The new Weakness of RSA and The Algorithm to Solve this Problem

Kritsanapong Somsuk*
Department of Computer and Communication Engineering, Faculty of Technology,
Udon Thani Rajabhat University, UDRU,
Udon Thani, Thailand
kritsanapong@udru.ac.th
*Corresponding author: Kritsanapong Somsuk

Received February 24, 2020; revised June 19, 2020; accepted August 1, 2020; published September 30, 2020

Abstract

RSA is one of the best well-known public key cryptosystems. This methodology is widely used at present because there is not any algorithm which can break this system that has all strong parameters within polynomial time. However, it may be easily broken when at least one parameter is weak. In fact, many weak parameters are already found and are solved by some algorithms. Some examples of weak parameters consist of a small private key, a large private key, a small prime factor and a small result of the difference between two prime factors. In this paper, the new weakness of RSA is proposed. Assuming Euler's totient value, $\Phi(n)$, can be rewritten as $\Phi(n) = ad + b$, where d is the private key and $a, b \in \mathbb{Z}$, if a divides both of $\Phi(n)$ and b and the new exponent for the decryption equation is a small integer, this condition is assigned as the new weakness for breaking RSA. Firstly, the specific algorithm which is created for this weakness directly is proposed. Secondly, two equations are presented to find a, b and d. In fact, one of two equations must be implemented to find a and b at first. After that, the other equation is chosen to find d. The experimental results show that if this weakness has happened and the new exponent is small, original plaintext, m, will be recovered very fast. Furthermore, number of steps to recover d are very small when a is large. However, if a is too large, d may not be recovered because m which must be always written as $m = h^a$ is higher than modulus.

Keywords: RSA, Weak Parameter, Private Key, Euler's totient value

1. Introduction

Cryptography [1] is the significant technique to secure the secret information by using encryption and decryption processes. It falls into two groups. The first group is called symmetric key cryptography. It uses the same key which is called the secret key to encrypt original plaintext and decrypt the ciphertext. The advantage is about time to finish processes in both sides. Moreover, low computational cost is required. Advanced Encryption Standard (AES) [2], [3] is the best algorithm. However, the disadvantage of all algorithms in this group is the problem to find the secret channels to exchange the secret key between sender and receiver. Later, the other technique which is different from the first group was proposed. Asymmetric key cryptography or public key cryptography is the name of this technique. In fact, a pair of keys is selected for encryption and decryption processes, both keys are called public key and private key. In depth, a public key is always disclosed to interlocutors. On the other hand, a private key must be kept secretly by owner. The first algorithm which was born in 1976 was discovered by W. Diffie and M.E. Hellman [4]. Although, Diffie and Hellman' s method cannot be chosen to secure the information, it is selected as the secret way to exchange a secret key which is transmitted via the insecure channel. In 1978, RSA [5], [6] which is another public key cryptography was proposed. It is different from Diffie and Hellman's algorithm because RSA can secure many types of information such as text and image [7], [8] by using encryption and decryption processes. In general, at least 1024 bits of modulus must be selected to avoid an attack by intruders. Although RSA is one of the best algorithms in this group, it may be easily broken whenever at least one hidden parameter is weak. For examples, in 1990, M. Wiener [9], [10] presented the method to recover a private key by using continued fractions. This algorithm has very high performance when a private key is very small especially a key which is less than $\frac{1}{3}n^{0.25}$, where n is modulus. Later, D. Boneh and G. Durfee [11] improved Wiener's technique. In that time, they showed that the modified technique is still efficient to recover a private key, although it is bigger than $\frac{1}{3}n^{0.25}$. However, their method becomes incapable when a private key is larger than $n^{0.292}$. In addition, brute force attack is suitable for a small private key, because the initial value for the investigation is the smallest odd integer. Furthermore, factoring [12] the modulus as prime numbers is another methodology to find a private key, because Euler's totient value is disclosed. Most of algorithms can finish the process very fast when the characteristic of some prime factors is weak. In 2017, the technique [13] to speed up RSA's decryption process with a large private key was proposed by using the modified decryption equation. The inversion of ciphertext modulo n is chosen as the base instead of the ciphertext and the private key is replaced by the new integer. In fact, this method is suitable for only the large private key, because the size of the new exponent is opposite of the private key. On the other hand, this technique may be chosen to attack RSA by using brute force attack whenever a private key is too large. It affects to the size of the new exponent which is too small. Therefore, it is an easy way for intruders to break RSA by using the equation with the base as inversion of ciphertext modulo n and the random exponent.

The aim of this paper is to propose the new idea to break RSA. Furthermore, the weak parameter for this method occurs when a private key, d, can be rewritten as $\frac{\Phi(n)-b}{a}$, where $a, b \in \mathbb{Z}$, $\Phi(n)$ is Euler's totient value, a divides b ($a|b$) and a divides $\Phi(n)$ ($a|\Phi(n)$). In fact, if d is from the above condition, a and b are easily disclosed. In contrast, d will not be discovered by using the proposed method in the case that it cannot be written as this form.

The rest of the paper is organized as follows. In section 2, the related works are mentioned. It consists of the overview of RSA and the techniques to recover d. The proposed method will be discussed in Section 3. In section 4, the experimental results are presented. Finally, the last section is about the conclusion of this research.

2. Related Work

In this section, the conceptual idea of RSA and the ways to recover d by intruders are mentioned. In fact, all algorithms which are in the special proposed group may break RSA in polynomial time when at least one parameter is weak.

2.1 RSA

RSA [5], [6] is one of public key cryptosystems to protect the secret information by using data encryption. Because of high security, when n is assigned at least 1024 bits, RSA is still widely used at present. In general, there are three processes to secure the plaintext by using RSA. First step is the key generation process. The beginning of this process is the generating two large prime numbers, p and q ($p < q$), randomly. The second step is to compute $n = p*q$ and $\Phi(n) = (p-1)*(q-1)$. The next step is to choose a public key, e, from the following condition: $1 < e < \Phi(n)$ and the greatest common divisor between e and $\Phi(n)$ must be equal to 1, $\gcd(e, \Phi(n)) = 1$. Computing d, from $e*d \bmod \Phi(n) = 1$, is performed in the last step. In fact, Extended Euclidean Algorithm or the improved methods [14], [15] are the method to calculate d. The second process is the encryption process. It will convert the original plaintext, m, as unreadable message or ciphertext, c, from the equation: $c = m^e \bmod n$. However, m will be recovered by using the decryption equation: $m = c^d \bmod n$ in the last process.

