature is common, as shown in [6], [7]. The private key must be assigned as a small integer to speed up RSA's decryption process. However, Michael J. Wiener demonstrated in 1990 that it is easily recovered when it is too small. That is, the private key must be sufficiently large in order to prevent intruder attacks [21]. The big value, on the other hand, has an immediate impact on the speed of the decryption process. Many methods for speeding up this process have been proposed [20]. Chinese Remainder Theorem (CRT) is a method for dividing the private key into multiple small integers that are used as exponents in the modified equation. The computation time is reduced because all values are less than the private key. Furthermore, CRT can be used with the Square and Multiplication Algorithms to speed up the computation. The Square and Multiplication Algorithm's main process is to compute both modular squares and modular multiplications. In general, the number of times to compute modular multiplication is determined by the Hamming Weight of the binary value of the private key. That is, the condition must be checked by if-statement before deciding to compute modular multiplication. Furthermore, when the Hamming Weight of sub value is high, the process may still be time-consuming.

In this paper, the special methods to speed up RSA's decryption process are presented. The first method is to find a new integer that is expanded from the private key and has a lower Hamming Weight than the private key. In fact, not all private keys of Hamming Weight are larger than their expansions. With only a few private keys, the first method is efficient. The second method includes reducing calculation time by improving the Square and Multiplication Algorithm by eliminating if-statements.

However, for the second proposed method, the exponent must meet the following criteria: number 1 of the exponent's binary value must not be adjacent to each other. Furthermore, in the combination with CRT, either of these methods can be utilized.

2. Related Works

This section will provide an overview of RSA as well as techniques for speeding up the decryption process.

2.1 *RSA Scheme*

RSA was first presented in 1977. In fact, three major processes are required to put this system in place.

1) **Key Generation Process:** It is the process of generating RSA's parameters, including a pair of keys, which are the primary parameters for the encryption and decryption processes. There are four steps as follows:

   Step 1 : Generate two large prime numbers randomly, p and q

   Step 2 : Compute modulus, $n = p*q$, and Euler totient function, $\Phi(n) = (p-1)*(q-1)$

   Step 3 : Select the public key, e, where $1 < e < \Phi(n)$ and $\gcd(e, \Phi(n)) = 1$

   Step 4 : Compute the private key, d, from $d = e^{-1} \bmod \Phi(n)$ by using [8], [9], [10]

2) **Encryption Process:** It is the process to convert the plaintext to the ciphertext before sending it via an insecure channel. The encryption equation is as follows:

$$c = m^e \bmod n \tag{1}$$

   Where m is the original plaintext, $1 < m < n$ and c is the ciphertext

3) **Decryption Process:** It is the process to recover the original plaintext from the ciphertext. The equation to decrypt the ciphertext is as follows:

$$m = c^d \bmod n \tag{2}$$

In general, e is often assigned as a small integer to decrease the computation time in client side, because this process must be implemented by multiple users, and one of them may use a low-power device to complete

the encryption process. Converting the plaintext to the ciphertext takes a lengthy time if the user encrypts it by using a low-power device and e is high. Furthermore, if d is chosen as a small integer, it can be quickly recovered by employing some of the attacking algorithms. The following examples are the efficient algorithm when d is small. The first technique is Wiener's attack [11] which was discovered by Michael J. Wiener in 1990. He showed that d is rapidly recovered when it is less than $\frac{1}{3}n^{\frac{1}{4}}$. Moreover, in 1999, Boneh and G. Durfee [12] improved on this method to broaden the scope of d, which is still easily broken. In fact, this scope is expanded from $d < \frac{1}{3}n^{\frac{1}{4}}$ to $d < n^{0.292}$. In [13], the method to assign the new initial value of d was proposed. In fact, when this value is discovered, the time required to find the target is decreased. Therefore, if d is too small, it may be easily recovered by using this technique. As a result, d should be assigned as a large integer to prevent intruder attacks. On the other side, it has a direct impact on the decryption process, which requires extremely high computation costs.

2.2 *Hamming Weight*

Hamming Weight is the number of letters that are not zero. Assigning H(z) is the Hamming Weight of z, where $z \in \mathbb{Z}^+$, the examples of Hamming Weight are as follows: H(13450) = 4, H(37) = 2 and H$((1001)_2)$ = 2. In fact, Hamming Weight of the exponent's binary value equal to number of modular multiplications when Square and Multiplication Algorithm is chosen to implement decryption process.

2.3 *Methods to Speed Up RSA's Decryption Process*

Because of the large private key, many methods have been proposed for speeding up the decryption process.

Method 1, Chinese Remainder Theorem: Its abbreviation is CRT**.** It can speed up modular exponentiation by dividing the exponent as multi small exponents. In fact, all of the sub exponents are less than their root. Furthermore, the process of recovering the original plaintext by combining CRT with the RSA decryption process, CRT+RSA, [14], [15] is as follows:

1) **Key Preparation:** Before recovering the original plaintext, it is necessary to prepare all parameters.

