An Improvement of Fermat's Factorization by Considering the Last m Digits of Modulus to Decrease Computation Time

Kritsanapong Somsuk, Kitt Tientanopajai

(Corresponding author: Kritsanapong Somsuk)

Department of Computer Engineering, Faculty of Engineering, Khon Kaen University
Khon Kaen 40002, Thailand
(Email: tonpor007@hotmail.com)

(Received Dec. 9, 2015; revised and accepted Jan. 23, 2016)

Abstract

Fermat's Factorization Algorithm (FFA) and the algorithms improved from FFA are the fast integer factorization algorithms when these algorithms are chosen to find two large prime factors of the balanced modulus. The key is a process to find two perfect squares such that their difference is equal to the modulus. However, it is time-consuming to find these two integers because there is only one solution but many integers are chosen in this experiment to find the solution. In this paper, a new improvement of FFA is proposed by leaving out some unrelated integers, which do not affect getting the correct solution. Leaving out these integers results from analyzing the last m digits of the modulus where m is a positive integer. The new faster and improved algorithm is called Specific Fermat's Factorization Algorithm Considered from X (SFFA-X) where X is represented as the last m digits of the modulus. The experimental results showed that SFFA-X can factor the modulus faster than FFA and many modified algorithms of FFA especially when at least 2 digits of X are chosen for the implementation.

Keywords: Fermat's factorization algorithm, integer factorization, RSA

1 Introduction

Integer Factorization is one of the famous techniques for breaking RSA [10] which is the most well-known public key cryptosystem. In general, if the modulus is factored as prime numbers, the private key kept secret will be recovered and then RSA is broken [7]. At present, many integer factorization algorithms were proposed such as [1, 2, 3, 4, 6, 8, 9, 11, 15, 18, 19]. However, the speed of each factorization algorithm for factoring the modulus depends on the size of the modulus and the size of prime factors of the modulus. Nevertheless, Fermat's Factorization Algorithm (FFA) [1, 18] discovered by Pierre de Fermat, is the efficient factorization algorithm whenever it is used to factor the balanced modulus that the difference between two large prime factors is very small [18]. In addition, to find the two large prime factors of the modulus, FFA will rewrite the modulus as the difference of the perfect squares. Although, many factorization algorithms improved from FFA [6, 13, 14, 16, 17, 18] were introduced, they are still time-consuming to factor the modulus.

Assume n is represented as the modulus which is equal to the product of two prime numbers and all prime numbers can be chosen to be a prime factor of n except 2 and 5. In this paper, an efficient technique to speed up FFA is proposed. This technique will consider the last m digits of n before choosing one of the new proposed specific algorithms for the implementation. The advantage of the specific algorithms is the removing unrelated iterations of the computation. In addition, the specific algorithms for the last $m + 1$ digits of n can factor n faster than the specific algorithms for the last m digits of n because the details of n are better known and more iterations of the computation are left.

In fact, the numbers of the specific algorithms for the last m digits of n are based on values of m. The numbers of these algorithms are $4 * 10^{m-1}$. For example, if m is equal to 1, there are 4 specific algorithms as follows: the algorithm for the last digit of n is 1, 3, 7 or 9. Another example assumes m is equal to 2 then there are 40 specific algorithms as follows: the algorithm for the last 2 digits of n is 01, 03, 07, 09, 11, $\cdots$, 97 or 99. The new fast and proposed algorithm is called Specific Fermat's Factorization Algorithm Considered from X (SFFA-X), where X is represented as the last m digits of n. For example, the specific algorithm for the last 2 digits of $n = 1287901$ is called SFFA-01, the specific algorithm for the last 2 digits of $n = 737$ is called SFFA-37 and the specific algorithm for last 3 digits of $n = 136313$ is called SFFA-313.