In general, d which is always kept secretly by owner is the intruders' target. However, there is not any efficient algorithm to recover d for breaking RSA within the polynomial time whenever at least 1024 bits of n is chosen and all secreted parameters are very difficult to be calculated.

On the other hand, RSA may be easily attacked when at least one of the parameters becomes weak. In the sections 2.2 – 2.5, the algorithms for recovering d are shown when at least one of hidden parameters which is weak is occured. All of them are suitable for the different characteristics of the weak parameters.

2.2 Wiener's attack

Assuming, the communication devices in decryption part are a low power electronic equipment, these divices are not suitable to be selected as the machine to decrypt the ciphertext. The reason is that d is usually assigned very large. Therefore, one of the best solutions is to choose the new private key that the size is smaller than the old one. However,

in 1990, M. Wiener [9], [10] proposed the efficient method to recover a small private key. His method has very high performance when d is smaller than $\frac{1}{3}n^{0.25}$. In fact, if this situation is occured, it can be computed by searching $\frac{k}{d}$ which is the convergence of $\frac{e}{n}$. Moreover, after d is disclosed, p and q are also found. Therefore, a small private key is considered as one of the weak parameters. Furthermore, D. Boneh and G. Durfee [10] improved Wiener's algorithm and the result from the experiment reported that it can recover d, although d is still larger than $\frac{1}{3}n^{0.25}$. In fact, to avoid an attack by using Boneh and Durfee's method, d should be larger than $n^{0.292}$.

2.3 Brute force attack

Brute force attack [16] is the simplest method to explore d. The main process is to find an integer which is equal to d. In general, the first initial value which is selected as the exponent of modular exponentiation is begun as 3 and it is increased by two whenever the result from decryption equation is not equal to m until the target is found. In fact, the reason that the exponent must be increased by two is to skip all even numbers out of the computation, because d is always an odd number. Therefore, it implies that if d is a small integer, the process can be finished very fast by using brute force attack.

2.4 High value of Private key

After Wiener's attack and some improved methods were proposed, d must be assigned very large to increase the security. However, it affects to get the result directly, because the process becomes slow. In 2017, the improvement of decryption equation [13] was proposed to speed up RSA's decryption process. Considering at the modified equation, the inversion of ciphertext modulo n , $c^{-1} \bmod n$, is selected as the base instead of c and d is replaced by $x = \Phi(n) - d$. Therefore, the equation to recover m is changed as $m = (c^{-1})^x \bmod n$. In fact, this method is suitable for a high private key because x becomes small. Therefore, if d is a large integer, m can be recovered very fast by using the above equation. In contrast, this equation may become the weak point for attackers to solve this problem. Assuming, d is a large integer, x becomes small, d can be found easily by using brute force attack with $c^{-1} \bmod n$ as the base. Therefore, the large private key may become one of the weak parameters.

2.5 Integer Factorization Algorithms

Integer Factorization is another strategy to retrieve d after p and q are found. It is distinguished as two groups. For the first group, it is called the general purposed group. All algorithms are based on only size of n. Number Field Sieve (NFS) [17] which is one of efficient algorithms in this group is considered as the best integer factorization algorithm. In fact, it has very high performance when n is large. However, if n is higher than 1024 bits, then NFS becomes an inefficient algorithm to find two prime factors. The second group is called the special purposed group. In fact, the performance of each algorithm in this group is

based on the different characteristics of the weak parameters. Although, NFS is considered as the best algorithm, it is not guaranteed as the fastest algorithm for all values of n. In fact, if there is at least one weak parameter that responds well to the algorithm in special purposed group, then it may factor n faster than NFS. The examples of algorithms in special purposed group are shown as follows:

1) Trial Division Algorithm

Trial division algorithm (TDA) [18], [19] is the simplest integer factorization algorithm and it is divided into two techniques. The concept of this algorithm is to find the correct divisor of n. First, the divisor is begun as 3 and it is increased whenever the result is not the target answer. In addition, only odd numbers or prime numbers are chosen as the divisor to decrease time. With the reason above, it implies that this technique is suitable for a small prime factor. Hence, it is considered as another weak point to break RSA. However, the maximum odd integer which is less than $\sqrt{n}$ is chosen as the first divisor instead of 3 for the second method [20]. In addition, this number will be decreased whenever it is not the real prime factor. Then, p which is very close to $\sqrt{n}$ is the weak parameter for the second method.

2) Pollard's *p*-1

Pollard's p-1 [21] is one of the strategies in special purposed group. It was discovered by J. Pollard in 1974. The main idea of this algorithm is improved from Fermat's little theorem [22]. In addition, this algorithm can recover both of p and q very fast whenever all prime factors of $p-1$ or $q-1$ are small. In general, all of them are examined as the weak point of Pollard's p - 1.

3) Fermat's Factorization

Fermat's Factorization algorithm (FFA) is the factoring algorithm which was discovered by Pirre der Fermat in 1600 [21]. During that time, he found that the equation of n which is the multiplication between p and q can be rewritten as the difference between two perfect square numbers. In addition, the algorithm to recover both of them was proposed in that time. In general, Fermat's equation is very suitable for the same size of p and q especially the result of $q-p$ is very close to 0. Furthermore, many improvement algorithms were presented to leave some unrelated loops out of the computation such as [24], [25].