Step 1: Compute $d_p = d \bmod p - 1$ and $d_q = d \bmod q - 1$

Step 2: Find $y_p = p^{-1} \bmod q$ and $y_q = q^{-1} \bmod p$

In fact, d_p, d_q, y_p and y_q are only calculated once and all of them are selected to recover all ciphertexts which are encrypted by using RSA's encryption with the exponent, e.

2) **Decryption Process:** The original plaintext can be recovered by applying CRT with RSA' s decryption process as follows:

Step 1: Compute $m_p = c^{d_p} \bmod p$ and $m_q = c^{d_q} \bmod q$

Step 2: Recover m by using the equation (3)

$$m = (m_p * y_q * q + m_q * y_p * p) \bmod n \quad (3)$$

Method 2, the method in [16]: In 2016, RSA's decryption process was altered by replacing the private key with the new integer, d_t, where $d_t * e = x \bmod \Phi(n)$ and x should be a small integer. In general, Hamming Weight of d_t must be lower than d to increase the computation speed**.** In addition, the improved equation to recover m is $m = \sqrt[x]{c^{d_t} \bmod n}$. However, as compared to traditional decryption method, the range of m is limited. In fact, the range of m is from 1 to a, where $a^x < n < a^{x+1}$. As a result, the disadvantage is that the plaintext's scope is too limited.

Method 3, the method in [17]: In 2017, the new exponent for RSA's decryption process was proposed. Assuming that $s = \Phi(n) - d$, the original plaintext can be also computed by using the equation, $m = (c^{-1})^s \bmod n$, where c^{-1} is the inverse of c modulo n. This method is very efficient when d is a large integer especially d is very close to $\Phi(n)$. When d is small, however, it becomes the inefficient method.

Method 4, Square and Multiplication Algorithm: It is a fast method [18] for computing modular exponentiation using only modular square and modular multiplication processes. Before proceeding with the implementation, the exponent must be converted to a binary system. The number of loops used in the implementation is determined by the length of the binary value. In addition, number of times to compute modular multiplication depends on Hamming Weight of the binary value. The algorithm is as follows:

Algorithm 1: Square and Multiplication Algorithm

Input: array of b = binary value of d, n and c

Output: $m = c^d \bmod n$

1. $i \leftarrow$ (length of b) - 2
2. $z \leftarrow c$
3. While $i \geq 0$ Do
4. $z \leftarrow z^2 \bmod n$
5. IF $b[i] = 1$ Then
6. $z \leftarrow z*c \bmod n$
7. End IF
8. $i \leftarrow i - 1$
9. End While
10. $m \leftarrow z$

Method 5, Akira's Method [19]: In 2000, A. Hayashi proposed the method to speed up modular multiplication. That is, it can be used in the combination with the Square and Multiplication Algorithm when step 6 in Algorithm 1 is required. To use Akira's Method, all prime factors of $n + 2$ must be found at first. However, for RSA, n is a large number and $n + 2$ is an odd number. Therefore, it is very hard to find all prime factors of them.

3. The Proposed Method

The aim of this paper is to propose methods for speeding up the RSA decryption process. The first method is to expand the size of the private key in order to find the new integer that is selected as the exponent for decrypting equation rather than the private key. In addition, Hamming Weight must be lower than the private key. In fact, if Hamming Weight is very low, it implies that the time required to compute modular exponentiation using the Square and Multiplication Algorithm is reduced. The second method is the improvement of Square and Multiplication Algorithm to decrease the computation time. The key is to remove if-statement from the algorithm while the process in this statement is still implemented when the condition is true. The limitation of the second method is that it can be implemented when number 1 of the exponent's binary value is not adjacent to each other. Furthermore, both of the first and second methods can be used in combination with CRT.

3.1 *Expanding the Private Key to Find the new Exponent with Low Hamming Weight*

In fact, the private key can be expanded by adding this value with the multiplication between the positive integer and Euler totient function. The result from decryption process is still equal to the original plaintext.

Theorem 1 : *Assigning* $d_r = d + a^*\Phi(n)$, *where* $a \in \mathbb{Z}^+$, *then m can be computed from:*

$$m = c^{d_r} \bmod n \tag{4}$$

Proof : From, $c^{d_r} \bmod n = (m^e)^{d_r} \bmod n = (m^e)^{d+a^*\Phi(n)} \bmod n$

$$= m^{ed} * m^{e^*a^*\Phi(n)} \bmod n = m^{1+k^*\Phi(n)} * m^{e^*a^*\Phi(n)} \bmod n$$

$$= m^* m^{k^*\Phi(n)} * m^{e^*a^*\Phi(n)} \bmod n$$

From, Euler's theorem, $a^{\Phi(n)} \bmod n = 1$, therefore, it implies that

$m^{k^*\Phi(n)} \bmod n = (m^{\Phi(n)})^k \bmod n = 1^k = 1$, also, $m^{e^*a^*\Phi(n)} \bmod n = 1$

Then, $c^{d_r} \bmod n = m^*1^*1 \bmod n = m$

Therefore, m can be recovered by using equation (4) □

On the other hand, if d_r is assigned before computing d, d can be calculated from $d = d_r \bmod \Phi(n)$. However, d_r becomes an important value when Hamming Weight of its binary value is very low. That is, to make it easier to find this value, it should be chosen before computing the private key. Therefore, the prime generation process must be modified as follows:

Algorithm 2: Improved Key Generation Process

Input: -

Output: all parameters of RSA including d_r

1. Choose two large prime numbers randomly, p and q
2. $n \leftarrow p^*q$
3. $\Phi(n) \leftarrow (p-1)^*(q-1)$
4. Choose d_r, where $d_r > \Phi(n)$ and $H(d_r)$ is very small
5. $d \leftarrow d_r \bmod \Phi(n)$
6. IF ($H(d_r) > H(d)$) or ($\gcd(d, \Phi(n)) \neq 1$) then

7. Back to Step 1 or using d as the exponent for decrypting equation
8. End IF
9. $e \leftarrow d^{-1} \bmod \Phi(n)$

Assuming that the cost of computing modular multiplication is the same as the cost of computing modular square, the example 1 shows that if $H(d_r)$ is very low. Then, d_r is a better choice for the exponent for decryption.

Example 1 :
Process 1 : Improved Prime Generation
Step 1-3 : $p = 937$, $q = 857$, $n = 803009$ and $\Phi(n) = 801216$

Step 4 : Choose, $d_r = 1048577 = 10000000000000000001_2$

Step 5 : $d = 1048577 \bmod 801216 = 247361 = 111100011001000001_2$

Step 6 – 8 : Because $H(d_r) < H(d)$ and $\gcd(d, \Phi(n)) = 1$, then continuing to Step 9

Step 9: $e = 247361^{-1} \bmod 801216 = 449921$

Process 2 : Encryption Process, Assuming $m = 123255$ and 307411, then $c_1 = 123255^{449921} \bmod 803009 = 222601$, $c_2 = 307411^{449921} \bmod 803009 = 76410$

Process 3: Decryption Process
Method 1: Using d as the exponent of the equation to recover m, then $m_1 = 222601^{247361} \bmod 803009 = 123255$, $m_2 = 76410^{247361} \bmod 803009 = 307411$

Method 2 (The Proposed method): Using d_r as the exponent for the equation to recover m, then $m_1 = 222601^{1048577} \bmod 803009 = 123255$, $m_2 = 76410^{1048577} \bmod 803009 = 307411$

Analyzing each decryption algorithm in Example 1

1) Using d as the exponent: Because bit length of d is 18, then there are 17 modular squares. In addition, $H(d) = 8$, then 7 modular multiplications are required. Therefore, total costs are 24 modular multiplications.
2) Using d_r (The proposed Method) as the exponent: Because bit length of d_r is 21, then there are 20 modular squares. In addition, $H(d_r) = 2$,

Table 1

Total costs of two algorithms to recover plaintext in Example 1

Method	Modular Square/Modular Multiplication Computing		Total Costs
	Equation	Costs	
Traditional RSA's decryption	$m = c^d \bmod n$	24	24
The Proposed Method	$m = c^{d_r} \bmod n$	21	21

then only one modular multiplication is required. Therefore, total costs are 21 modular multiplications.

Table 1 shows the conclusion regarding to the total costs to find m of two techniques.

3.2 *The improvement of Square and Multiplication Algorithm*

Although, d_r is a large number, H(d_r) is very small. That is, when compared to d, it implies that the computation time is reduced, where H(d) > H(d_r). However, if Square and Multiplication algorithm is chosen to recover m, the condition (if statement) in this algorithm must be selected to check the value of each bit to compute the modular multiplication when the value is 1. The goal of this method is to eliminate the process of checking the condition from the Square and Multiplication Algorithm. Assume that after d_r has been converted to a binary value, the number 1 must not be adjacent to each other. This algorithm can be modified as follows:

Step 1 : Splitting a binary value of d_r by using '1' as a bisector. Assuming the binary value is as follows:

1	0		1		1

In fact, The cells with gray backgrounds are the bisector. As a result of the splitting process, there are three parts as follows:

Part 1: null, Part 2: "00", Part 3: "0"

Because, the most significant bit is always 1, then the member in Part 1 is null.

Step 2: Computing modular square for each part. In fact, the process must be begun at part 2, part 3, part 4, ⋯ respectively. Furthermore, the number of modular squares for each part is determined by the number of members in the group. For example, in part 2, modular square is computed twice. On the other hand, one modular square is required in part 3. After completing the modular square process for each part, two additional processes must be calculated. The first is modular square, while the second is modular multiplication. In fact, both of them are the representative of the digit '1' which is removed from each part.