2 Priliminary

2.1 Fermat's Factorization Algorithm: FFA

FFA is one of some integer factorization algorithms which can factor all composite integers. The speed of FFA depends on the difference between two large prime factors of n: $n = p * q$, where p and q are prime numbers and p is larger than q, and the size of n. However, this algorithm is appropriate with the small result of the subtraction between p and q. The key of FFA is that n is rewritten as the difference of perfect squares as follows:

$$n = (\frac{p+q}{2})^2 - (\frac{p-q}{2})^2 \tag{1}$$

In fact, FFA can be distinguished as 2 different algorithms as follows. For the first algorithm, is called FFA-1, assign $x = \frac{p+q}{2}$ and $y = \frac{p-q}{2}$ then,

$$n = x^2 - y^2. \tag{2}$$

However, assign the initial value of x is equal to $\lceil\sqrt{n}\rceil$, then the process to factor n by using FFA-1 is to find the integer of y, $y = \sqrt{x^2 - n}$. If the integer of y is found, the values of p and q can be computed from $p = x + y$ and $q = x - y$. On the other hand, the value of x must be increased by 1 when y is not an integer to compute the new value of y.

Algorithm 1 FFA-1

```
1: Begin
2: Initialize the value of $x = \lceil\sqrt{n}\rceil$
3: $y = \sqrt{x^2 - n}$
4: while $y$ is not an integer do
5:     $x = x + 1$
6:     $y^2 = x^2 - n$
7:     $y = \sqrt{y^2}$
8: end while
9: $p = x + y$
10: $q = x - y$
11: End
```

In Algorithm 1, it implies that the total iterations for computing x and y are same; their total iterations are $(p + q)/2 - \lceil\sqrt{n}\rceil$. However, FFA-1 must take time to compute the square root of integer for all iterations. At present, many modified factorization algorithms modified from FFA-1 were proposed to decrease computation time. The key of some algorithms which will be mentioned in Section 2.2 to 2.5 is to remove some of unrelated steps in the main loop.

The process of the second algorithm, is called FFA-2, is different from FFA-1. This algorithm does not compute the square root of integer as follows: From Equation (1), we have

$$4n = u^2 - v^2$$

where $u = p+q$ and $v = p-q$, therefore, the aim of FFA-2 is to find the corrected values of u and v, respectively.

However, the initial values of u and v are $2\lceil\sqrt{n}\rceil$ and 0, respectively. Assign $r = u^2 - v^2 - 4n$, then two prime factors of n, $p = \frac{u+v}{2}$ and $q = \frac{u-v}{2}$, are found whenever the value of r which is equal to 0 is found. However, two conditions of r are considered when r is not equal to 0 as follows:

Condition 1 ($r > 0$):
The value of v is too small but the value of r is too large. Therefore, v must be increased. On the other hand r must be decreased:

$$\begin{aligned} r &= r - (4v + 4) \\ v &= v + 2. \end{aligned} \tag{3}$$

From Equation (3), the value of $4v + 4$ is from: $(v + 2)^2 - v^2 = 4v + 4$. Because, $v = p - q$ is always an even number, the value of v can be increased by 2.

Condition 2 ($r < 0$):
The values of u and r are too small. Therefore, they have to be increased:

$$\begin{aligned} r &= r + (4u + 4) \\ u &= u + 2 \end{aligned} \tag{4}$$

From Equation (4), the value of $4u + 4$ is from: $(u + 2)^2 - u^2 = 4u + 4$ Because, $u = p + q$ is always an even number like the value of v, the value of u can be increased by 2. Therefore, the algorithm of FFA-2 for finding two prime factors of n, p and q, is as follows.

Algorithm 2 FFA-2

```
1: Begin
2: Initialize the value of $u = 2\lceil\sqrt{n}\rceil$ and $v = 0$
3: $r = u^2 - v^2 - 4n$
4: while $r$ is not equal to zero do
5:     if $r$ is more than zero then
6:         $r = r - (4v + 4)$
7:         $v = v + 2$
8:     else
9:         $r = r + (4u + 4)$
10:        $u = u + 2$
11:    end if
12: end while
13: $p = \frac{u+v}{2}$
14: $q = \frac{u-v}{2}$
15: End
```

In Algorithm 2, it implies that the value of u is always increased by 2 whenever the value of r is less than 0. On the other hand, the value of v is always increased by 2 whenever the value of r is more than 0. In general, Algorithm 2 shows that the total iterations for computing u and v are different, total iterations of u and v are $(p + q) - 2\lceil\sqrt{n}\rceil$ and $p - q$ in order.

