The alternative Method to Finish Modular Exponentiation and Point Multiplication Processes

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

Received April 4, 2021; revised May 30, 2021; accepted June 28, 2021;
published July 31, 2021

Abstract

The aim of this paper is to propose the alternative algorithm to finish the process in public key cryptography. In general, the proposed method can be selected to finish both of modular exponentiation and point multiplication. Although this method is not the best method in all cases, it may be the most efficient method when the condition responds well to this approach. Assuming that the binary system of the exponent or the multiplier is considered and it is divided into groups, the binary system is in excellent condition when the number of groups is small. Each group is generated from a number of 0 that is adjacent to each other. The main idea behind the proposed method is to convert the exponent or the multiplier as the subtraction between two integers. For these integers, it is impossible that the bit which is equal to 1 will be assigned in the same position. The experiment is split into two sections. The first section is an experiment to examine the modular exponentiation. The results demonstrate that the cost of completing the modular multiplication is decreased if the number of groups is very small. In tables 7 – 9, four modular multiplications are required when there is one group, although number of bits which are equal to 0 in each table is different. The second component is the experiment to examine the point multiplication process in Elliptic Curves Cryptography. The findings demonstrate that if the number of groups is small, the costs to compute point additions are low. In tables 10 – 12, assigning one group is appeared, number of point addition is one when the multiplier of a point is an even number. However, three-point additions are required when the multiplier is an odd number. As a result, the proposed method is an alternative way that should be used when the number of groups is minimal in order to save the costs.

Keywords: Modular Exponentiation, Modular Multiplication, Point Multiplication, Point Addition, Public Key Cryptography

http://doi.org/10.3837/tiis.2021.07.017 **ISSN : 1976-7277**

1. Introduction

One of two encryption techniques is asymmetric key cryptography, sometimes known as public key cryptography [1]. It can be selected to implement a variety of tasks such as data security, digital signature and key changing protocol. Using the encryption process, this method will convert the original plaintext into an unreadable message known as the ciphertext. The ciphertext is then sent to the receiver via the network channel. As a result, attackers cannot decipher the information embedded in the ciphertext, though they can intercept it over the network channel. In general, the receiver with the key can recover the original plaintext through the decryption process. In addition, a pair of keys that is mathematically related to each other must be chosen for the implementation. One is called the public key. It is revealed to all members of a group. The other key is known as the private key, and it is kept secretly by the owner. If one of the keys is chosen for encryption process, the other must be selected for decryption. In fact, to accomplish the task, the algorithms in public key cryptography require either a modular exponentiation process or a point multiplication process. RSA [2], the Diffie–Hellman Protocol, and Elgamal Cryptography [3] are three examples of modular exponentiation algorithms. Furthermore, the key must be selected as the exponent in modular exponentiation equation. That is, when the exponent is large, this process consumes high computation costs. The small key may be chosen to decrease the costs. Unfortunately, when the private key is a small integer, attackers can quickly recover it. Many efficient attacking algorithms are currently being developed to address this issue. In fact, they are intended to solve one of the following issues: Discrete Logarithm Problem (DLP) and the RSA's private key disclosing. The example algorithms to solve DLP are brute force attack, Baby-Step Giant-Step [4], Pohlig-Hellman Algorithm [5] and Index Calculus Algorithm [6]. Moreover, the example algorithms to recover RSA's private key are Wiener's Attack [7], [8], [9] and the new estimated initial value for brute force attack [10]. Therefore, the private key should be expanded to avoid being easily attacked. Although, many techniques were proposed to speed up modular exponentiation by using multi modular multiplications and modular squares instead of computing modular exponentiation directly, computation time remains high whenever the exponent' s Hamming Weight is large, because number of modular multiplications is based on Hamming Weight. In addition, Elliptic Curve Cryptography [11], [12], [25], [26] is an example of a point multiplication algorithm. For calculating the point multiplication equation, the key must be chosen as the multiplier of the point. That is, when the multiplier is large and the Hamming weight is high, this process consumes high costs.

The purpose of this paper is to present a new special algorithm for completing one of the modular exponentiation process and point multiplication process. The exponent in modular exponentiation or the multiplier in point multiplication will be replaced by two integers whose difference equals the exponent or multiplier. If the condition responds well to the given approach, it may be the fastest algorithm to finish the task.

2. Related Works

Four parts will be discussed in this section. The first part goes through several public key cryptography algorithms that call for modular exponentiation computing. The second part is the reviews about the algorithms to compute modular exponentiation. Moreover, the overviews for both of Elliptic Curve Cryptography and Point multiplication will be mentioned in Section 2.3 and 2.4, respectively. Assigning, H(z) is Hamming Weight of z, B(z) is the

binary value of z and L(z) is bits length of z, where $z \in ¢$.

2.1 The Algorithms requiring Modular Exponentiation Computing

In fact, there are three main algorithms in this group as follows:

1) Diffie – Hellman Protocol: this algorithm cannot be chosen to secure the secret information. However, it is represented as a secret method of exchanging the secret key for all algorithms in the symmetric key cryptography group. Assuming, Alice and Bob have to communicate the secret information to each other by using one of symmetric key algorithms and they have to use this protocol to exchange the secret key. **Table 1** shows the process to exchange the secret key between Alice and Bob.

Table 1. Diffie – Hellman Protocol

Alice	Bob
1. Alice chooses $a \in \{0, 1, 2, \cdots, p-2\}$	1. Bob chooses $b \in \{0, 1, 2, \cdots, p-2\}$
2. Alice computes $A = g^a \bmod p$	2. Bob computes $B = g^b \bmod p$
3. Alice sends A to Bob	3. Bob sends B to Alice
4. Alice computes $K = K_A = B^a \bmod p$	4. Bob computes $K = K_B = A^b \bmod p$

In fact, modular exponentiation occurs in step 2 on both of Alice and Bod.

2) Elgamal Cryptography: it was proposed by T. Elgamal in 1985. Moreover, Elgamal Cryptography can be chosen for both of data security and digital signature. For data security, it is divided into three processes as follows:

Process 1 (Key Generation Process): the process to generate a pair of keys as follows:

Step 1: Choose prime number, p, and the primitive root, g.

Step 2: Choose $a \in \{0, 1, 2, \cdots, p-2\}$ to compute $A = g^a \bmod p$

Public Parameters are $\{p, g, A\}$, Private Parameter is a

Process 2 (Encryption Process): the process to transform the plaintext as the ciphertext:

Step 1: Choose $b \in \{0, 1, 2, \cdots, p-2\}$ to compute B: $B = g^b \bmod p$

Step 3: Compute c: $c = A^b m \bmod p$, where m is the plaintext and c is the ciphertext

In fact, $\{c, B\}$ will be transferred to the receiver, while b will be kept secretly.

Process 3 (Decryption Process): the process to recover the plaintext by using the equation: $m = (B^{-1})^{a*} c \bmod p$.

3) RSA Cryptography: RSA was proposed in 1987. It is also chosen for both of data security and digital signature. For data security, the processes are divided into three parts.

Process 1 (Key Generation Process): the process to generate a pair of keys as follows:

Step 1: Choose two prime numbers, p and q, randomly

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

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

Step 4: Compute the private key, $d = e^{-1} \bmod \Phi(n)$ by using [13], [14], [15].