Algorithm 3: Improved Square and Multiplication Algorithm

Input: binary value of d, n and c

Output: $m = c^d \bmod n$

Condition: Number 1 in binary value is not adjacent to each other

1. Array of b ← Splitting binary value of d by using '1' as a bisector
2. $i \leftarrow 1$
3. $z \leftarrow c$
4. While $i <$ length of b Do
5. $\quad j \leftarrow 0$
6. $\quad$ While $j <$ length of $b[i]$ Do
7. $\quad\quad z \leftarrow z^2 \bmod n$
8. $\quad\quad j \leftarrow j + 1$
9. $\quad$ End While
10. $\quad z \leftarrow z^2 \bmod n$
11. $\quad z \leftarrow z*c \bmod n$
12. $\quad i \leftarrow i + 1$
13. End While
14. $m \leftarrow z$

Example 2 : Compute $13^{37} \bmod 23$

Sol : Because $37 = 100101_2$, then Algorithm 3 can be selected to find the result as follows:

Step 1 : $b[0]$ = null, $b[1]$ = "00", $b[2]$ = "0"

Step 2-3 : $i = 1$, $z = 13$

Step 4 – 13 : Because $i = 1 < 3$, then (**Process in while loop 1**)

Step 5 : $j = 0$

Step 6 – 9 : Because $j = 0 < 2$ (length of $b[1] = 2$), then (**Process in while loop 2**)

Step 7 – 8 : $z = 13^2 \bmod 23 = 8$ (**Modular Square**), $j = 1$
Because $j = 1 < 2$ (length of $b[1] = 2$), then (**Process in while loop 2**)

Step 7 - 8 : $z = 8^2 \bmod 23 = 18$ (**Modular Square**) , $j = 2$

Step 10 - 11 : $z = 18^2 \bmod 23 = 2$ (**Modular Square**), $z = 2*13 \bmod 23 = 3$ (**Modular Multiplication**)

Step 12 : $i = 2$
Because $i = 2 < 3$, then (**Process in while loop 1**)

Step 5 : $j = 0$

Step 6 – 9 : Because $j = 0 < 1$ (length of $b[2] = 1$), then (**Process in while loop 2**)

Step 7 - 8 : $z = 3^2 \bmod 23 = 9$ (**Modular Square**), $j = 1$

Step 10-11 : $z = 9^2 \bmod 23 = 12$ (**Modular Square**), $z = 12*13 \bmod 23 = 18$ (**Modular Multiplication**)

Step 12 : $i = 3$
Because $i = 3$, then Jumping to Step 14

S**tep 14 :** $m = z = 18$
Therefore, $13^{37} \bmod 23 = 18$

Assuming that the cost to find modular square equals to modular multiplication, total processes in while loops are 2 (main loop) + 3 (inner loop) = 5 and there are 7 modular multiplications.

If Square and Multiplication Algorithm is chosen, total processes in while loop equal to 5 and the process requires 7 modular multiplications that both of them equal to the proposed method using Improved Square and Multiplication Algorithm. However, an if-statement is required to check the condition of the Square and Multiplication Algorithm. Furthermore, number of times to check the condition equals to total

processes in while loop. In general, if H(d_r) is low, it is high possible that number 1 in the binary value of d_r is not adjacent to each other. As a result, if this situation arises, the proposed method using d_r as the exponent can be combined with the Improved Square and Multiplication Algorithm to reduce computation time. Moreover, the second method can be also applied with d when number 1 in its binary value is not adjacent to each other. Before using Algorithm 3 to calculate modular exponentiation, the exponent should be checked to see if it can be used as the exponent for this algorithm. Algorithm 4 is proposed to check the exponent before choosing either Square and Multiplication Algorithm or Improved Square and Multiplication Algorithm. The idea behind this algorithm is the dividing a binary value into many parts by using '0' as a bisector. After dividing a binary value, if at least one part has lengths greater than 1, this value cannot be used as the exponent in Algorithm 3.

Algorithm 4: Checking the binary value that number 1 is adjacent to each other or not.

Input: binary value of d_r

Output: Yes/No

1. Array of b ← Splitting d_r by using '0' as a bisector
2. i ← 0
3. r ←0
4. While i < length of b Do
5. IF length of $b[i]$ > 1 Then
6. r ← 1
7. Break
8. End IF
9. i ← i + 1
10. End While
11. IF r equals to 1 Then
12. Answer is "Yes"
13. Else
14. Answer is "No"
15. End IF

Example 3: Checking that 43 can be selected as the exponent for Improved Square and Multiplication Algorithm or not, by using Algorithm 4.

Sol. Because, 43 = 101011, then

Step 1-3 : $b[0]$ = "1", $b[1]$ = "1", $b[2]$ = "11" (length of b = 3), i = 1, r = 0

Step 4 – 10 : Because $i = 0 < 3$ and length of $b[0] = 1$, then (**Process in while loop**)

Step 9 : i = 1
Because $i = 1 < 3$ and length of $b[1] = 1$, then (**Process in while loop**)

Step 9 : i = 2
Because $i = 2 < 3$ and length of b[2] = 2 , then (**Process in while loop**)

Step 6 - 7 : r = 1 and Jumping out of the loop

Step 11 - 15 : Because r = 1, then

Step 12 : Answer is "Yes"

Because, there is a part that number 1 is adjacent to each other, then this value cannot be selected as the exponent for Improved Square and Multiplication Algorithm.

Although implementing Algorithm 4 before deciding on the best way to implement modular exponentiation is time consuming, it is only done once. Assuming d_r is passed from algorithm 4, every time that decryption process is required, Algorithm 3 can be selected for the implementation without requiring to use Algorithm 4 again.

3.3 *The Proposed Method applied with CRT*

In the real situation, CRT+RSA is often chosen to combine with some modular exponentiation algorithms to speed up decryption process. Therefore, in this part, the proposed method is used in combination with CRT to determine the best exponents for speeding up the decryption process.