2.2 Modified Fermat Factorization Version 2: MFFV2

MFFV2 [14] does not compute the value of y, Step 6 (or Line 7) of FFA-1, whenever the least significant digit of y^2 is 2, 3, 7 or 8 because y is not certainly an integer. That means the computation time is decreased, although the iterations in the loop are still stable.

2.3 Modified Fermat Factorization Version 3: MFFV3

In 2014, MFFV3 [16] was proposed by using Difference's Least Significant Digit Table (DLSDT) for removing some values of y^2 and y, Step 5 - 6 of FFA-1, from the computation. DLSDT will show the result of the least significant digit of y^2 without the computation of y^2 directly. That means it does not take time to compute y^2 and y when the least significant digit of y^2 is equal to 2, 3, 7 or 8. Furthermore, the information in DLSDT implies that the sequence of 10 iterations in the loop of FFA-1, MFFV3 can remove the 4 steps for computing y^2 and y when the least significant digit of n is 1 or 9. However, this algorithm becomes removing the 6 steps to compute y^2 and y when the least significant digit of n is 3 or 7.

2.4 Modified Fermat Factorization Version 4: MFFV4

MFFV4 [13] can remove more steps for computing y^2 and y when compared with MFFV3. This algorithm uses the new table called Y2MOD20 for analyzing the result of y^2 modulo 20 without computing y^2 directly. From the mathematical theorem, the result of perfect square modulo 20 is always equal to 0, 1, 4, 5, 9 or 16 [9]. Therefore, if the result of y^2 modulo 20 is not equal to 0, 1, 4, 5, 9 or 16, then y^2 is not certainly a perfect square. However, for MFFV4, the value of n is divided into 8 cases. Each case is from the result of n modulo 20 which is 1, 3, 7, 9, 11, 13, 17 or 19. In addition, the sequence of 10 iterations in the loop of FFA-1, MFFV4 can remove 7 steps to compute y^2 and y when the result of n modulo 20 is 1, 9, 11 or 19. Nevertheless, this algorithm can remove 8 steps when the result of n modulo 20 is 3, 7, 13 or 17. Furthermore, the experiment showed that MFFV4 is the fastest improved Fermat's factorization algorithm when compared to FFA-1, MFFV2 and MFFV3.

2.5 Possible Prime Modified Fermat Factorization: P^2MFF

The concept of P^2MFF [17] is different from all improved Fermat's factorization algorithms which had been mentioned. In general, this algorithm will be divided into 2 sub algorithms. Each algorithm is arised from the result of n modulo 6. If the result is equal to 5, the form of n is $6k$ - 1 and the form of x must be always $3k_1$, k and k_1 are any integers. That means all iterations of the computation are from the result of $x\%3 = 0$. On the other hand, if the result of n modulo 6 is equal to 1, the form of n is $6k+1$ and the form of x must not be $3k_1$. That means all iterations of the computation are from the result of x%3 = 1 or 2.

3 The Proposed Method

Notation: Assume $z, z_1, z_2 \in Z^+$ and $z_1 \geq z_2$

1) $LSG_1(z)$ = LSG(z) is the last digit of z
2) LSG($z_1 + z_2$) = LSG($LSG(z_1)$ + LSG(z_2))
3) LSG($z_1 - z_2$) = LSG(LSG(z_1) − LSG(z_2)+10)
4) LSG($z_1 * z_2$) = LSG(LSG(z_1) ∗ LSG(z_2))
5) $LSG_m(z)$ is the last m digits of $z, m > 1$
6) $LSG_m(z_1 + z_2) = LSG_m(LSG_m(z_1) + LSG_m(z_2))$
7) $LSG_m(z_1 - z_2) = LSG_m(LSG_m(z_1) - LSG_m(z_2) + 10^m)$
8) $LSG_m(z_1 * z_2) = LSG_m(LSG_m(z_1) * LSG_m(z_2))$

For the notation above, if the result of $LSG(z_1) - LSG(z_2)$ or $LSG_m(z_1) - LSG_m(z_2)$ is a negative integer, it will be increased by 10 or 10^m respectively for changing the result to be the positive integer.