4) VFactor

VFactor [26] is the integer factorization algorithm that was presented by P. Sharma et. al. in 2012. It has very high performance when $q-p$ is close to 0, the characteristics of p and q are similar to FFA. That means, the weak point is that p and q are very close to each others. To implement VFactor, two odd integers are chosen as the initial values. One is the maximum integer which is less than $\sqrt{n}$ and the other is the minimum integer which is larger than $\sqrt{n}$. The main process is about the multiplication between these integers. If the result is equal to n, both of them are certainly prime factors. On the other hand, one of them must be changed until the target is found. In addition, the improvement of VFactor [27], [28] were proposed to skip some loops to decrease computation time.

Therefore, all algorithms which are in the special proposed group and are mentioned in this section can recover d very fast when one of hidden parameters is weak.

3. The Proposed Method

In this paper, the new methodology to recover m without disclosing d and the new algorithm to find d are proposed. Assuming $\Phi(n) = ad + b$, where $a,b \in \mathbb{Z}$, $a \mid b$ and $a \mid \Phi(n)$, if a is not too large and the result of $\frac{b}{a}$ is a small integer, then both of them become the new weakness of RSA which is very easy to be solved by using the proposed method. Furthermore, after a and b are disclosed, they can be selected to estimate the new initial value of d.

Theorem 1: Assigning $a,b,h \in \mathbb{Z}$ and $\Phi(n) = ad + b$, where $a \mid b$ and $a \mid \Phi(n)$, if $m = h^a$, then m can be recovered by using the following equation: $m = (c^{-1})^{\frac{b}{a}} \bmod n$

Proof:

From, $c^{\Phi(n)} \bmod n = c^{ad+b} \bmod n$

$= (c^d)^a * c^b \bmod n$

From Euler's Theorem, $c^{\Phi(n)} \bmod n = 1$, when c is relatively prime to n, then

$(c^d)^a * c^b \bmod n = 1$

That means, $(c^d)^a * c^{-1} * c^b \bmod n = c^{-1} \bmod n$

Or, $(c^d)^a * (c^{-1})^b * c^b \bmod n = (c^{-1})^b \bmod n$

$c^{da} * c^{-b} * c^b \bmod n = (c^{-1})^b \bmod n$

Because, $c^{-b} * c^b \bmod n = 1$

Then, $c^{da} \bmod n = (c^{-1})^b \bmod n$

Because $m = h^a$, it implies that the result of $(h^{ea})^{\frac{1}{a}}$ is always an integer. Therefore,

$$c^{\frac{1}{a}} \bmod n = (m^e)^{\frac{1}{a}} \bmod n$$

$$= (h^{ae})^{\frac{1}{a}} \bmod n$$

$$= h^e \bmod n$$

Because $a \mid b$, the result of $\frac{b}{a}$ is always an integer.

Therefore,

$$(c^{ad})^{\frac{1}{a}} \bmod n = ((c^{-1})^b)^{\frac{1}{a}} \bmod n$$

Or, $$c^d \bmod n = (c^{-1})^{\frac{b}{a}} \bmod n$$

Therefore,

$$c^d \bmod n = (c^{-1})^{\frac{b}{a}} \bmod n \quad (1)$$

In fact, a and b must have the same type, both of them are either odd numbers or even numbers. The proof will be shown in theorem 2. Furthermore, a must be the divisor of $\Phi(n)$. In depth, if it is not in the condition, there is certainly the remainder from result of $d =$

$\frac{\Phi(n)-b}{a}$ that is impossible.

Theorem 2: *a* and *b* must be the same type, both of them are either odd numbers or even numbers.

Proof: First, assuming the type of *a* is different from *b*, there are two cases as follows:

Case 1: *a* is an even number and *b* is an odd number

Assuming $a = 2x$, $b = 2y + 1$ and $d = 2z + 1$, then

$$\begin{aligned} ad + b &= (2x)*(2z + 1) + 2y + 1 \\ &= 4xz + 2x + 2y + 1 \\ &= 2(2xz + x + y) + 1 \\ &= 2u + 1, \text{ where } u = 2xz + x + y \end{aligned}$$

Then, $ad + b$ is an odd number. However, $\Phi(n)$ is always an even number. Therefore, it becomes the contradiction.

Case 2: *a* is an odd number and *b* is an even number

Assuming $a = 2x + 1$, $b = 2y$ and $d = 2z + 1$, then

$$\begin{aligned} ad + b &= (2x + 1)*(2z + 1) + 2y \\ &= 4xz + 2x + 2z + 1 + 2y \\ &= 2(2xz + x + y + z) + 1 \\ &= 2u + 1, \text{ where } u = 2xz + x + y + z \end{aligned}$$

The same reason with case 1 that $\Phi(n)$ is always an even number. Therefore, the contradiction is occured.

From case 1 and case 2, the conclusion is that the contradiction will be happened whenever *a* and *b* are the different type.

Next, the similar type of *a* and *b* is assummed, there are also two cases as follows:

Case 3: *a* and *b* are an odd number

Assuming $a = 2x + 1$, $b = 2y + 1$ and $d = 2z + 1$, then

$$\begin{aligned} ad + b &= (2x + 1)*(2z + 1) + 2y + 1 \\ &= 4xz + 2x + 2z + 1 + 2y + 1 \\ &= 2(2xz + x + y + z + 1) \\ &= 2u, \text{ where } u = 2xz + x + y + z + 1 \end{aligned}$$

Case 4: *a* and *b* are an even number

Assuming $a = 2x$, $b = 2y$ and $d = 2z + 1$, then

$$\begin{aligned} ad + b &= (2x)*(2z + 1) + 2y \\ &= 4xz + 2x + 2y \\ &= 2(2xz + x + y) \\ &= 2u, \text{ where } u = 2xz + x + y \end{aligned}$$

The information from case 3 and case 4 shows that the result of $ad + b$ is always an even number. Then, it is possible to be $\Phi(n)$. Therefore, both of *a* and *b* must be the same type.

Example 1: Assuming n = 4624614191 (46279*99929), $\Phi(n)$ = 4624467984, e = 1761702089 and d = 1541489321, encrypting m = 8 (2^3) , 27 (3^3) and 64 (4^3) and recovering all of them.