Public Parameters are $\{n, e\}$, Private Parameters are $\{p, q, \Phi(n), d\}$

Process 2 (Encryption Process): the process to transform the plaintext as the ciphertext by using the following equation, $c = m^e \bmod n$, where m is the plaintext and c is the ciphertext.

Process 3 (Decryption Process): the process to recover the plaintext by using the following equation: $m = c^d \bmod n$.

In fact, three algorithms above require modular exponentiation computing. In addition, the large exponent can avoid easily attacks. However, the process becomes slow. The next section is about the overviews of some algorithms to speed up modular exponentiation.

2.2 Overviews of Some Algorithms to Speed up Modular Exponentiation

In fact, many algorithms have been developed to accelerate the modular exponentiation process. Additionally, some methods can be chosen to accelerate all public key cryptography algorithms. However, some techniques can be applied with some algorithms as follows:

1) Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) [16], [17], [18] is a method that can be used in conjunction with RSA to speed up the decryption process. The private key is usually given a very large value. The time that required to complete the process is then directly affected. Therefore, the private key is divided into two integers. In fact, these integers will be selected as the exponents for modular exponentiation and the results will be combined to obtain the target.

Assigning, $d_p = d \bmod p - 1$, $d_q = d \bmod q - 1$, $m_p = c^{d_p} \bmod p$, $m_q = c^{d_q} \bmod q$, $y_p = p^{-1} \bmod q$ and $y_q = q^{-1} \bmod p$, then m can be recovered by using the equation: $m = (m_p * y_q * q + m_q * y_p * p) \bmod n$.

2) The improved RSA's decryption equation using the new private key

In 2016, a special equation [19] was proposed to recover plaintext encrypted with the RSA scheme. In fact, the new integer, d_t, will be chosen, as shown in the following equation: $d_t * e = x \bmod \Phi(n)$ and the plaintext can be recovered by using the equation: $m = \sqrt[x]{c^{d_t} \bmod n}$. The condition under which this method is more efficient than the traditional decryption process is that H(B(d_t)) is smaller than H(B(d)). However, if this equation is implemented, m must be assigned in the following scope: $1 < m^x < n$, where m is the plaintext.

3) The Improved RSA's decryption equation suitable for the Large Private Key

In 2017, an improved RSA decryption equation [20] that was suitable for the large private key was proposed to speed up the RSA decryption process. When the private key is large, this method may be the most efficient method. Assuming $x = \Phi(n) - d$, then the plaintext can be computed from: $m = (c^{-1})^x \bmod n$.

4) The Improved CRT to speed up RSA's decryption Process

In addition, the idea in [20] is included to apply with RSA to speed up decryption process. In fact, the technique in [21] is only the choice that will be selected when the appropriate condition is occurred. Assuming, $x_p = d_p - p$, and $y = c^{-1} \bmod p$, then $m_p = y^{x_p - 1} \bmod p$. If d_p is large, then this equation to find m_p is more suitable than the traditional equation. Furthermore, the condition for selecting the equation to find m_q is similar to the concept for finding m_p.

5) Fast Exponent

Fast Exponent [22] is the method to compute modular exponentiation. That is, it can be used with any public key cryptography scheme that requires this process. Furthermore, the outcome is derived from the computation of multi modular squares and modular multiplications. The central idea is that the exponent will be converted to a binary system. Assigning $L(d) = l$, therefore $d = \sum_{i=0}^{l-1} d_i = (d_{l-1}d_{l-2}d_{l-3}\cdots d_1 d_0)_2$ and the algorithm is as follows:

Algorithm: Fast Exponent
Input: c, n, $d = d_{l-1}d_{l-2}d_{l-3}\cdots d_1 d_0$
Output: $m = c^d \bmod n$

1) *len* ← Length of *d*
2) *t* ← c
3) *f* ← *t*
4) *i* ← 1
5) While *i* <= *len*-1 Do

6) $t \leftarrow t^2 \bmod n$
7) IF $d_i == 1$ Then
8) $f \leftarrow (f * t) \bmod n$
9) $i \leftarrow i + 1$
10) End While
11) $m \leftarrow f$

If the bit length of d is l and Fast Exponent is used to find the result, the number of modular squares is always $l - 1$. Number of modular multiplications is based on H(B(d)). Therefore, the costs to compute modular multiplications are high when Hamming Weight is large.

6) Akira's Method

In 2000, H. Akira [23] proposed the idea to speed up modular multiplication. Then, it implies that the algorithms requiring modular exponentiation can select Akira's method to accelerate the task. Assuming, n is an odd number and it is the modulus. To use Akira's method, all prime factors of n+2 should reveal first. However, if n is large, then it is very difficult to find all prime factors of $n + 2$.

2.3 Overviews of Elliptic Curve Cryptography

N. Koblitz and V. Miller presented Elliptic Curve Cryptography (ECC) as a public key cryptography, in 1985. Because it does not require modular exponentiation, this approach differs from the public key techniques in section 2.1. In fact, point multiplication is the main process to find the target. Assigning, $P = (x_p, y_p)$ is one of the points on the curve from the following equation for ECC over Finite Field: $y^2 \equiv x^3 + ax + b \bmod p$, where $a, b \in \mathbb{C}_p$ and $4a^3 + 27b^2 \bmod p \neq 0$, then point doubling process ($Q = (x_q, y_q) = 2P$) can be computed from: $x_q = m^2 - 2x_p \bmod p$ and $y_q = m*(x_p - x_q) - y_p \bmod p$, where $m = \frac{3x_p^2 + a}{2y_p} \bmod p$. On the other hand, point addition ($R = (x_r, y_r) = P + Q$) can be computed from: $x_r = m^2 - x_q - x_p \bmod p$ and $y_r = m*(x_p - x_r) - y_p \bmod p$ where $m = \frac{y_q - y_p}{x_q - x_p} \bmod p$. In addition, the point $T = -P$ is $(x_p, -y_p)$.

2.4 Point Multiplication Computing

For ECC, the main process for encrypting the plaintext and decrypting the ciphertext is point multiplication [24]. This procedure necessitates a large number of point doublings and point additions, which pay significant computational costs. Assuming, $T = k*P$, k is transformed as the binary system before using the efficient algorithm known as the point multiplication algorithm to speed up point multiplication. The length of k is equal to the number of point doublings, but the length of H(B(k)) is equal to the number of point additions.

Algorithm: Point multiplication Over finite field
Input: P, p, $k = k_x k_{x-1} k_{x-2} \cdots k_1 k_0$
Output: $T = kP$

1. $len \leftarrow$ Length of k
2. $P_t \leftarrow P$
3. $P_r \leftarrow 0$
4. $i \leftarrow 0$
5. While $i <= len-1$ Do
6. IF $k_i == 1$ Then
7. IF $P_r = 0$ Then
8. $P_r = P_t$

9. Else
10. $P_r = P_r + P_t$
11. EndIF
12. EndIF
13. $P_t \leftarrow 2P_t$
14. End While
15. $T \leftarrow P_r$

3. The Proposed Method

Assuming, $d = a - b$, $L(d) = x$, $L(a) = L(b) = x + 1$, where $a, b, x \in ¢^+$, the aim of this paper is to find both of a and b for the new algorithm to speed up modular exponentiation. When $g \in ¢^+$ is expressed as all modular squares necessary for Fast Exponent, the proposed approach always requires $g + 1$. However, the number of iterations required to compute modular multiplication using the proposed method is reduced when the number of groups is small. Each group is generated from number of 0 in the binary system which is adjacent to each other.