Example 1.

$$\begin{aligned}
LSG(31) &= 1 \\
LSG(17+21) &= LSG(LSG(17)+LSG(21)) \\
&= LSG(7+1) = LSG(8) = 8 \\
LSG(38-15) &= LSG(LSG(38)-LSG(15)+10) \\
&= LSG(8-5+10) = LSG(13) = 3 \\
LSG(32-15) &= LSG(LSG(32)-LSG(15)+10) \\
&= LSG(2-5+10) = LSG(7) = 7 \\
LSG(5*14) &= LSG(LSG(5)*LSG(14)) \\
&= LSG(5*4) = LSG(20) = 0.
\end{aligned}$$

The objective of this paper is to propose the new modified algorithm improved from FFA-2. The key is that the algorithm of FFA-2 will be divided into many sub algorithms based on the last m digits of n. These algorithms are called Specific Fermat's Factorization Algorithm Considered from X (SFFA-X) where X is represented as the last m digits of n. SFFA-X can leave out some values of u and v which are not the expected values. Nevertheless, if $LSG_m(n)$ is considered, numbers of SFFA-X for last m digits of n are based on the values of m that is equal to $4 * 10^{m-1}$.

In fact, the set of the possible values of u and v for SFFA-X are from the following definition: (Assign a, b, p_t and $q_t \in Z^+$ and n is represented as the modulus).

$$\begin{aligned}
S = \{&(a, b, p_t, q_t, n) \mid a = LSG_m(p_t), \\
&b = LSG_m(q_t) \text{ and } LSG_m(ab) = LSG_m(n)\}
\end{aligned}$$

From the set of S, if the last m digits of the multiplication result between the last m digits of p_t, $a = LSG_m(p_t)$, and the last m digits of q_t, $b = LSG_m(q_t)$, is equal to the last m digits of n, $LSG_m(n) = LSG_m(ab)$, then, the values of $LSG_m(u)$ and $LSG_m(v)$ can be computed from $LSG_m(a+b)$ and $LSG_m(a-b)$ or $LSG_m(b-a)$, respectively.

In general, all possible values of $LSG_m(u)$ and $LSG_m(v)$ can be found by considering the both following theorems.

Theorem 1. *Assign a and b are any positive odd integers, where $LSG_m(a) = a_m a_{m-1} \cdots a_2 a_1$, $LSG_m(b) = b_m b_{m-1} \cdots b_2 b_1$ and $m > 1$, that the result of $LSG_m(ab)$ is equal to $LSG_m(n)$. If a_1 is equal to b_1, then the results of $LSG_m((a + 10^{m-1})(b + 9(10^{m-1})))$ and $LSG_m((a + 9(10^{m-1}))(b + 10^{m-1}))$ are also equal to $LSG_m(n)$.*

Proof. Assign: $LSG_m(ab) = c_m c_{m-1} \cdots c_2 c_1$, $LSG_m(a + b) = LSG_m(u)$, $LSG_m(a - b) = LSG_m(v_1)$, $LSG_m(b - a) = LSG_m(v_2)$ and $c_i = d_i\%10$, where $i = 1, 2, \cdots, m$.