Theorem 2 : *Assigning* $d_{pr} = d_p + a^*(p - 1)$, *where* $a \in \mathbb{Z}^+$, *then* m_p *can be computed from:*

$$m_p = c^{d_{pr}} \bmod p \tag{5}$$

Proof : From, $c^{d_{pr}} \bmod p = c^{d_p + a^*(p-1)} \bmod p = c^{d_p} * c^{a^*(p-1)} \bmod p = c^{d_p} * (c^a)^{p-1} \bmod p$

From, Fermat's little Theorem, $a^{p-1} \bmod p = 1$, therefore, it implies that:

$$(c^a)^{p-1} \bmod p = 1, \text{then, } c^{d_{pr}} \bmod p = c^{d_p} \bmod p = m_p$$

Therefore, m_p can be recovered by using equation (5) □

In addition, assuming $d_{qr} = d_q + b^*(q-1)$, where $b \in \mathbb{Z}^+$, then m_q can be recovered by using (6).

$$m_q = c^{d_{qr}} \bmod q \tag{6}$$

Therefore, if $H(d_p) > H(d_{pr})$, then the equation (5) is more efficient than the traditional equation to recover m_p. The same reason, if $H(d_q) > H(d_{qr})$, then the equation (6) is more efficient than the traditional equation.

Example 4 : Assigning, $p = 230807$, $q = 186019$, $n = 42934487333$ and $\Phi(n) = 42934070508$

Assuming, choosing $e = 34132305811$, then $d = 5362611223$

Therefore,

$d_p = 5362611223 \bmod 230806 = 64{,}619 = 1111110001101011_2$, $H(d_p) = 11$

$d_q = 5362611223 \bmod 186018 = 84{,}319 = 10100100101011111_2$, $H(d_q) = 10$

$d_{pr} = 64619 + 1^*(230807 - 1) = 295425 = 1001000001000000001_2$, $H(d_{pr}) = 4$

$d_{qr} = 84319 + 1^*(186019 - 1) = 270337 = 1000010000000000001_2$, $H(d_{qr}) = 3$

Assuming, $c = 1311$, finding m_p and m_q by using CRT+RSA and the proposed method with CRT

Method 1: Choosing CRT+RSA to find m_p and m_q, $m_p = 1311^{64619} \bmod 230807 = 94666$, $m_q = 1311^{84319} \bmod 186019 = 26401$

Method 2: Choosing the proposed method with CRT $m_p = 1311^{295425} \bmod 230807 = 94666$, $m_q = 1311^{270337} \bmod 186019 = 26401$

Analyzing each decryption algorithm in Example 4

1) **Using d_p and d_q as the exponents to find m_p and m_q respectively:**

   Considering the costs to find m_p

   Because bit length of d_p is 16, then there are 15 modular squares. In addition, H(d_p) = 11, then 10 modular multiplications are required, therefore, total costs to find m_p are 25.

   Considering the costs to find m_q

   Because bit length of d_q is 17, then there are 16 modular squares. In addition, H(d_q) = 10, then 9 modular multiplications are required, therefore, total costs to find m_q are 25.

   Considering the costs to find m

   Equation (3) which requires 2 modular multiplications (m_p*x mod n and m_q*y mod n, where $x = y_q$*q mod n and $y = y_p$*p mod n) is also chosen to find m. Therefore, total costs are increased as 25 + 25 + 2 = 52.

2) **Using d_{pr} and d_{qr} as the exponents to find m_p and m_q respectively:**

   Considering the costs to find m_p

   Because bit length of d_{pr} is 19, then there are 18 modular squares. In addition, H(d_{pr}) = 4, then 3 modular multiplications are required, therefore, total costs to find m_p are 21.

   Considering the costs to find m_q

   Because bit length of d_{qr} is 19, then there are 18 modular squares. In addition, H(d_{qr}) = 3, then 2 modular multiplications are required, therefore, total costs to find m_q are 20.

   Considering the costs to find m

   Equation (3) which requires 2 modular multiplications (m_p*x mod n and m_q*y mod n, where $x = y_q$*q mod n and $y = y_p$*p mod n) is also chosen to find m. Therefore, total costs are increased as 21 + 20 + 2 = 43.

The information in Table 2 is shown that the costs for the proposed method with CRT are less than CRT+RSA. In addition, if Number 1 in a binary value of the exponent is not adjacent to each other, Improved Square and Multiplication Algorithm can be selected to speed up the process.

Table 2
The comparison about costs between CRT+RSA and the proposed method in Example 4

Method	Modular Square/Modular Multiplication Computing		Total Costs
	Equation	Costs	
CRT+RSA	$m_p = c^{d_p} \bmod p$	25	52
	$m_q = c^{d_q} \bmod q$	25	
	$m = (m_p y_q q + m_q y_p p) \bmod n$	2	
The Proposed Method with CRT	$m_p = c^{d_{pr}} \bmod p$	21	**43**
	$m_q = c^{d_{qr}} \bmod q$	20	
	$m = (m_p y_q q + m_q y_p p) \bmod n$	2	

However, it is possible that ($H(d_{pr}) < H(d_p)$ but $H(d_{qr}) > H(d_q)$) or ($H(d_{qr}) < H(d_q)$ but $H(d_{pr}) > H(d_p)$) or ($H(d_{pr}) > H(d_p)$ and $H(d_{qr}) > H(d_q)$). Algorithm 5 is introduced to choose the best exponents for the computation.