From, $m = c^d \bmod n = c^{a-b} \bmod n = c^a * (c^b)^{-1} \bmod n$, assigning $r = c^b \bmod n$, therefore

$$m = c^a * r^{-1} \bmod n \quad (1)$$

Furthermore, if d is represented as the ECC's private key, then both of a and b can be also chosen as the multiplier of the point to find the other point over ECC.

Assuming, $Q = d*P = (a - b)*P$, then

$$Q = a*P - b*P \quad (2)$$

3.1 Finding *a* and *b* for RSA's private key

Assigning, $a = \sum_{i=0}^{k} a_i$ or $a = a_k a_{k-1} a_{k-2} \cdots a_1 a_0$ and $b = \sum_{i=0}^{k} b_i$ or $b = b_k b_{k-1} b_{k-2} \cdots b_1 b_0$ where $a_j = b_j$ when both of them must be equal to 0. Therefore, if $a_j = 1$, then $b_j = 0$. On the other hand, a_j becomes 0 when $b_j = 1$. Assuming, d is an odd number, the process to find both of a and b is divided into two cases.

Case 1: $H(B(d)) = L(d)$

Assuming, $L(d) = x$, a and b can be assigned as $a = 2^{x+1}$ and $b = 1$. If this condition is occurred, the proposed method is the most suitable to finish modular exponentiation, because there is no group of the exponent's binary system that 0 is adjacent to each other.

Example 1: Assuming, $d = 31$, finding a and b

Sol. Because $B(31) = 11111$ and $L(31) = 5$, then $b = 1$, $L(a) = 6$ and $a = 2^6 = 32$.

In addition, **Table 2** shows the result of $d = a - b$ in the binary system

Table 2. The result of $d = a - b$ in binary system from Example 1

Variable	Bit' s Position					
	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
a	1	0	0	0	0	0
b	0	0	0	0	0	1
d = a - b	0	1	1	1	1	1

Table 2 shows that when $(c^a)*(c^{-b}) \bmod n$ is used instead of $c^d \bmod n$, the number of modular multiplications is reduced. In addition, it implies that only one modular multiplication is required in this case, although d is large.

Case 2: $H(B(d)) \neq L(d)$

The process to find a and b is different from the first case. In fact, it is divided into 5 steps.

Step 1: Assigning $a = 2^{x+1} = a_x a_{x-1} a_{x-2} \cdots a_1 a_0$, where $a_x = 1$ and the others are 0

Step 2: Assigning $b = b_x b_{x-1} b_{x-2} \cdots b_1 b_0$, where $b_0 = 1$ and the others are 0

Step 3: Considering only positions of B(d) that a number of 0 is adjacent to each other. In fact, there may be many groups. Assigning $d_{i+2} = 1$, $d_{i+1} = 0$, $d_i = 0$, $d_{i-1} = 0$, $d_{i-2} = 1$ are members of a group to consider step 4 and step 5.

Step 4: Finding the first position and the last position of each group in Step 3.

1) The first position of B(d) in Step 3 which is appeared as 0 is $i - 1$
2) The last position of B(d) in Step 3 which is appeared as 0 is $i + 1$

Step 5: Changing the values of a and b as shown below:

1) $a_{i-1} = 1$, where $i - 1$ is the first position of B(d) that number 0 is found in a group
2) $b_{i+2} = 1$, where $i + 1$ is the last position of B(d) that number 0 is found in a group

Example 2: Assuming $d = 57$, finding a and b

Sol. First, B(57) = 111001, L(57) = 6 and H(57) = 4, each position of B(57) is as follows:

d_6	d_5	d_4	d_3	d_2	d_1	d_0
-	1	1	1	0	0	1

Because H(d) ≠ L(d), then case 2 must be chosen to find a and b as follows:

Step 1: $a = 2^6 = 100000$ that each position of B(a) is as follows:

a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	0	0	0	0	0	0

Step 2: $b = 1$ that each position of B(b) is as follows:

b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	1

Step 3: It is found that $d_3 = 1$, $d_2 = 0$, $d_1 = 0$, $d_0 = 1$

Step 4:

1) The first position of B(57) which is appeared as 0 is 1
2) The last position of B(57) which is appeared as 0 is 2

Step 5:

$a_1 = 1$, $b_3 = 1$

Therefore, **Table 3** shows the result of $d = a - b$ in the binary system

Table 3. The result of $d = a - b$ in the binary system from Example 2

Variable	Bit' s Position						
	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
a	1	0	0	0	0	1	0
b	0	0	0	1	0	0	1
d = a - b	0	1	1	1	0	0	1

If number of groups is very small, then equation (1) should be selected for the implementation to decrease the cost.

Example 3: Assuming $d = 101$, finding a and b

Sol. B(101) = 1100101, L(101) = 7 and H(B(101)) = 4, each position of B(101) is as follows:

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0
-	1	1	0	0	1	0	1

Because H(d) ≠ L(d), then case 2 must be chosen to find a and b as follows:

Step 1: $a = 2^8 = 10000000$ that each position of B(a) is as follows:

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	0	0	0	0	0	0	0

Step 2: $b = 1$, then each position of B(b) is as follows:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	1

Step 3: This example has two groups that a number of 0 is adjacent to each other as follows:

Group 1: $d_2 = 1, d_1 = 0, d_0 = 1$

Group 2: $d_5 = 1, d_4 = 0, d_3 = 0, d_2 = 1$

Step 4: Group 1:

1) The first position in group 1 which is appeared as 0 is 1
2) The last position in group 1 which is appeared as 0 is 1

Group 2:

1) The first position in group 2 which is appeared as 0 is 3
2) The last position in group 2 which is appeared as 0 is 4

Step 5: Group 1: $a_1 = 1, b_2 = 1$, **Group 2:** $a_3 = 1, b_5 = 1$

Therefore, **Table 4** shows the result of $d = a - b$ in the binary system

Table 4. The result of $d = a - b$ in the binary system from Example 3

Variable	Bit' s Position							
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
a	1	0	0	0	1	0	1	0
b	0	0	1	0	0	1	0	1
d = a - b	0	1	1	0	0	1	0	1

3.2 The alternative method to compute modular exponentiation

Assigning both of a and b are found, the steps to prepare the new exponent for the improvement of Fast Exponent are as follows:

Step 1: Changing all positions of b_i which equal to 1 as 2

Step 2: Computing $f = f_x f_{x-1} f_{x-2} \cdots f_1 f_0$, where $f_i = a_i + b_i$

Note: f_i is not a bit in the binary system but it is just one of the following digits: 0, 1 or 2 to be referred in the proposed algorithm.

After c is found, the proposed algorithm can be selected to compute modular exponentiation by using f as the exponent instead of d that the result is not changed.