From the multiplication technique, we have

$$\begin{aligned} d_1 &= a_1 b_1 \\ d_2 &= a_2 b_1 + a_1 b_2 + \lfloor \frac{d_1}{10} \rfloor \\ d_m &= a_m b_1 + a_{m-1} b_2 + \cdots + a_1 b_m + \lfloor \frac{d_{m-1}}{10} \rfloor \end{aligned}$$

Case 1: Assign $LSG_m(a + 10^{m-1}) = e_m e_{m-1} \cdots e_2 e_1$, $LSG_m(b+9(10^{m-1})) = f_m f_{m-1} \cdots f_2 f_1$, $LSG_m((a+10^{m-1})(b + 9(10^{m-1}))) = g_m g_{m-1} \cdots g_2 g_1$ and $g_i = h_i\%10$. Because $LSG_{(m-1)}(a) = LSG_{(m-1)}(a + 10^{m-1})$ and $LSG_{(m-1)}(b) = LSG_{(m-1)}(b + 9(10^{m-1}))$, hence $e_j = a_j$, $f_j = b_j$ and $h_j = d_j$ where $j = 1, 2, \cdots, m-2, m-1$. In addition, $e_m = (a_m+1)\%10$ and $f_m = (b_m+9)\%10$. Therefore, From the multiplication technique, we have

$$\begin{aligned} g_m &= (e_m f_1 + e_{m-1} f_2 + \cdots + e_1 f_m + \lfloor \frac{h_{m-1}}{10} \rfloor)\%10 \\ &= ((a_m + 1)b_1 + a_{m-1} b_2 + \cdots + a_1(b_m + 9) \\ &\qquad + \lfloor \frac{d_{m-1}}{10} \rfloor)\%10 \\ &= (a_m b_1 + a_{m-1} b_2 + \cdots + a_1 b_m + \lfloor \frac{d_{m-1}}{10} \rfloor \\ &\qquad + (b_1 + 9a_1))\%10. \end{aligned}$$

Because $a_1 = b_1$, that means $(b_1 + 9a_1)\%10 = 0$. Hence, $g_m = (a_m b_1 + a_{m-1} b_2 + \cdots + a_1 b_m + \lfloor \frac{d_{m-1}}{10} \rfloor)\%10 = c_m$.

Moreover, g_j is also equal to c_j because a_j and b_j are always equal to e_j and f_j, respectively. Therefore, $LSG_m((a+10^{m-1})(b+9(10^{m-1})))$ is always equal to $LSG_m(ab)$ that means it is also equal to $LSG_m(n)$.

However, this case implies that for any pair of a and b that a_1 and b_1 are fixed, there is only one value of $LSG_m(u)$, $LSG_m((a + 10^{m-1}) + (b + 9(10^{m-1}))) = LSG_m(a + b + 10^m) = LSG_m(a + b) = LSG_m(u)$. On the other hand, the value of $LSG_m(v_1)$ is always decreased by $8(10)^{m-1}$, $LSG_m((a + 10^{m-1}) - (b + 9(10^{m-1}))) = LSG_m(v_1 - 8(10)^{m-1})$, and the value of $LSG_m(v_2)$ is always increased by $8(10)^{m-1}$, $LSG_m((b + 9(10^{m-1})) - (a + 10^{m-1})) = LSG_m(v_2 + 8(10^{m-1}))$, where v_1 and v_2 are represented as the possible values of v.

Case 2: Assign $LSG_m(a + 9(10^{m-1}))=e_m e_{m-1} \cdots e_2 e_1$, $LSG_m(b + 10^{m-1})= f_m f_{m-1} \cdots f_2 f_1$, $LSG_m((a + 9(10^{m-1}))(b + 10^{m-1})) = g_m g_{m-1} \cdots g_2 g_1$ and $g_i = h_i\%10$.

Because $LSG_{(m-1)}(a) = LSG_{(m-1)}(a + 9(10^{m-1}))$ and $LSG_{(m-1)}(b) = LSG_{(m-1)}(b + 10^{m-1})$, hence $e_j = a_j$, $f_j = b_j$ and $h_j = d_j$ where $j = 1, 2, \cdots, m-2, m-1$. In addition, $e_m = (a_m + 9)\%10$ and $f_m = (b_m + 1)\%10$.