Sol:

Encryption Process:

Encrypting $m_1 = 8$: $c_1 = 8^{1761702089} \bmod 4624614191 = 4582115474$

Encrypting $m_2 = 27$: $c_2 = 27^{1761702089} \bmod 4624614191 = 4250185739$

Encrypting $m_3 = 64$: $c_3 = 64^{1761702089} \bmod 4624614191 = 2498965230$

Decryption Process:

Because of 4624467984 = 3*1541489321 + 21, 3 | 21 and 3 | 4624467984, then a = 3, b = 21 and the exponent is $\frac{21}{3} = 7$.

Decrypting c_1 = 4582115474: because of 4063456358 * 4582115474 mod 4624614191 = 1, then $m_1 = 4063456358^7 \bmod 4624614191 =$ **8**

Decrypting c_2 = 4250185739: because of 4038271146* 4250185739 mod 4624614191 = 1, then $m_2 = 4038271146^7 \bmod 4624614191 =$ **27**

Decrypting c_3 = 2498965230: because of 3200318111* 2498965230 mod 4624614191 = 1, then $m_3 = 3200318111^7 \bmod 4624614191 =$ **64**

Example 2: Assuming n = 4624614191 (46279*99929), $\Phi(n)$ = 4624467984, e = 105101545 and d = 1156116985, encrypting m = 16 (2^4), 81 (3^4) and 256 (4^4) and recovering all of them.

Sol:

Encryption Process:

Encrypting $m_1 = 16$: $c_1 = 16^{105101545} \bmod 4624614191 = 3695640335$

Encrypting $m_2 = 81$: $c_2 = 81^{105101545} \bmod 4624614191 = 827143163$

Encrypting $m_3 = 256$: $c_3 = 256^{105101545} \bmod 4624614191 = 1504158843$

Decryption Process:

Because of 4624467984 = 4*1156116985 + 44, 4 | 44 and 4 | 4624467984, then a = 4, b = 44 and the exponent is $\frac{44}{4} = 11$.

Decrypting c_1 = 3695640335: because of 4330254396 * 3695640335 mod 4624614191 = 1, then $m_1 = 4330254396^{11} \bmod 4624614191 =$ **16**

Decrypting c_2 = 827143163: because of 2428627615* 827143163 mod 4624614191 = 1, then $m_2 = 2428627615^{11} \bmod 4624614191 =$ **27**

Decrypting c_3 = 1504158843: because of 1756256207* 1504158843 mod 4624614191 = 1, then $m_3 = 1756256207^{11} \bmod 4624614191 =$ **256**

The information from example 1 and example 2 shows that the new equation in the theorem 1 can be chosen to recover m. However, the correct result will be occurred whenever m must be generated from h^a. The example 3 shows that the result becomes an incorrect answer when the pattern of m cannot be written as h^a.

Example 3: From n, $\Phi(n)$, e and d which are in example 1, encrypting m = 11 and recovering this value.

Sol:

Encryption Process:

Encrypting m = 11: c = $11^{1761702089}$ mod 4624614191 = 4348997977

Decryption Process:

Decrypting c = 4348997977: because of 2334698442 * 4348997977 mod 4624614191 = 1, then m' = 2334698442^7 mod 4624614191= **182070649 → it is the wrong answer**

In fact, the decrytion process in example 3 cannot recover m = 11 because 11 cannot be written as h^3.

Nevertheless, some values which are not written as the form h^a can be recovered by using the proposed equation, it is shown in example 4.

Example 4: From n, $\Phi(n)$, e and d which are in example 1, encrypting m = 15 and recovering this value.

Sol:

Encryption Process:

Encrypting m = 15: c = $15^{1761702089}$ mod 4624614191 = 4542134671

Decryption Process:

Decrypting c = 4542134671: because of 1511425060 * 4542134671 mod 4624614191 = 1, then m = 1511425060^7 mod 4624614191= **15**

From the information in the examples above, it implies that if $m = h^a$, it can be always recovered by using the proposed equation. On the other hand, the result from decryption process may be the wrong answer when $m \neq h^a$.

Assuming, the form of d is in the condition of this research, the concept to find $i = \frac{b}{a}$ is as follows. First, it is the process to assign i = 1 as the initial exponent for the proposed equation. The next process is to select m which is a small integer to find some possible plaintexts which are generated from m. In fact, all of them are m, m^2, m^3, ⋯,m^a, ⋯,m^l where $m^l < n < m^{l+1}$. Next, all chosen plaintexts are encrypted. The last process is to decrypt all ciphertexts by using i as the key and the inversion of ciphertext modulo n as the base. If there is the matching result, thoroughly rechecking with the other values to ensure that $i = \frac{b}{a}$. On the other hand, i will be increased to find the correct key whenever there is no matching result.

Example 5: Assuming n = 290831 (863*337), and e = 203815 are disclosed, find i,

Sol:

First, choosing m = 3

Because m^{11} = 177147 < n < m^{12} = 531441, therefore, m_1 = 3, m_2 = 3^2, m_3 = 3^3, ⋯, m_{11}= 3^{11} are chosen as the plaintexts for the implementation. The results are m_1=3, m_2=9, m_3=27, m_4=81, m_5=243, m_6=729, m_7=2187, m_8=6561, m_9=19683, m_{10}=59049, m_{11}=177147

The next process is to find $c_1 = m_1^e \bmod n = 255559$, $c_2 = m_2^e \bmod n = 229797$, $c_3 = m_3^e \bmod n = 60186$, $c_4 = m_4^e \bmod n = 185708$, $c_5 = m_5^e \bmod n = 94037$, $c_6 = m_6^e \bmod n = 54491$, $c_7 = m_7^e \bmod n = 95527$, $c_8 = m_8^e \bmod n = 139622$, $c_9 = m_9^e \bmod n = 184970$, $c_{10} = m_{10}^e \bmod n = 240814$, $c_{11} = m_{11}^e \bmod n = 18778$.