Algorithm: The Improvement of Fast Exponent

Input: $c, n, f = f_x f_{x-1} f_{x-2} \cdots f_1 f_0$

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

1. $len \leftarrow$ Length of f
2. $m' \leftarrow c$
3. $m_a \leftarrow 1$
4. $m_b \leftarrow m'$
5. $i \leftarrow 1$
6. While $i <= len\text{-}1$ Do
7. $m' \leftarrow (m')^2 \bmod n$
8. IF $f_i == 1$ Then
9. $m_a \leftarrow (m_a * m') \bmod n$
10. Else IF $f_i == 2$ Then
11. $m_b \leftarrow (m_b * m') \bmod n$
12. End IF
13. $i \leftarrow i + 1$
14. End While
15. $r \leftarrow m_b^{-1} \bmod n$
16. $m \leftarrow m_a * r \bmod n$

Because d is always an odd integer, it implies that $f_0 = 2$. Therefore, in step 3 and step 4, m_a and m_b are always begun as 1 and c, respectively. In addition, modular multiplication will be implemented when $f_i = 1$ or 2. The number of steps to compute modular multiplication is the result of $H(B(a)) + H(B(b))$. In fact, time to finish modular exponentiation by using the Improvement of Fast Exponent is low whenever number of groups is small.

Example 4: Computing 7^{2035} mod 137

Sol: Because B(2035) = 11111110011_2, then $a = 100000000100_2$, $b = 000000010001_2$ and $f = 100000020102$, each position of $f = f_8f_7f_6f_5f_4f_3f_2f_1f_0$ is as follows:

f_{11}	f_{10}	f_9	f_8	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_0
1	0	0	0	0	0	0	2	0	1	0	2

From, the Improvement of Fast Exponent,

Step 1-5: $len = 12$, $m' = 7$, $m_a = 1$, $m_b = 7$, $i = 1$

Step 6 -14, Loops

Loop 1: ($i = 1 <= 11$)

Step 7: $m' = 7^2$ mod 137 = 49, → **modular square**

Because $f_1 = 0$, continuing to Step 13, $i = 2$

Loop 2: ($i = 2 <= 11$)

Step 7: $m' = 49^2$ mod 137 = 72, → **modular square**

Because $f_2 = 1$, Step 9 is required

Step 9: $m_a = (1 * 72)$ mod 137 = 72 → **modular multiplication**

Loop 3: ($i = 3 <= 11$)

Step 7: $m' = 72^2$ mod 137 = 115, → **modular square**

Because $f_3 = 0$, continuing to Step 13, $i = 4$

Loop 4: ($i = 4 <= 11$)

Step 7: $m' = 115^2$ mod 137 = 73, → **modular square**

Because $f_4 = 2$, Step 11 is required

Step 11: $m_b = (7 * 73)$ mod 137 = 100 → **modular multiplication**

Loop 5: ($i = 5 <= 11$)

Step 7: $m' = 73^2$ mod 137 = 123, → **modular square**

Because $f_5 = 0$, continuing to Step 13, $i = 6$

Loop 6: ($i = 6 <= 11$)

Step 7: $m' = 123^2$ mod 137 = 59, → **modular square**

Because $f_6 = 0$, continuing to Step 13, $i = 7$

Loop 7: ($i = 7 <= 11$)

Step 7: $m' = 59^2$ mod 137 = 56, → **modular square**

Because $f_7 = 0$, continuing to Step 13, $i = 8$

Loop 8: ($i = 8 <= 11$)

Step 7: $m' = 56^2$ mod 137 = 122, → **modular square**

Because $f_8 = 0$, continuing to Step 13, $i = 9$

Loop 9: ($i = 9 <= 11$)

Step 7: $m' = 122^2$ mod 137 = 88, → **modular square**

Because $f_9 = 0$, continuing to Step 13, $i = 10$

Loop 10: ($i = 10 <= 11$)

Step 7: $m' = 88^2$ mod 137 = 72, → **modular square**

Because $f_{10} = 0$, continuing to Step 13, $i = 11$

Loop 11: ($i = 11 <= 11$)

Step 7: $m' = 72^2$ mod 137 = 115, → **modular square**

Because $f_{11} = 1$, Step 9 is required

Step 9: $m_a = (72 * 115)$ mod 137 = 60 → **modular multiplication**

Because $i = 11$, jumping out of the loop.

Step 15: $r = 100^{-1}$ mod 137 = 37 → **modular inverse**

Step 16: $m = 60 * 37$ mod 137 = **28** → **modular multiplication**

Therefore, $7^{2035} \bmod 137 = 28$. In fact, in this example, the proposed method requires 11 modular squares, 1 modular inverse and 4 modular multiplications, total costs are 11+4+1 = 16. However, if Fast Exponent is chosen, it requires 10 modular squares and 8 modular multiplications, total costs are 10+8 = 18. As a result, it implies that the total costs to finish modular exponentiation by using the proposed method is less than Fast Exponent.

In general, number of modular multiplications by using the proposed method is based on number of groups that 0 is adjacent to each other. It is different from Fast Exponent that number of modular multiplications is depended on Hamming Weight of the exponent. As a result, when the number of groups is small, the proposed method can complete the procedure rapidly. Furthermore, as compared to combining CRT and RSA, this method can be applied with CRT to reduce the time required to complete the decryption process.

Consider the equation to find m_p: Assuming, $d_p = a_p - b_p$, where a_p and $b_p \in ¢$, then

From, $m_p = c^{d_p} \bmod p = c^{(a_p - b_p)} \bmod p = c^{a_p} * (c^{b_p})^{-1} \bmod p$

Assigning, $r_p = c^{b_p} \bmod p$, then

$$m_p = c^{a_p} * r_p^{-1} \bmod p \tag{3}$$

Consider the equation to find m_q: Assuming, $d_q = a_q - b_q$, where a_q and $b_q \in ¢$, then

From, $m_q = c^{d_q} \bmod q = c^{(a_q - b_q)} \bmod q = c^{a_q} * (c^{b_q})^{-1} \bmod q$

Assigning, $r_q = c^{b_q} \bmod q$, then

$$m_q = c^{a_q} * r_q^{-1} \bmod q \tag{4}$$

Assuming, $f_p = a_p + b_p$, $f_q = a_q + b_q$, n_p is number of groups in f_p and n_q is number of groups in f_q where $H(d_p)$ and $H(d_q)$ are high, there are four cases to select the best parameters to finish both of m_p and m_q as follows:

Case 1: n_p and n_q are small

$m_p = c^{a_p} * r_p^{-1} \bmod p$ and $m_q = c^{a_q} * r_q^{-1} \bmod q$ → **The Proposed Method**

Case 2: n_p is high, n_q is small

The best algorithm to compute m_p should be decided after all parameters are analyzed

$m_q = c^{a_q} * r_q^{-1} \bmod q$ → **The Proposed Method**

Case 3: n_p is small, n_q is high

$m_p = c^{a_p} * r_p^{-1} \bmod p$ → **The Proposed Method**

The best algorithm to compute m_q should be decided after all parameters are analyzed

Case 4: n_p and n_q are high

The best algorithm to compute m_p and m_q should be decided after all parameters are analyzed deeply. Based on the preceding cases, it follows that the proposed method is selected to compute the equation in cases 1, 2, and 3 for the quickest process.

3.3 The alternative method to point multiplication

The concept of the proposed method in section 3.1 and section 3.2 can be also applied with ECC over finite field to speed up point multiplication. Assuming, d is the multiplier of the point, P, and $d = a - b$, then $T = d*P = (a - b)*P$. However, for ECC, d can be either an odd number or an even number. Therefore, the process to find both of a and b for an even value of the multiplier must be also considered. Based on the number of groups, the multiplier is

divided into two cases.

Case 1: One group that a number of 0 is adjacent to each other and the position is from the least significant bit to the l position, where l is the last position that bit's value is equal to 0

In fact, a and b can be assigned as $a = 2^{x+1}$ and $b = 2^{l+1}$.