From the multiplication technique, we have

$$\begin{aligned} g_m &= (e_m f_1 + e_{m-1} f_2 + \cdots + e_1 f_m + \lfloor \frac{h_{m-1}}{10} \rfloor)\%10 \\ &= ((a_m + 9)b_1 + a_{m-1} b_2 + \cdots + a_1(b_m + 1) \\ &\qquad + \lfloor \frac{d_{m-1}}{10} \rfloor)\%10 \\ &= (a_m b_1 + a_{m-1} b_2 + \cdots + a_1 b_m + \lfloor \frac{d_{m-1}}{10} \rfloor \\ &\qquad + (a_1 + 9b_1))\%10. \end{aligned}$$

Because $a_1 = b_1$, that means $(a_1 + 9b_1)\%10 = 0$. Hence, $g_m = (a_m b_1 + a_{m-1} b_2 + \cdot + a_1 b_m + \lfloor \frac{d_{m-1}}{10} \rfloor)\%10 = c_m$.

Moreover, g_j is also equal to c_j because a_j and b_j are always equal to e_j and f_j, respectively. Therefore, $LSG_m((a+9(10^{m-1}))(b+10^{m-1}))$ is always equal to $LSG_m(ab)$. That means it is also equal to $LSG_m(n)$.

However, this case implies that there is only one value of $LSG_m(u)$, $LSG_m((a+9(10^{m-1})) + (b+10^{m-1}) = LSG_m(a + b) = LSG_m(u)$. On the other hand, the value of $LSG_m(v_1)$ is always increased by $8(10)^{m-1}$, $LSG_m((a+9(10^{m-1})) - (b+10^{m-1})) = LSG_m(v_1 + 8(10)^{m-1})$, and the value of $LSG_m(v_2)$ is always decreased by $8(10)^{m-1}$, $LSG_m((b + 10^{m-1}) - (a + 9(10^{m-1}))) = LSG_m(v_2 - 8(10^{m-1}))$, when v_1 and v_2 are represented as the possible values of v.

$\square$

Theorem 2. *Assign a and b are any positive odd integers, where $LSG_m(a) = a_m a_{m-1} \cdots a_2 a_1$, $LSG_m(b) = b_m b_{m-1} \cdots b_2 b_1$ and $m > 1$, that the result of $LSG_m(ab)$ is equal to $LSG_m(n)$. If a_1 is not equal to b_1 and there are various pairs of positive odd integers, k_1 and k_2, which are equal or smaller than 9 and not equal to 5 that the result of $(a_1 k_2 + b_1 k_1)\%10 = 0$, then the result of $LSG_m((a + 10^{m-1}k_1)(b + 10^{m-1}k_2))$ is also equal to $LSG_m(n)$.*

Proof. Assign: $LSG_m(ab) = c_m c_{m-1} \cdots c_2 c_1$, $LSG_m(a + 10^{m-1}k_1) = x_m x_{m-1} \cdots x_2 x_1$, $LSG_m(b + 10^{m-1}k_2) = y_m y_{m-1} \cdots y_2 y_1$, $LSG_m((a + 10^{m-1}k_1)(b + 10^{m-1}k_2)) = z_m z_{m-1} \cdots z_2 z_1$, $LSG_m(a+b) = LSG_m(u)$, $LSG_m(a-b) = LSG_m(v_1)$, $LSG_m(b-a) = LSG_m(v_2)$, $c_i = d_i\%\ 10$ and $z_i = w_i\%10$, where $i = 1, 2, 3, \cdots, m$.

From the multiplication technique, we have $c_m = (a_m b_1 + a_{m-1}b_2 + \cdots + a_1 b_m + \lfloor \frac{d_{m-1}}{10} \rfloor)\%10$.