Then computing $c_1^{-1} \bmod n = 22749$, $c_2^{-1} \bmod n = 128652$, $c_3^{-1} \bmod n = 71995$, $c_4^{-1} \bmod n = 144894$, $c_5^{-1} \bmod n = 205883$, $c_6^{-1} \bmod n = 89943$, $c_7^{-1} \bmod n = 117222$, $c_8^{-1} \bmod n = 53839$, $c_9^{-1} \bmod n = 94070$, $c_{10}^{-1} \bmod n = 63932$, $c_{11}^{-1} \bmod n = 234068$

Assigning $j \in \mathbb{Z}$ where $j = 1, 2, 3, \cdots, 11$, the next process is to find the exponent i which is the correct result, the initial value is 1,

Loop 1 (i = 1): Computing $t_j = (c_j^{-1})^1 \bmod n$, that means $t_j = (c_j^{-1}) \bmod n$, however there is no t_j which is matched to m_j.

Loop 2 (i = 2): Computing $t_j = (c_j^{-1})^2 \bmod n$, then the results are $t_1 = 128652$, $t_2 = 144894$, $t_3 = 89943$, $t_4 = 53839$, $t_5 = 63932$, $t_6 = 278984$, $t_7 = 105027$, $t_8 = 216175$, $t_9 = 50063$, $t_{10} = 252581$, $t_{11} = 212351$, there is no t_j which is matched to m_j.

Loop 3 (i = 3): Computing $t_j = (c_j^{-1})^3 \bmod n$, then the results are $t_1 = 71995$, $t_2 = 89943$, $t_3 = 94070$, $t_4 = 278984$, $t_5 = 82558$, $t_6 = 50063$, $t_7 = 17102$, $t_8 = 170867$, $t_9 = 27$, $t_{10} = 198879$, $t_{11} = 101813$, there is no t_j which is matched to m_j.

Loop 4 (i = 4): Computing $t_j = (c_j^{-1})^4 \bmod n$, then the results are $t_1 = 144894$, $t_2 = 53839$, $t_3 = 278984$, $t_4 = 216175$, $t_5 = 252581$, $t_6 = 170867$, $t_7 = 32561$, $t_8 = 33052$, $t_9 = 213242$, $t_{10} = 182570$, $t_{11} = 182313$, there is no t_j which is matched to m_j.

Loop 5 (i = 5): Computing $t_j = (c_j^{-1})^5 \bmod n$, then the results are $t_1 = 205883$, $t_2 = 63932$, $t_3 = 82558$, $t_4 = 252581$, $t_5 = 97068$, $t_6 = 198879$, $t_7 = 290329$, $t_8 = 182570$, $t_9 = 188377$, $t_{10} = 144717$, $t_{11} = 6654$, there is no t_j which is matched to m_j.

Loop 6 (i = 6): Computing $t_j = (c_j^{-1})^6 \bmod n$, then the results are $t_1 = 89943$, $t_2 = 278984$, $t_3 = 50063$, $t_4 = 170867$, $t_5 = 198879$, $t_6 = 213242$, $t_7 = 193249$, $t_8 = 170923$, $t_9 = 729$, $t_{10} = 131472$, $t_{11} = 88467$, there is no t_j which is matched to m_j.

Loop 7 (i = 7): Computing $t_j = (c_j^{-1})^7 \bmod n$, then the results are $t_1 = 117222$, $t_2 = 105027$, $t_3 = 17102$, $t_4 = 32561$, $t_5 = 290329$, $t_6 = 193249$, $t_7 = 207688$, $t_8 = 139726$, $t_9 = 231745$, $t_{10} = 252004$, $t_{11} = 126556$, there is no t_j which is matched to m_j.

Loop 8 (i = 8): Computing $t_j = (c_j^{-1})^8 \bmod n$, then the results are $t_1 = 53839$, $t_2 = 216175$, $t_3 = 170867$, $t_4 = 33052$, $t_5 = 182570$, $t_6 = 170923$, $t_7 = 139726$, $t_8 = 73468$, $t_9 = 142052$, $t_{10} = 245652$, $t_{11} = 118303$, there is no t_j which is matched to m_j.

Loop 9 (i = 9): Computing $t_j = (c_j^{-1})^9 \bmod n$, then the results are $t_1 = 94070$, $t_2 = 50063$, **$t_3 = 27 = m_3$**, $t_4 = 213242$, $t_5 = 188377$, **$t_6 = 729 = m_6$**, $t_7 = 231745$, $t_8 = 142052$, **$t_9 = 19683 = m_9$**, $t_{10} = 149664$, $t_{11} = 54601$

From all results in 9th Loop, the result of a may be equal to 3, 6 or 9. In fact, both of m_6 and m_9 can be rewritten as the form h^3 as follows: $m_6 = m^3 * m^3 = (m^2)^3$ and $m_9 = m^3 * m^3 * m^3 = (m^3)^3$. Therefore, it implies that a is equal to 3 and then $b = 3*9 = 27$. However, to ensure a

= 3 is the correct result, the other plaintexts should be selected to verify.

Assuming $m_1^{'}$ = 125 (5^3) and $m_2^{'}$ = 1331 (11^3) are chosen, then

Encryption Process:

Encrypting $m_1^{'}$ = 125: $c_1^{'} = 125^{203815} \bmod 290831 = 194710$

Encrypting $m_2^{'}$ = 1331: $c_2^{'} = 1331^{203815} \bmod 290831 = 124812$

Decryption Process:

Decrypting $c_1^{'}$ = 194710: because 55503* 194710 mod 290831 = 1, then

$m_1^{'} = 55503^9 \bmod 4624614191 =$ **<u>125</u>**

Decrypting $c_2^{'}$ = 124812: because 181922* 124812 mod 290831 = 1, then

$m_2^{'} = 181922^9 \bmod 4624614191 =$ **<u>1331</u>**

Therefore, i = 9 with a = 3, b = 27

In addition, the total loops to find a and b are 11 (number of $c^{-1} \bmod n$) * 9 (number of loops) = 99.

Furthermore, if a and b are disclosed, the initial integer can be estimated to decrease loops. After this integer is found, d can be recovered by using brute force attack.