Example 5: Assuming $d = 60$, finding a and b

Sol. Because B(60) = 111100, L(60) = 6, $l = 1$, $b = 2^2 = 4 = 100_2$, L(a) = 6 and $a = 2^6$.

In addition, Table 5 shows the result of $d = a - b$ in the binary system

Table 5. The result of $d = a - b$ in the binary system from Example 5

Variable		Bit' s Position					
	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
a	1	0	0	0	0	0	0
b	0	0	0	0	1	0	0
d = a - b	0	1	1	1	1	0	0

The information in **Table 5** shows that H(B(b)) + H(B(a)) < H(B(d)), then it implies that number of point additions is decreased.

Case 2: Multi groups that a number of 0 is adjacent to each. In this case, the process to find pattern in the first group is similar to case 1. However, the process to find the patterns in the other groups are similar to case 2 in section 3.1.

Example 6: Assuming $d = 1662$, finding a and b

Sol. B(1662) = 11001111110, L(1662) = 11 and H(B(1662)) = 8, then

d_{10}	d_9	d_8	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0
1	1	0	0	1	1	1	1	1	1	0

group 2 **group 1**

Because, there are two groups of number 0 that is adjacent to each other, then

Considering group 1:

First, assigning $a = 2^{12} = 100000000000$ that each position of B(a) is as follows:

a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	0	0	0	0	0	0	0	0	0	0	0

Because, the 2nd position is the first position that bit's value equal to 1, therefore $b_1 = 1$,

b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	0	0	1	0

Considering group 2:

In group 2, it is found that $d_9 = 1$, $d_8 = 0$, $d_7 = 0$, $d_6 = 1$, then

1) The first position of B(1662) which is appeared as 0 in the second group is 7
2) The last position of B(1662) which is appeared as 0 in the second group is 8

Therefore,

$a_7 = 1$, $b_9 = 1$

Table 6 shows the result of $d = a - b$ in the binary system

Table 6. The result of $d = a - b$ in binary system from Example 6

Variable	Bit' s Position											
	11	10	9	8	7	6	5	4	3	2	1	0
a	1	0	0	0	1	0	0	0	0	0	0	0
b	0	0	1	0	0	0	0	0	0	0	1	0
d = a - b	0	1	1	0	0	1	1	1	1	1	1	0

In addition, after both of a and b are found, $f = f_x f_{x-1} f_{x-2} \cdots f_1 f_0$, where $f_i = a_i + b_i$ and $f_i = 2$ when $b_i = 1$, must be also computed for the proposed algorithm, known as the Improvement of Point multiplication Over finite field. In example 6, f = 102010000020. Assuming f is found, the Improvement of Point multiplication Over finite field is as follows:

Algorithm: The Improvement of Point multiplication Over Finite Field
Input: P, p, $f_x f_{x-1} f_{x-2} \cdots f_1 f_0$ which are the representative of d
Output: $Q = d*P$

```
1)  len ← Length of f
2)  Pt ← P
3)  Pr ←0
4)  Ps ← 0
5)  i ← 0
6)  While i <= len-1 Do
7)     IF fi == 1 Then
8)        IF Pr = 0 Then
9)           Pr ←  Pt
10)       Else
11)          Pr ← Pr + Pt
12)       EndIF
13)    Else IF fi == 2 Then
14)       IF Ps = 0 Then
15)          Ps ←  Pt
16)       Else
17)          Ps ←  Ps + Pt
18)       EndIF
19)    EndIF
20)    Pt ← 2Pt
21)    i ← i + 1
22) End While
23) Q ← Pr + (-Ps)
```

Example 7: Assuming $y^2 \equiv x^3 + 419x + 21351 \bmod 24359$ and P = (1217, 331), finding Q = 1662*P by using The Improvement of Point multiplication Over Finite Field
Sol. In example 6, f = 102010000020, then Q = 1662*P can be computed by using The Improvement of Point multiplication Over Finite Field as follows:

Step 1-5: len = 12, P_t = (1217, 331), P_r = 0, P_s = 0, i = 0
Steps 6 -21, Loops
Loop 1: (i = 0 <= 12)
Because f_0 = 0, then there is no required step which is in condition.
Step 20: $P_t = 2P_t$ = (20589, 11022) → **Point Doubling**
Loop 2: (i = 1 <= 12)
Because f_1 = 2, then steps 13 – 19 are required
Because P_s = 0, then Steps 14 – 15 are required
Step 15: $P_s = P_t$ = (20589, 11022)
Step 20: $P_t = 2P_t$ = (3638, 18887) → **Point Doubling**
Loop 3: (i = 2 <= 12)
Because f_2 = 0, then there is no required step which is in the condition.
Step 20: $P_t = 2P_t$ = (18533, 2525) → **Point Doubling**
Loop 4: (i = 3 <= 12)
Because f_3 = 0, then there is no step which is required in the condition.
Step 20: $P_t = 2P_t$ = (22671, 20089) → **Point Doubling**
Loop 5: (i = 4 <= 12)

Because $f_4 = 0$, then there is no step which is required in the condition.
Step 20: $P_t = 2P_t = (16158, 22384)$ → **Point Doubling**

Loop 6: ($i = 5 <= 12$)
Because $f_5 = 0$, then there is no step which is required in the condition.
Step 20: $P_t = 2P_t = (10647, 1447)$ → **Point Doubling**

Loop 7: ($i = 6 <= 12$)
Because $f_6 = 0$, then there is no step which is required in the condition.
Step 20: $P_t = 2P_t = (4829, 22146)$ → **Point Doubling**

Loop 8: ($i = 7 <= 12$)
Because $f_7 = 1$, then steps 7 – 12 are required
Because $P_r = 0$, then Steps 8 – 9 are required
Step 8: $P_r = P_t = (4829, 22146)$
Step 20: $P_t = 2P_t = (19682, 16531)$ → **Point Doubling**

Loop 9: ($i = 8 <= 12$)
Because $f_8 = 0$, then there is no step which is required in the condition.
Step 20: $P_t = 2P_t = (8041, 16914)$ → **Point Doubling**

Loop 10: ($i = 9 <= 12$)
Because $f_9 = 2$, then steps 13 – 19 are required
Because $P_s \neq 0$, then Steps 16 – 18 are required
Step 17: $P_s = P_s + P_t = (17983, 22767)$ → **Point Addition**
Step 20: $P_t = 2P_t = (16002, 21223)$ → **Point Doubling**

Loop 11: ($i = 10 <= 12$)
Because $f_{10} = 0$, then there is no step which is required in the condition.
Step 20: $P_t = 2P_t = (17585, 12707)$ → **Point Doubling**

Loop 12: ($i = 11 <= 12$)
Because $f_{11} = 1$, then steps 7 – 12 are required
Because $P_r \neq 0$, then Steps 10 – 12 are required
Step 8: $P_r = P_r + P_t = (17230, 12579)$ → **Point Addition**
Step 20: $P_t = 2P_t = (21206, 993)$ → **Point Doubling**

End of Loop
Step 23: $Q = P_r + (-P_s) = (6797, 4186)$ → **Point Addition**

Therefore, $Q = 1662*P = (6797, 4186)$. In fact, in this example, the proposed method requires 12-point doublings and 3-point additions, total costs are 12 + 3 = 15. However, if point multiplication algorithm is chosen, it requires 11-point doublings and 7-point additions, total costs are 11 + 7 = 18. Therefore, in this example, it implies that the total costs to finish point multiplication by using the proposed method is less than point multiplication algorithm.