Because $LSG_{(m-1)}(a) = LSG_{(m-1)}(a + 10^{m-1}k_1)$ and $LSG_{(m-1)}(b) = LSG_{(m-1)}(b + 10^{m-1}k_2)$, then $x_j = a_j, y_j = b_j$ and $w_j = d_j$ where $j = 1, 2, \cdots, m-2, m-1$. In addition, $x_m = (a_m + k_1)\%10$ and $y_m = (b_m + k_2)$ % 10. Therefore, $z_m = (x_m y_1 + x_{m-1}y_2 + \cdots + x_1 y_m + \lfloor \frac{w_{m-1}}{10} \rfloor)\%10 = ((a_m + k_1)b_1 + a_{m-1}b_2 + \cdots + a_1(b_m + k_2) + \lfloor \frac{d_{m-1}}{10} \rfloor)\%10 = (a_m b_1 + a_{m-1}b_2 + \cdots + a_1 b_m + \lfloor \frac{d_{m-1}}{10} \rfloor + (a_1 k_2 + b_1 k_1))\%10$.

Because the result of $(a_1k_2 + b_1k_1)\%10$ is equal to 0, $z_m = (a_m b_1 + a_{m-1}b_2 + \cdots + a_1 b_m + \lfloor \frac{d_{m-1}}{10} \rfloor)\%10 = c_m$.

Moreover, z_j is also equal to c_j because a_j and b_j are always equal to x_j and y_j, respectively. Therefore, $LSG_m((a + 10^{m-1}k_1)(b + 10^{m-1}k_2))$ is always equal to $LSG_m(ab)$ and $LSG_m(n)$ whenever the result of $(a_1k_2 + b_1k_1)\%10$ is equal to 0.

However, this theory implies that for any pair of a and b that a_1 and b_1 are fixed, the value of $LSG_m(u)$ is always increased by $10^{m-1}k$, where $k = k_1 + k_2$, as follows:

$$\begin{aligned} & LSG_m((a + 10^{m-1}k_1) + (b + 10^{m-1}k_2)) \\ = \; & LSG_m(a + b + 10^{m-1}k_1 + 10^{m-1}k_2) \\ = \; & LSG_m(a + b + 10^{m-1}(k_1 + k_2)) \\ = \; & LSG_m(a + b + 10^{m-1}k) \\ = \; & LSG_m(u + 10^{m-1}k) \end{aligned}$$

On the other hand, the value of $LSG_m(v_1)$ is always increased by $10^{m-1}l$, where $l = ((k_1 - k_2) + 10)\ \%\ 10$ is always a positive integer, as follows:

$$\begin{aligned} & LSG_m((a + 10^{m-1}k_1) - (b + 10^{m-1}k_2)) \\ = \; & LSG_m(a - b + 10^{m-1}k_1 - 10^{m-1}k_2) \\ = \; & LSG_m(a - b + 10^{m-1}(k_1 - k_2)) \\ = \; & LSG_m(a - b + 10^{m-1}(((k_1 - k_2) + 10)\%10)) \\ = \; & LSG_m(a - b + 10^{m-1}l) \\ = \; & LSG_m(v_1 + 10^{m-1}l). \end{aligned}$$

And the value of $LSG_m(v_2)$ is always increased by $10^{m-1}h$, where $h = ((k_2 - k_1) + 10)\ \%\ 10$ is always a positive integer, as follows:

$$\begin{aligned} & LSG_m((b + 10^{m-1}k_2) - (a + 10^{m-1}k_1)) \\ = \; & LSG_m(b - a + 10^{m-1}k_2 - 10^{m-1}k_1) \\ = \; & LSG_m(b - a + 10^{m-1}(k_2 - k_1)) \\ = \; & LSG_m(b - a + 10^{m-1}(((k_2 - k_1) + 10)\%10)) \\ = \; & LSG_m(b - a + 10^{m-1}h) \\ = \; & LSG_m(v_2 + 10^{m-1}h). \end{aligned}$$

Therefore, we can conclude that the values of $LSG_m(u)$, $LSG_m(v_1)$ and $LSG_m(v_2)$ are always increased by $10^{m-1}k$, $10^{m-1}l$ and $10^{m-1}h$, respectively, where k, l and h are any positive integers and v_1 and v_2 are represented as the possible values of v. □

Notation: If k_1 and k_2 are an even number or equal to 5, some values of $LSG_m(u)$ and $LSG_m(v)$ cannot be computed because we cannot find some pairs of $LSG_m(a)$ and $LSG_m(b)$ that $LSG_m(ab) = LSG_m(n)$.