Theorem 3: Assuming a, b are disclosed, then $d \le \dfrac{n - 2\left\lceil \sqrt{n} \right\rceil - (b-1)}{a}$

Proof:

From, $$\begin{aligned} ad + b &= \Phi(n) \\ &= (p-1)(q-1) \\ &= n - (p+q) + 1 \end{aligned}$$

Because, $p + q \ge 2\left\lceil \sqrt{n} \right\rceil$,

$$ad + b \le n - 2\left\lceil \sqrt{n} \right\rceil + 1$$

$$ad \le n - 2\left\lceil \sqrt{n} \right\rceil - (b-1)$$

Then, $$d \le \frac{n - 2\left\lceil \sqrt{n} \right\rceil - (b-1)}{a}$$

Therefore, $$d \le \frac{n - 2\left\lceil \sqrt{n} \right\rceil - (b-1)}{a}$$

Therefore, the scope to find d is narrowed by using the following equation,

$$d \le \frac{n - 2\left\lceil \sqrt{n} \right\rceil - (b-1)}{a} \tag{2}$$

Assigning $d_i = \frac{n-2\lceil\sqrt{n}\rceil-(b-1)}{a}$, it can be chosen as the initial value to find d and it is always decreased by two (d is always an odd number) whenever the decrypted result is still incorrect.

Example 6: From the solution in example 5, the aim of this example is to find the initial value by using the equation in theorem 3

Sol:

From,
$$d_i = \frac{n-2\lceil\sqrt{n}\rceil-(b-1)}{a}$$
$$= \frac{290831-2\lceil\sqrt{290831}\rceil-(27-1)}{3}$$
$$= 96575$$

In fact, the real value of d = 96535. Then, the distance between d_i and d is only 96575 – 96535 = 40. Therefore, there are only 20 loops of the computation to find the target. On the other hand, if the process is implemented by using traditional brute force attack choosing d_o = 3 as the initial exponent, the loops are 48266. In depth, it is very higher than the proposed equation.

4. Experimental Results

The aim of this section is to ensure the hypothesis that the proposed method can recover both of m and d very fast when a is large, but 2^a is still less than n, and $\frac{b}{a}$ is small. Many pairs of e and d from the same value of n are chosen for the experiment. However, all values of d must generate a and b in the condition. In addition, 32 bits of n is chosen randomly. In fact, n in this experiment is 2078092697 (55619 * 37363) and then $\Phi(n)$ is 2077999716. In addition, m = 2 is selected as the base of original plaintext because it is the smallest plaintext. Because of 2^{30} = 1073741824 < n < 2^{31} = 2147483648, then the maximum value of a is 30. Despites of 2077999716 = 2^2*3*13*479*27809. Therefore, there are 6 possible values of a consist of 2, 3, 4 (2*2), 6 (2 * 3), 12 (2 * 2 * 3), 13 and 26 (2 *13).

From **Table 1**, Number of steps to find a, b can be estimated from the following equation:

$$t_{ab} = l * \frac{b}{a} \quad (3)$$

Where, t_{ab} is represented as number of steps to find a, b
l is number of all possible plaintexts which are from $m = m^1, m^2, \cdots, m^l$.

Table 1. Number of steps to find *a*, *b* and *d* when n = 2078092697 and $\Phi(n)$ = 2077999716

Row	A pair of keys		*a*	*b*	Steps to find *a*, *b*	Steps to find *d*
	e	***d***				
1	230888857	692666569	3	9	90	301
2	1108266515	692666567	3	15	150	301
3	538740667	692666563	3	27	270	301
4	881575637	692666561	3	33	330	301
5	1818249751	519499927	4	8	60	226
6	1688374769	519499925	4	16	120	226
7	1103937349	519499921	4	32	240	226
8	2026049723	519499919	4	40	300	226
9	1962555287	346333283	6	18	90	150
10	1533761695	346333279	6	42	210	150
11	192407381	346333277	6	54	270	150
12	253977743	346333271	6	90	450	150
13	1471916465	173166641	12	24	60	75
14	1760527537	173166637	12	72	180	75
15	757604063	173166635	12	96	240	75
16	1486347019	173166631	12	144	360	75
17	1214830603	159846127	13	65	150	69
18	388197749	159846125	13	91	210	69
19	348755197	159846121	13	143	330	69
20	1561573751	159846119	13	169	390	69
21	1934138197	79923061	26	130	150	34
22	947659211	79923059	26	182	210	34
23	1867293451	79923055	26	286	330	34
24	1961189081	79923053	26	338	390	34

For example, in 1st Row, l = 30 and $\frac{b}{a}$ = 3, therefore t_{ab} = 90. Furthermore, total steps to find d, t_d , can be calculated by using the following equation:

$$t_d = \frac{d_i - d}{2} \tag{4}$$

However, d_i must be computed as first. For the example, d_i in 1st Row is d_i = $\frac{2078092697 - 2\left\lceil \sqrt{2078092697} \right\rceil - (9-1)}{3}$ = 692667171.67. In fact, d is always an odd integer and d_i is larger than d. Therefore, d_i can be estimated as 692667171. Then,

$$t_d = \frac{692667171 - 692666569}{2} = 301$$

The information in this table shows that from the same value of a, the number of steps to find a and b are larger when b is higher. The reason is that the exponent for the new decryption equation is from $\frac{b}{a}$. However, for the same value of a, after a and b are found, the number of steps to find d from the different values of b are same. In addition, it will be shown in theorem 4.

Theorem 4: From the condition in theorem 1, the total steps to find d from the same value of a are not changed.

Proof: From

$$t_d = \frac{d_i - d}{2}$$

$$= \frac{n - 2\left\lceil \sqrt{n} \right\rceil - (b-1)}{a} - \frac{\Phi(n) - b}{a}$$

$$= \frac{n - 2\left\lceil \sqrt{n} \right\rceil - \Phi(n) + 1}{a}$$

From this equation, it implies that any values of b do not affect to t_d. Therefore, the total steps to find d from the same value of a is not changed.