Algorithm 5: Choosing the best exponents for computing m_p and m_q

Input: d_p, d_q, d_{pr}, d_{qr} (length of d_p is close to d_{pr} and length of d_q is close to d_{qr})

Output: Exponents for computing m_p and m_q

1. IF $H(d_p) > H(d_{pr})$ Then
2. d_{pr} is chosen as the exponent to find m_p
3. Else
4. d_p is chosen as the exponent to find m_p
5. End IF
6. IF $H(d_q) > H(d_{qr})$ Then
7. d_{qr} is chosen as the exponent to find m_q
8. Else
9. d_q is chosen as the exponent to find m_q
10. End IF

4. Experimental Results

In this section, the experimental results are presented. Bit length of modulus in the experment is 512, 1024 and 2048. In fact, the results are distinguished as two parts. The aim of both experiments is to demonstrate the special case in which the proposed method completes the task faster than the compared method. Because the proposed method is appropriate for some cases, only option can be selected when the special case occurs.

The first experiment compares the time required to complete a task using the traditional decryption process and the proposed method. In fact, When this special case occurs, all information in this experiment is from $H(d) > H(d_r)$, which can be implied that the proposed method is more appropriate than the compared method.

The second experiment is the comparison about time required to complete the task between using CRT+RSA by selecting d_p and d_q as the exponents and the proposed method with CRT by selecting d_{pr} and d_{qr} as the exponents. In fact, all information in this experiment is from $H(d_p) > H(d_{pr})$, $H(d_q) > H(d_{qr})$ and the length of d_p and d_q are close to d_{pr} and d_{qr} repectively, implying that the proposed method is more suitable than the compared method when this special case occurs.

Moreover, assuming number 1 of binary value of d_r, d_{pr} and d_{qr} is not adjacent to each other to choose the Improved Square and Multiplication Algorithm to decrease time to finish the task.

In addition, because of the unlimited size, BigInteger class in Java is chosen as the variables of each parameter. In addition, all experiments

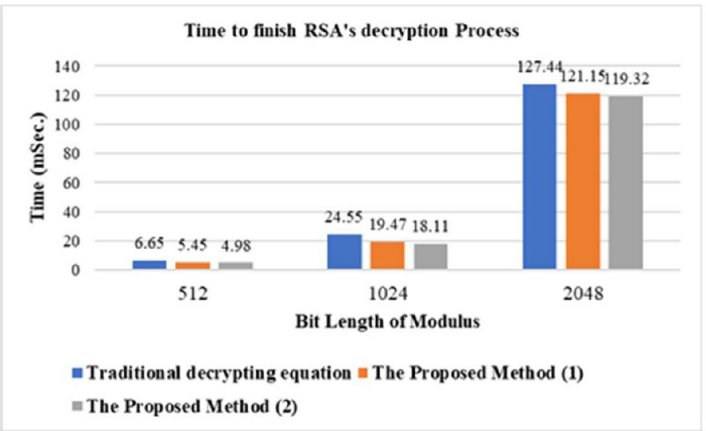

Figure 1

Time to finish decryption process

were conducted on 2.53 GHz an Intel® Core i5 with 8 GB memory in order to control the same settings.

For the first experiment, assigning that the traditional decryption equation is $m = c^d \bmod n$, the proposed method (1) is: $m = c^{d_r} \bmod n$ and the proposed method (2) is: $m = c^{d_r} \bmod n$ by using Algorithm 3

Figure 1 depicts a comparison of the average time required to complete the decryption process for three algorithms. The experimental results show that the proposed method (2) is the fastest algorithm when $H(d) > H(d_r)$ and number 1 of the binary value of d_r is not adjacent to each other.

In fact, it is possible that there is not d_r in which the Hamming Weight is smaller than $H(d)$. Therefore, if this situation is occured, then the proposed method should not be selected to recover the plaintext.

For the second experiment, assigning that $m = (m_p * y_q * q + m_q * y_p * p) \bmod n$, CRT+RSA is: $m_p = c^{d_p} \bmod p$, $m_q = c^{d_q} \bmod q$, the proposed method (3) is: $m_p = c^{d_{pr}} \bmod p$ and $m_q = c^{d_{qr}} \bmod q$, the proposed method (4) is: $m_p = c^{d_{pr}} \bmod p$, $m_q = c^{d_{qr}} \bmod q$ and recover m by using Algorithm 3

In addition, the second experiment is divided into two parts:

Figure 2 depicts a comparison of the average time to complete the decryption process among three algorithms when d_p and d_q are balanced. The information in Figure 3 differs from the information in Figure 2 in that d_p and d_q are imbalance. The experimental results in both tables show that