4. Experimental Results

The experimental results will be displayed and discussed in this section. The experiment is divided into two sections. Part 1 is a cost comparison of the proposed method and Fast Exponent for computing modular exponentiation. Part 2 will show and discuss the cost comparison of computing point doubling and point addition to find the new point over the curve.

4.1 Experimental Results from Modular Exponentiation Computing

In this part, the comparison about the costs to compute $c = a^b \bmod n$ between Fast Exponent and the proposed method is discussed. There are three different costs for the consideration,

modular square, modular multiplication and modular inverse. Bit's length of all values of n and b in this section are assigned as 1024 and 900, respectively. In addition, 10 – 15 cases which are considered from number of groups are selected to analyze the performance.

Table 7. Modular Square and Modular Multiplication for 1024 bits of Modulus, 900 bits of the exponent and number of bits which is equal to 0 are 10

Algorithm	Number of Groups (X)	Modular Square	Modular Multiplication	Modular Inverse
Fast Exponent	-	899	889	-
The Proposed Method	1	**900**	**4**	**1**
	2		6	
	3		8	
	4		10	
	5		12	
	6		14	
	7		16	
	8		18	
	9		20	
	10		22	

H(b) in **Table 7** is 890. However, the distribution of each bit that equals to 0 is divided into 10 cases. Each case is determined by the number of groups in which 0 is adjacent to each other. The experimental results show that costs of Fast Exponent are stable, 899 modular squares and 889 modular multiplications. The reason is that Fast Exponent is depended on Hamming Weight and H(B(b)) is stable, H(B(b)) = 890. However, the proposed method is not depended on the Hamming Weight. It is based on only number of groups in which 0 is adjacent to each other. The information shows that the number of modular squares is not changed, but the number of modular multiplications is based on the number of groups. In fact, the proposed method can finish the process rapidly when number of groups is small. In this table, if there is only one group, the number of modular multiplications is only four. Furthermore, 22 modular multiplications are the maximum costs, because number of bits which are equal to 0 are always 10 for all exponent in this experiment. On the other hand, 899 modular multiplications are required for Fast Exponent. Although modular inverse is required for the proposed method, it is only used once to find the result.

Table 8. Modular Square and Modular Multiplication for 1024 bits of Modulus, 900 bits of the exponent and number of bits which equal to 0 are 100

Algorithm	Number of Groups (X)	Modular Square	Modular Multiplication	Modular Inverse
Fast Exponent	-	899	799	-
The Proposed Method	1	**900**	**4**	**1**
	2		6	
	3		8	
	4		10	
	5		12	
	6		14	
	7		16	
	8		18	
	9		20	
	10		22	
	100		202	

Table 8 differs from **Table 7** in terms of the number of bits equal to 0. In fact, there are 100 bits in **Table 8**. Because Hamming Weight is lower than the information in **Table 7**, the experimental results show that the number of modular multiplications in Fast Exponent is reduced. However, for each case of the proposed technique, the number of modular multiplications is stable. Nonetheless, the worst case from the proposed methods is that all neighbors of a bit equal to 0 are 1. In fact, there are 202 modular multiplications in this case. However, they are still cheaper than the costs in Fast Exponent.

In **Table 9**, the number of bits in the exponent that are equal to 0 is 400, and the worst case of modular multiplications for the proposed method is 802. The costs are higher than the same process in Fast Exponent, only 499 modular multiplications are required. However, there are the most of groups which are 248 in the proposed method, despite the fact that the number of modular multiplications is still less than 499. As a result, it is very likely that the proposed method will be chosen as the quickest method to complete the modular exponentiation process.

Table 9. Modular Square and Modular Multiplication for 1024 bits of Modulus, 900 bits of the exponent and number of bits which equal to 0 are 400

Algorithm	Number of Groups (*X*)	Modular Square	Modular Multiplication	Modular Inverse
Fast Exponent	-	899	499	-
The Proposed Method	1	**900**	**4**	**1**
	2		6	
	3		8	
	4		10	
	5		12	
	6		14	
	7		16	
	8		18	
	9		20	
	10		22	
	100		202	
	200		402	
	248		498	
	249		500	
	400		802	

4.2 Experimental Results from Point Multiplication Computing

This section compares the costs of computing $Q = d*P$ over a finite field $¢_p$ using the Point Multiplication Algorithm and the proposed method. There are two costs to consider: point doubling and point addition. All values of modulus (p) and d in this section have bit lengths of 160 and 150, respectively. Furthermore, 10 – 15 cases from a number of groups are chosen for performance analysis. Because d can be either an odd number or an even number, both types are selected for the implementation.

H(d) in **Table 10** is 140, then number of bits with a value of 0 is 10. The distribution of each bit that equals to 0 is divided into ten cases. The experimental results show that the Point Multiplication Algorithm's costs are constant, with 150-point doublings and 139-point additions. In fact, H(d) is always stable and number of point additions for Point Multiplication Algorithm is based on H(d). On the other hand, the proposed method which is not depended on the Hamming Weight is based on only number of groups that 0 is adjacent to each other. The information shows that number of point doublings is stable but number of point additions is based on number of groups. In general, the proposed method can finish the process rapidly when number of groups is small. If there is one group, number of point additions is always

one when *d* is an even number but they are three when *d* is an odd number. Furthermore, the proposed method's maximum costs are 21-point additions, because the number of bits equal to 0 is always 10. On the other hand, 139-point additions are required for Point Multiplication Algorithm.

Table 10. Point doubling and Point Addition for 160 bits of the modulus, 150 bits of the multiplier and number of bits which are equal to 0 are 10

Algorithm	**Number of Groups (*X*)**	**Point Doubling**	**Point Addition**	
			***d* is an even number**	***d* is an odd number**
Point Multiplication Algorithm	-	150	139	139
The Proposed Method	1	**151**	**1**	**3**
	2		3	5
	3		5	7
	4		7	9
	5		9	11
	6		11	13
	7		13	15
	8		15	17
	9		17	19
	10		19	21

Table 11. Point doubling and Point Addition for 160 bits of the modulus, 150 bits of the multiplier and number of bits which equal to 0 are 50

Algorithm	**Number of Groups (*X*)**	**Point Doubling**	**Point Addition**	
			d is an even number	**d is an odd number**
Point Multiplication Algorithm	-	150	99	99
The Proposed Method	1	**151**	**1**	**3**
	2		3	5
	3		5	7
	4		7	9
	5		9	11
	6		11	13
	7		13	15
	8		15	17
	9		17	19
	10		19	21
	49		97	99
	50		99	101

Table 11 differs from **Table 10** in terms of the number of bits equal to 0. In fact, there are 50 bits in this table. Because Hamming Weight is lower than the information in **Table 10**, the experimental results show that the number of point additions in the Point Multiplication Algorithm is reduced. However, number of point additions for each case of the proposed method is stable. Furthermore, the worst case from the proposed methods is that all neighbors of a bit equal to 0 are 1. **Table 11** shows what happens when there are 50 groups, the number of point additions is 99 for an even multiplier and it is 101 for an odd multiplier. Furthermore, 50 groups are only the condition that the costs of the proposed method are greater than the costs of the Point Multiplication Algorithm. As a result, if the bit length of *d* is 150 bits and H(*d*) = 100, the proposed method has 50 cases. If the condition of *d* falls between cases 1 and 49, the proposed method should be used for implementation.