Example 7: From the value of n in **Table 1**, t_d is always equal to $\frac{2078092697 - 2\left\lceil \sqrt{2078092697} \right\rceil - 2077999716 + 1}{3} \approx 301$, when $a = 3$.

Furthermore, t_d is very small when a is large. On the other hand, if a is too large, the solution is not found by using the proposed method. The reason is that 2^a which is the smallest plaintext becomes larger than n.

Therefore, to avoid an attack by using the proposed method, the parameters should not be assigned in the condition of theorem 1. In fact, the condition is as follows. If $\Phi(n) = ad + b$, then both of $\Phi(n)$ and b must not be divided by a.

The next experiment is about the comparison between the proposed method and some algorithms in special proposed group in order to finish the process. The selected algorithms to compare with the proposed method consist of:

1) The improvement of FFA which was proposed in [23]
2) The improvement of TDA which was proposed in [18]
3) The improvement of VFactor which was proposed in [28]
4) Bruteforce attack that the exponent must be always an odd number
5) The algorithm for high value of private key which was proposed in [13]

In fact, the objective of this experiment is to ensure that the proposed method is the best algorithm when both of a and b are weak, they respond to the proposed method well. In fact, a = 3 and small value of b are chosen for this experiment. In addition, bits length of n consist of 128, 256 and 512.

However, the total loops to find d from g bits of n are usually very large when they are compared with the others from g – 1 bits of n. Therefore, the exponent of each result is chosen for the comparison instead of the real result.

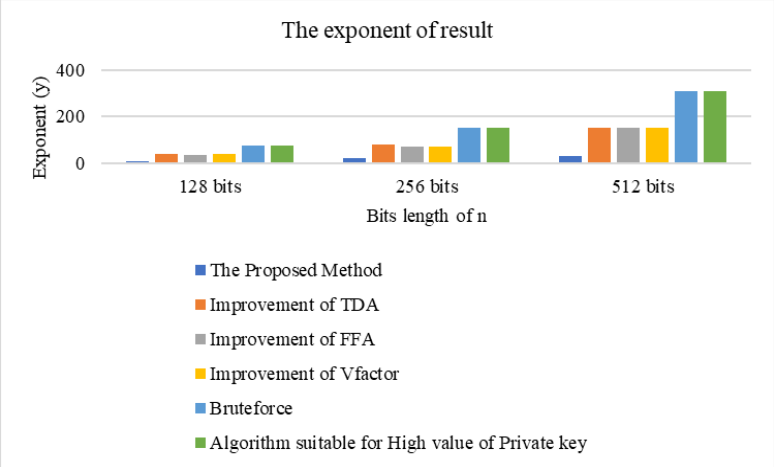

Fig. 1. the exponent of result from each algorithm to discover d

Assuming the total loops to find d are written as the form x x 10^y, where $x \in \mathbb{R}$ and $y \in \mathbb{Z}$, the information in **Fig. 1** is that the exponent, y, is shown at y-axis and three different bits length of n are represented at x-axis. For example, y = 75 when 128 bits of n is chosen. Then, it implies that the total loops are about 10^{75}.

The experimental results show that the proposed method requires the smallest loops of the computation. The reason is that both of a and b in this experiment are good responses to the proposed method.

However, the proposed method becomes an inefficient algorithm when a and b are strong, b is very large and a is very small. In addition, if a is not a divisor of b or $\Phi(n)$, then d cannot be recovered by using the proposed method, because the result of $\frac{b}{a}$ is not an integer. Therefore, the proposed method must be organized in the special proposed group.

5. Conclusion

This research has disclosed the new weakness of RSA. The condition to create the weak point is that Euler's totient value, $\Phi(n)$, must be written as the form $\Phi(n) = ad + b$ where d is the private key, $a, b \in \mathbb{Z}$ and a divides both of $\Phi(n)$ and b. The experimental results show that total steps to recover d is small when a is large and $\frac{b}{a}$ is small. However, if a is too large, the solution may not be found because the smallest possible value of plaintext is larger than the modulus. Therefore, to avoid an attack by using the proposed method, a and b should not be in the condition.

References

[1] C.Chen, Y. Xiang, J.Du and Z. Cheng, "An Improved Data Cache Timing Attack against RSA Based on Hidden Markov Model," *Journal of Computers*, vol 30, pp. 87 - 95, 2019. Article (CrossRef Link).

[2] Ritambhara, A. Gupta and M. Jaiswal, "An enhanced AES algorithm using cascading method on 400 bits key size used in enhancing the safety of next generation internet of things (IOT)," in *Proc. of International Conference on Computing, Communication and Automation*, pp. 422-427, May 5 – 6, 2017. Article (CrossRef Link).

[3] Y. Yuan, Y. Yang, L. Wu and X. Zhang, "A High Performance Encryption System Based on AES Algorithm with Novel Hardware Implementation," in *Proc. of IEEE International Conference on Electron Devices and Solid State Circuits, Shenzhen*, pp. 1-2, June 6 – 8, 2018. Article (CrossRef Link).

[4] X. Zhang, R. Lu , H. Zhang, and C.Xu, "A New Public Key Encryption Scheme based on Layered Cellular Automata," *KSII Transactions on Internet and Information Systems*, vol. 8, no. 10, pp. 3572-3590, 2014. Article (CrossRef Link).

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

[6] L.D. Tran, T.D. Tran, D. Choi and T.D. Nguyen, "RSA-type Algebra Structures," *KSII Transactions on Internet and Information Systems*, vol. 10, no. 6, 2835-2850, 2016. Article (CrossRef Link).

[7] P. V. V. Kishore, N. Venkatram, C. Sarvya and L. S. S. Reddy, "Medical image watermarking using RSA encryption in wavelet domain," in *Proc. of International Conference on Networks & Soft Computing*, pp. 258-262, August 19 – 20, 2014. Article (CrossRef Link).