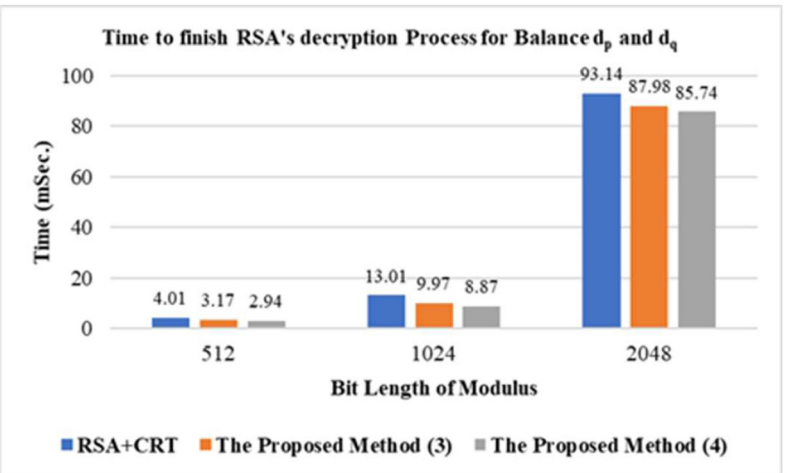

Figure 2

Time to finish decryption process for Balance d_p and d_q

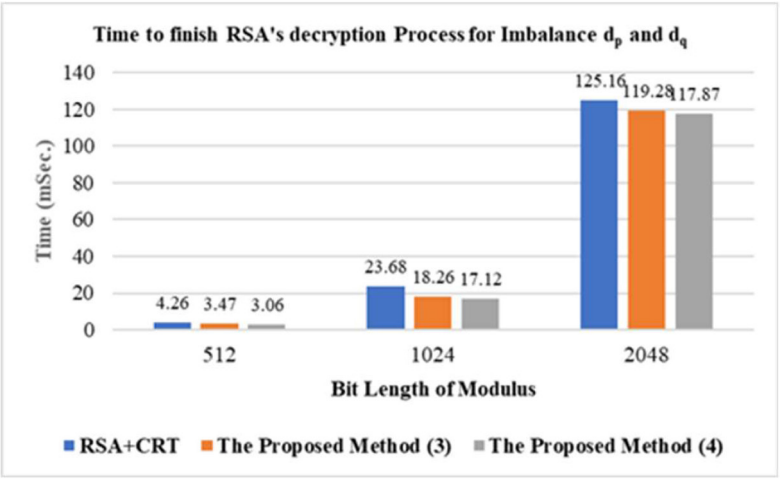

Figure 3

Time to finish decryption process for Imbalance d_p and d_q

the proposed method (4) is the fastest algorithm when $H(d_p) > H(d_{pr})$, $H(d_q) > H(d_{qr})$ and number 1 of a binary value of d_{pr} and d_{qr} is not adjacent to each other. However, if there are some parts that number 1 of a binary value of the proposed exponent is adjacent to each other, then the proposed method (4) can not be performed. Therefore, the proposed method (3) becomes the fastest algorithm in this case.

Furthermore, when the proposed method is used to complete the task with a modulus length of 2048 bits, the average time is reduced about 8%.

5. Discussion

From all results in this paper, it implies that to save the costs in decryption side, CRT which d_p and d_q are balance to each other must be selected and to be used with some modular exponentiation algorithms. However, after the proposed method are presented, d_{pr} and d_{qr} which the length are close to d_p and d_q in order are disclosed, there are four cases to select the best exponents to save the most computation costs as follows:

Case 1 : $H(d_p) < H(d_{pr})$ and $H(d_q) < H(d_{qr})$

Using $m_p = c^{d_p} \bmod p$ and $m_q = c^{d_q} \bmod q$ (The proposed method with CRT is not requires)

Case 2 : $H(d_p) > H(d_{pr})$ and $H(d_q) < H(d_{qr})$

Using $m_p = c^{d_{pr}} \bmod p$ and $m_q = c^{d_q} \bmod q$ (**The proposed method with CRT is required to find m_p**)

Case 3 : $H(d_p) < H(d_{pr})$ and $H(d_q) > H(d_{qr})$

Using $m_p = c^{d_p} \bmod p$ and $m_q = c^{d_{qr}} \bmod q$ (**The proposed method with CRT is required to find m_q**)

Case 4 : $H(d_p) > H(d_{pr})$ and $H(d_q) > H(d_{qr})$

Using $m_p = c^{d_{pr}} \bmod p$ and $m_q = c^{d_{qr}} \bmod q$ (**The proposed method with CRT is required to find m_p and m_q**)

In fact, three-fourths cases require the proposed method with CRT to save the most computation costs in decryption side. As a result, there is a good chance that the proposed method will be chosen as the most important part to complete the task.