In **Table 12**, number of bits which are equal to 0 is 70, the results show that the costs to compute both of point addition and point doubling are similar to the results in **Table 11** when 1 – 50 groups are considered. That is, the proposed method should be chosen to complete the process when the number of groups is between 1 and 24, because number of point additions by using Point Multiplication Algorithm is reduced to 49. However, when the number of groups exceeds to 25, the method becomes inefficient. The maximum groups in **Table 12** are 70. It demonstrates that, while number of point doublings remain stable, number of point additions are the most expensive cost.

As a result, when the number of groups is small, the proposed method is very efficient. Assuming length of d is k bits, number of processes to compute point doubling is always $k + 1$. However, point addition is based on number of groups. In fact, two-point addition processes are required for each group. It implies that the proposed method is faster than Point Multiplication Algorithm when number of groups is small. On the other hand, it becomes an inefficient method when number of groups is large. However, the efficiency of point multiplication algorithm is based on Hamming Weight of the multiplier. From **Table 10 - 12**, the best performance of this algorithm is in **Table 12**, because Hamming Weight is the smallest when it is compared with the multipliers in **Table 10** and **Table 11**.

Table 12. Point doubling and Point Addition for 160 bits of the modulus, 150 bits of the multiplier and number of bits which equal to 0 are 70

Algorithm	**Number of Groups (*X*)**	**Point Doubling**	**Point Addition**	
			***d* is an even number**	***d* is an odd number**
Point multiplication algorithm	-	150	49	49
The Proposed Method	1	**151**	**1**	**3**
	2		3	5
	3		5	7
	4		7	9
	5		9	11
	6		11	13
	7		13	15
	8		15	17
	9		17	19
	10		19	21
	24		47	49
	25		49	51
	50		99	101
	70		139	141

4.3 Experimental Results about Computation Time

This section mentions time to finish the process. To control the same settings, all experiments were performed on a 2.53 GHz Intel® Core i5 with 8 GB memory. BigInteger class in Java is chosen to manage the variables that are large. The experiment is split into two sections. The first section compares modular multiplication with completion. The other is a comparison to finish point multiplication. Because the proposed method performs well when number of groups is small, one group and two groups are selected for the experiment to assure this mention.

In **Fig. 1** shows a comparison between Fast Exponent and the proposed method. Two cases are chosen to examine the outcome of the proposed method. When there is only one group, the proposed method (1) produces the result, and when there are two groups, the proposed method (2) produces the result. The bit lengths of the modulus and the exponent are 1024 and 1000,

respectively. However, there are seven cases of Hamming Weight of the exponent, which are 8, 98, 198, 398, 598, 798 and 998. The experimental results show that both of the proposed method (1) and (2) are faster than Fast Exponent. Furthermore, it is quite stable, with a time span of 18.46 to 20.66 msec. When Fast Exponent is used, the time to complete the process ranges from 21.16 to 52.03 msec. In **Fig. 2** shows a comparison between Point Multiplication and the proposed method. Two cases are chosen to examine the outcome of the proposed method. When there is only one group, the proposed method (3) produces the result, and when there are two groups, the proposed method (4) produces the result. The bit lengths of the modulus and the multiplier are 160 and 150, respectively. However, there are seven cases of Hamming Weight of the multiplier, which are 8, 18, 38, 58, 88, 118 and 148. The experimental results show that both of the proposed method (3) and (4) are faster than Point Multiplication. Furthermore, it is quite stable, with a time span of 6.29 to 7.22 msec. When Point Multiplication is used, the time to complete the process ranges from 17.41 to 24.97 msec.

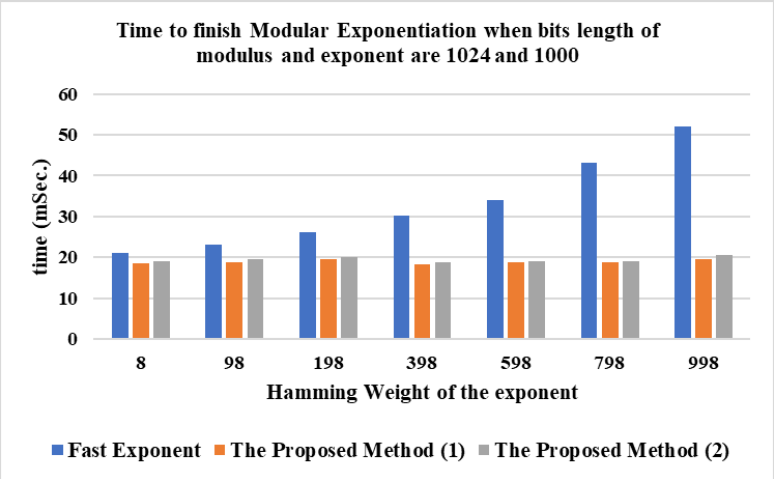

Fig. 1. Time to finish Modular Exponentiation

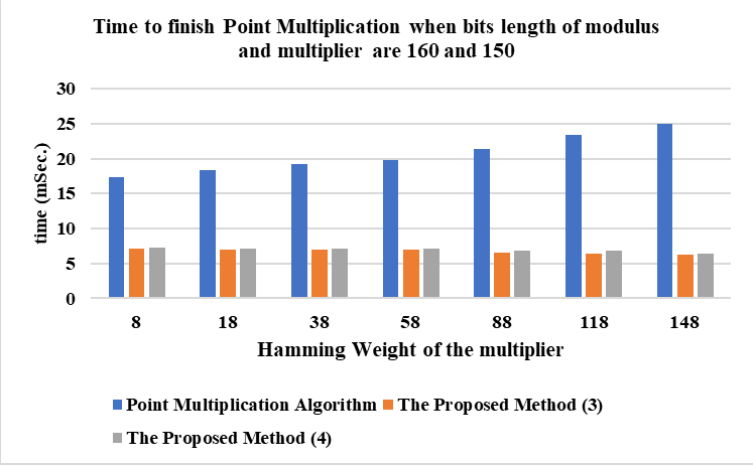

Fig. 2. Time to finish Point Multiplication

Furthermore, the information in **Fig. 1** and **Fig. 2** demonstrates that the proposed method

is the most efficient way to complete the task when the condition is correctly responding to this procedure. Furthermore, the information in sections 4.1 and 4.2 shows that Hamming Weight is unaffected to the proposed method. It is based solely on the number of groups and length. On the other hand, Fast Exponent and Point Multiplication are based on Hamming Weight and length of n. However, when security is taken into account, one group may be vulnerable. Assigning, d is represented as the exponent or the multiplier and there is only one group, Maximum loop to discover d is L(d) and it is the worst case. On the other hand, if at least two groups are selected, there are numerous patterns to generate d. Assuming d's bit length is 10, all of these values are examples of d that has two groups: 1010111111, 1001110011, 1101011111, 1111000010, 1010001111, 1100100011 etc. Therefore, it is very difficult to discover d when there are at least two groups. In addition, d is provided a value greater than 10 bits in a real world. This means that the pattern's range has been broadened. However, to avoid easily attacks, such as brute force attack, number of bits that is equal to 0 should not be assigned too large.