[8] B. J. S. Kumar, A. Nair and V. K. R. Raj, "Hybridization of RSA and AES algorithms for authentication and confidentiality of medical images," in *Proc. of International Conference on Communication and Signal Processing*, pp. 1057-1060, April 6-8, 2017. Article (CrossRef Link).

[9] M. Wiener, "Cryptanalysis of short RSA secret exponents," *IEEE Transactions on Information Theory*, vol. 36, pp. 553-558, 1990. Article (CrossRef Link).

[10] M.E. Wu, C.M. Chen, Y.H. Lin and H.M. Sun, "On the Improvement of Wiener Attack on RSA with Small Private Exponent," *The Scientific World Journal*, pp. 1 – 9, 2014. Article (CrossRef Link).

[11] D. Boneh, and G. Durfee, "Cryptanalysis of RSA with Private Key d less than $N^{0.292}$," *Lecture Notes in Computer Science*, vol. 1592, pp. 1 – 11, 1999. Article (CrossRef Link) .

[12] C. Duta, L. Gheorghe and N. Tapus, "Framework for evaluation and comparison of integer factorization algorithms," in *Proc. of SAI Computing Conference*, pp. 1047-1053, July 13-15, 2016. Article (CrossRef Link) .

[13] K. Somsuk, "The New Equation for RSA's Decryption Process Appropriate with High Private Key Exponent," in *Proc. of International Computer Science and Engineering Conference*, pp. 1-5, November 15 – 18, 2017. Article (CrossRef Link) .

[14] M.M. Asad, L. Marouf, Q. A. Al-Haija, A. Alshuaibi, "Performance Analysis of 128-bit Modular Inverse Based Extended Euclidean Using Altera FPGA Kit," *Procedia Computer Science*, vol. 160, pp. 543-548, 2019. Article (CrossRef Link) .

[15] Q. Zhou, C. Tian, H. Zhang, J. Yu, F. Li, "How to securely outsource the extended euclidean algorithm for large-scale polynomials over finite fields," *Information Sciences*, vol. 512, pp. 641-660, 2020. Article (CrossRef Link) .

[16] V. Shende, G. Sudi and M. Kulkarni, "Fast cryptanalysis of RSA encrypted data using a combination of mathematical and brute force attack in distributed computing environment," in *Proc. of IEEE International Conference on Power, Control, Signals and Instrumentation Engineering*, pp. 2446-2449, September 21 – 22, 2017. Article (CrossRef Link) .

[17] S. M. Hamdi, S. T. Zuhori, F. Mahmud and B. Pal, "A Compare between Shor's quantum factoring algorithm and General Number Field Sieve," in *Proc. of International Conference on Electrical Engineering and Information & Communication Technology*, pp. 1-6, April 10 – 12, 2014. Article (CrossRef Link) .

[18] S. Murat, "Generalized Trial Division," *International Journal of Contemporary Mathematical Science*, vol. 6(2), pp. 59 – 64, 2011.
[19] N. Lal, A. P. Singh and S. Kumar, "Modified trial division algorithm using KNJ-factorization method to factorize RSA public key encryption," in *Proc. of International Conference on Contemporary Computing and Informatics*, pp. 992-995, November 27 – 29, 2014. Article (CrossRef Link) .
[20] K. Somsuk, T. Chiawchanwattana and C. Sanemueang, "Estimating the new Initial Value of Trial Division Algorithm for Balanced Modulus to Decrease Computation Loops," in *Proc. of International Joint Conference on Computer Science and Software Engineering*, pp. 143-147, July 10 – 12, 2019. Article (CrossRef Link) .
[21] S. Sarnaik, R. Bhakkad and C. Desai, "Comparative study on Integer Factorization Algorithm-Pollard's RHO and Pollard's P-1," in *Proc. of the International Conference on Computing for Sustainable Global Development*, pp.677 – 679, March 11 - 13, 2015.
[22] G. Xiang and Z. Cui, "The Algebra Homomorphic Encryption Scheme Based on Fermat's Little Theorem," in *Proc. of International Conference on Communication Systems and Network Technologies*, pp.978 – 981, May 11 – 13, 2012. Article (CrossRef Link) .
[23] K. Somsuk, "The improvement of initial value closer to the target for Fermat's factorization algorithm," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 21, no. 7-8, pp. 1573 – 1580, 2018. Article (CrossRef Link) .
[24] M.E. Wu, R. Tso and H.M. Sun, "On the improvement of Fermat factorization using a continued fraction technique," *Future Generation Computer Systems*, vol. 30(1), pp.162 – 168, 2014. Article (CrossRef Link).
[25] K. Omar and L. Szalay, "Sufficient conditions for factoring a class of large integers," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 13, pp. 95-103, 2010. Article (CrossRef Link).
[26] P. Sharma, A. K. Gupta and A. Vijay, "Notice of Violation of IEEE Publication Principles: Modified Integer Factorization Algorithm Using V-Factor Method," in *Proc. of International Conference on Advanced Computing & Communication Technologies*, pp. 423-425, January 7 – 8, 2012. Article (CrossRef Link).
[27] K. Somsuk and S. Kasemvilas, "MVFactor: A method to decrease processing time for factorization algorithm," in *Proc. of International Computer Science and Engineering Conference*, pp. 339-342, September 4 – 6, 2013. Article (CrossRef Link) .
[28] K. Somsuk, "MVFactorV2: An improved integer factorization algorithm to speed up computation time," in *Proc. of International Computer Science and Engineering Conference*, pp. 308-311, July 30 – August 1, 2014. Article (CrossRef Link) .

Kritsanapong Somsuk is an assistant professor of the department of Computer and Communication Engineering in Faculty of Technology, Udon Thani Rajabhat University, Udon Thani, Thailand. He obtained his M.Eng. (Computer Engineering) from department of Computer Engineering in Faculty of Engineering, Khonkaen University, M.Sc. (Computer Science) from department of Computer Science in Faculty of Science, Khonkaen University and his Ph.D. (Computer Engineering) from department of Computer Engineering in Faculty of Engineering, Khonkaen University. The area of research interests include computer security, cryptography and integer factorization algorithms.