5. Conclusion

In this paper, the special method to speed up decryption process is proposed. Furthermore, this method can be used in combination with CRT in order to reduce the computation time. Moreover, Improved Square and Multiplication Algorithm which is modified from Square and Multiplication Algorithm is also proposed to compute modular exponentiation. This algorithm has the advantage in which it can remove the condition in the loop. However, the condition to select Improved Square and Multiplication Algorithm is that number 1 of the binary value of the exponent is not adjacent to each other. That is, this algorithm is appropriate for the exponent with low Hamming Weight. The experimental results show that if the exponent of the proposed method has the lowest Hamming Weight , the proposed method is the best option to recover the plaintext. Furthermore, when the proposed method is used to complete the task with a 2048 bits of modulus, the average time is reduced by 8%. In addition, it is also shown that three-fourths cases require the proposed method with CRT in order to save the most computation costs in decryption side.

References

[1] R.L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems", *Communications of ACM,* vol. 21, pp. 120 – 126, 1978.

[2] V. Guleria, S. Sabir and D.C. Mishra, "Security of multiple RGB images by RSA cryptosystem combined with FrDCT and Arnold transform", *Journal of Information Security and Applications*, vol. 54, pp. 1 – 13, 2020.

[3] L. Yang, T. Shanyu, L. Ran, Z. Liping and M. Zhao, "Secure and robust digital image watermarking scheme using logistic and RSA encryption", *Expert Systems with Applications,* vol. 97, pp. 95 - 105, 2018.

[4] K. Jiao, G. Ye, Y. Dong, X. Huang and J. He, "Image Encryption Scheme Based on a Generalized Arnold Map and RSA Algorithm", *Security and Communication Networks,* vol. 2020, pp. 1 - 14, 2020.

[5] Y. Wang and Z. Tan, "Modified RSA Encryption Algorithm Based on Chaos and Its Application in Voice Encryption System", in Proc. of International Conference on Computational Intelligence and Security, pp. 439 - 444, November 16 – 19, 2018.

[6] H. Om, and M.R. Reddy, "RSA based remote password authentication using smart card", *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 15, pp. 105-111, 2012 .

[7] F.J. Aufa, Endroyono and A. Affandi, "Security System Analysis in Combination Method: RSA Encryption and Digital Signature Algorithm", in Proc. of International Conference on Science and Technology, pp. 1-5, November 15, 2018.

[8] K.G. Chol, L.S. Chol and H.H. Cho, "Fast rebalanced RSA signature scheme with typical prime generation", *Theoretical Computer Science*, vol. 830 - 831, pp.1 - 19, 2020.

[9] S.M. Sedjelmaci, "On a parallel extended Euclidean algorithm", in Proc. of ACS/IEEE International Conference on Computer Systems and Applications, pp. 235 - 241, June 25 – 29, 2001.

[10] D. Chandravathi and P.V. Lakshmi, "Privacy Preserving Using Extended Euclidean Algorithm Applied To RSA-Homomorphic Encryption Technique", *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, pp.3175 - 3179, 2019.

[11] M. Wiener, "Cryptanalysis of short RSA secret exponents", *IEEE Transactions on Information Theory*, vol. 36, pp. 553-558, 1990.

[12] D. Boneh, and G. Durfee, "Cryptanalysis of RSA with Private Key d less than N0.292". *Lecture Notes in Computer Science*, vol.1592, pp. 1 – 11, 1999.

[13] K. Somsuk, "A New Methodology to Find Private Key of RSA Based on Euler Totient Function", *Baghdad Science Journal*, vol. 18(2), pp.338–348, 2021.

[14] F. Kong, D. Zhou, Y. Jiang, J. Shang and J. Yu, "Fault Attack on an Improved CRT-RSA algorithm with the Modulus Chaining Method", in Proc. of IEEE International Conference on Computational Science and Engineering and IEEE International Conference on Embedded and Ubiquitous Computing, pp. 866 - 869, July 21 - 24, 2017.

[15] J. Park, E. Park, S. Moon, D. Choi, Y. Kang and J. Ha, "Fault resistant CRT-RSA scheme adopting a small exponent", in Proc. of International Conference on Computer Sciences and Convergence Information Technology, pp. 544 - 549, November 30 – December 2, 2010.

[16] K. Somsuk, "The improving decryption process of RSA by choosing new private key", in Proc. of International Computer Science and Engineering Conference, pp. 1-5, November 15 – 18, 2017.

[17] K. Somsuk, "The New Equation for RSA's Decryption Process Appropriate with High Private Key Exponent", in Proc. of International Conference on Information Technology and Electrical Engineering, pp. 1-4, October 5 - 6, 2016.

[18] A.Mazzero, L.Romano, G.P. Saggese and N. Mazzocca, "FPGA-based Implementation of a serial RSA processor", in Proc. of Design, Automation and Test in Europe Conference and Exhibition, pp. 582 - 587, December 19, 2003.

[19] A. Hayashi, "A New Fast Modular Multiplication Method and Its Application to Modular Exponentiation-Based Cryptography", *Electronics and Communications in Japan*, Part 3, vol. 83, pp. 1372 - 1376, 2000.

[20] A. Rawat, K. Sehgal, A. Tiwari, A. Sharma and A. Joshi, "A novel accelerated implementation of RSA using parallel processing", *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 22(2), pp. 309 - 322, 2019.

[21] M. Mumtaz and L. Ping, "Forty years of attacks on the RSA cryptosystem: A brief survey", *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 22(1), pp. 9-29, 2019.

Received February, 2021
Revised June, 2021