Next, total loops to finish process are considered. For RSA, assigning M is number of modular multiplication and I is represented as one modular inverse. Number of modular squares is always equal to L(d). However, M is based on number of groups. Therefore, total loops to finish task are:

$$t_R = \mathrm{L}(d) + \mathrm{M} + \mathrm{I} \tag{5}$$

where t_R is total loops to finish modular exponentiation

In fact, total loops for Fast Exponent are about $\mathrm{L}(d) - 1 + \mathrm{M}_f$, where M_f is number of modular multiplications. In general, it is high possible that M_f is very bigger than M when number of groups is small.

In addition, assigning, A is number of point addition for ECC, number of point doubling is always equal to L(d)+1. However, A is based on number of groups. Therefore, total loops to finish task are:

$$t_E = \mathrm{L}(d) + 1 + \mathrm{A} \tag{6}$$

where t_E is total loops to finish point multiplication

In fact, total loops for Point Multiplication are about $\mathrm{L}(d) + \mathrm{A}_p$, where A_p is number of point additions. In general, it is high possible that A_p is very bigger than A when number of groups is small.

5. Conclusion

In this paper, the new special method is presented. Assuming that the number of groups is generated from number of 0 in the binary system that is adjacent to each other, the following is the condition under which the proposed method should be chosen; For modular exponentiation, number of groups of the exponent's binary value is very small, because there are only a few modular multiplications. Nevertheless, number of groups of the multiplier's binary value is very small for point multiplication because there are only a few point additions. The main process of the proposed method is to select two integers whose difference equals to the exponent or the multiplier instead of one of these values. When the number of groups is very small, the experimental results for modular exponentiation computation show that the proposed method is the fastest algorithm, because the number of modular multiplications is small. In addition, the proposed method is also the best algorithm to finish point multiplication process when number of groups is small.

References

[1] W. Diffie and M.E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, pp. 644 – 654, 1976. Article (CrossRef Link) .

[2] 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. Article (CrossRef Link) .

[3] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, pp. 469 – 472, 1985. Article (CrossRef Link) .

[4] E. Bach and B. Sandlund, "Baby-step giant-step algorithms for the symmetric group," *Journal of Symbolic Computation*, vol. 85, pp. 55 – 71, 2018. Article (CrossRef Link) .

[5] E. Teske, "The Pohlig–Hellman Method Generalized for Group Structure Computation," *Journal of Symbolic Computation*, vol. 27(6), pp. 521 – 534, 1999. Article (CrossRef Link) .

[6] R.Padmavathy and C. Bhagvati, "Discrete logarithm problem using index calculus method," *Mathematical and Computer Modelling*, vol. 55(1-2), pp.161-169, 2012. Article (CrossRef Link) .

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

[8] D. Boneh, and G. Durfee, "Cryptanalysis of RSA with Private Key d less than $N^{0.292}$," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1339-1349, 2000. Article (CrossRef Link) .

[9] 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*, vol. 2014, pp. 1 - 9, 2014. Article (CrossRef Link) .

[10] 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. Article (CrossRef Link)

[11] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203 - 209, 1987. Article (CrossRef Link) .

[12] V.S. Miller, "Uses of elliptic curves in cryptography," *Lecture Notes in Computer Science*, vol. 218, pp. 417 - 426, 1986. Article (CrossRef Link) .

[13] Q. Zhou, C. Tian, H. Zhang, J. Yu and 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) .

[14] I. Hazmi, F. Gebali and A. Ibrahim, "High Speed and Low Area Complexity Extended Euclidean Inversion Over Binary Fields," *IEEE Transactions on Consumer Electronics*, vol. 65(3), pp. 408 - 417, 2019. Article (CrossRef Link) .

[15] S.J. Horng, S.F. Tzeng, P. Fan, X. Wang, T. Li and M.K. Khan, "Secure Convertible Undeniable Signature Scheme Using Extended Euclidean Algorithm without Random Oracles," *KSII Transactions on Internet and Information Systems*, vol. 7(6), pp. 1512 - 1532, 2013. Article (CrossRef Link) .

[16] S.G.R. Ekodeck and R. Ndoundam, "PDF steganography based on Chinese Remainder Theorem," *Journal of Information Security and Applications*, vol. 29, pp. 1 - 15, 2016. Article (CrossRef Link)

[17] R. Zhong and J. Ma, "An algorithm for the chinese remainder problem," *International Journal of Computer Mathematics*, vol. 64(3-4), pp. 225 - 233, 1997. Article (CrossRef Link) .

[18] S. Qina, Z. Tan, B. Zhang and F. Zhou, "Distributed secret sharing scheme based on the high-dimensional rotation paraboloid," *Journal of Information Security and Applications*, vol. 58, pp. 1 - 10, 2021. Article (CrossRef Link) .

[19] K. Somsuk, "The improving decryption process of RSA by choosing new private key," in *Proc. of International Conference on Information Technology and Electrical Engineering*, pp. 1 – 4, August 5 – 6, 2016. Article (CrossRef Link) .

[20] 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. 45 – 48, November 15 - 18, 2017. Article (CrossRef Link) .

[21] K. Somsuk, T. Chiawchanwattana and C. Sanemueang, "Speed up RSA's Decryption Process with Large sub Exponents using Improved CRT," in *Proc. of International Conference on Information Technology*, pp. 1 – 5, October 24 – 26, 2018. Article (CrossRef Link) .

[22] T.Q. Ban, T.T.T. Nguyen, V.T. Long, P.D. Dung and B.T. Tung, "A Benchmarking of the Effectiveness of Modular Exponentiation Algorithms using the library GMP in C language," in *Proc. of International Conference on Computational Intelligence*, pp. 237 – 241, October 8 – 9, 2020. Article (CrossRef Link) .
[23] H. Akira, "A New Fast Modular Multiplication Method and Its Application to Modular Exponentiation-Based Cryptography," *Electronics and Communications in Japan Part 3*, vol. 83(12), pp. 88 - 93, 2000. Article (CrossRef Link) .
[24] H. Li, "Pseudo-random scalar multiplication based on group isomorphism," *Journal of Information Security and Applications*, vol. 53, pp. 1 - 14, 2020. Article (CrossRef Link) .
[25] L. Cao and W. Ge, "Analysis of Certificateless Signcryption Schemes and Construction of a Secure and Efficient Pairing-free one based on ECC," *KSII Transactions on Internet and Information Systems*, vol. 12(9), pp. 4527 - 4547, 2018. Article (CrossRef Link) .
[26] J. Zhang, J. Ma, X. Li and W. Wang, "A Secure and Efficient Remote User Authentication Scheme for Multi-server Environments Using ECC," *KSII Transactions on Internet and Information Systems*, vol. 8(8), pp. 2930 - 2947, 2014. Article (CrossRef Link) .

Kritsanapong Somsuk is an associate professor at the Department of Computer and Communication Engineering, Faculty of Technology, Udon Thani Rajabhat University, Udon Thani, Thailand. He obtained his M.Eng. (Computer Engineering) from Department of Computer Engineering, Faculty of Engineering, Khon Kaen University, M.Sc. (Computer Science) from Department of Computer Science, Faculty of Science, Khon Kaen University and his Ph.D. (Computer Engineering) from Department of Computer Engineering, Faculty of Engineering, Khon Kaen University. The area of research interests includes computer security, cryptography and integer factorization algorithms.