From both of two theorems above, assume a and b are represented as any positive odd integers which the last digit is not equal to 5 and all pairs of $LSG_{(m-1)}(a)$ and $LSG_{(m-1)}(b)$ that $LSG_{(m-1)}(ab)$ is equal to $LSG_{(m-1)}(n)$ are known. All values of $LSG_m(u)$ and $LSG_m(v)$ will be found whenever only one pair of $LSG_m(a)$ and $LSG_m(b)$ that $LSG_m(ab)$ is equal to $LSG_m(n)$ for each pair of $LSG_{(m-1)}(a)$ and $LSG_{(m-1)}(b)$ is found. In deep, all the other pairs of $LSG_m(a)$ and $LSG_m(b)$ are computed by using Theorem 1 when $LSG(a)$ is equal to $LSG(b)$ or using Theorem 2 when $LSG(a)$ is not equal to $LSG(b)$, until the repeated solution is found.

Due to all pairs of $LSG(a)$ and $LSG(b)$ that $LSG(ab)$ is equal to $LSG(n)$ cannot be computed by using Theorem 1 and Theorem 2, therefore these pairs must be considered directly. Table 1 shows all possible pairs of $LSG(a)$ and $LSG(b)$ for all possible values of $LSG(n)$.

Generally, Table 1 shows the following information:

1) If $LSG(n)$ equals to 1, there are 3 possible pairs of $(LSG(a), LSG(b))$, 3 values of $LSG(u)$ and 3 values of $LSG(v)$.

2) If $LSG(n)$ equals to 3, there are 2 possible pairs of $(LSG(a), LSG(b))$, 2 values of $LSG(u)$ and 2 values of $LSG(v)$.

3) If $LSG(n)$ equals to 7, there are 2 possible pairs of $(LSG(a), LSG(b))$, 2 values of $LSG(u)$ and 2 values of $LSG(v)$.

4) If $LSG(n)$ equals to 9, there are 3 possible pairs of $(LSG(a), LSG(b))$, 3 values of $LSG(u)$ and 3 values of $LSG(v)$.

Assume all pairs of $(LSG_{(m-1)}(a), LSG_{(m-1)}(b))$ that the last $m-1$ digits of their multiplication which is equal to $LSG_{(m-1)}(n)$ are known. To complete the speed up for factoring n by improving FFA-2, the algorithm of SFFA-X is divided into 3 algorithms.

First is the main algorithm. This algorithm is used to find two prime factors, p and q.

Moreover, it implies that some unrelated values of U and V which their last m digits are not in the sets of $LSG_m(u)$ and $LSG_m(v)$ are left out from the computation. Leaving out values of U and V is from

Table 1: All possible values of LSG(u), LSG(v) and pairs of LSG(a) and LSG(b) when considered from LSG(n)

LSG(n)	**Pair of (LSG(a),LSG(b))**	**LSG(LSG(a)+LSG(b)) (LSG(u))**	**LSG(LSG(a)-LSG(b)) or LSG(LSG(b)-LSG(a)) (LSG(v))**
1	(1, 1)	2	0
	(3, 7)	0	6 or 4
	(9, 9)	8	0
3	(1, 3)	4	8 or 2
	(7, 9)	6	8 or 2
7	(1, 7)	8	4 or 6
	(3, 9)	2	4 or 6
9	(1, 9)	0	2 or 8
	(3, 3)	6	0
	(7, 7)	4	0

Algorithm 3 FFA-X

```
1: Begin
2: Compute all members of $LSG_m(u)$, $LSG_m(v)$, $disu$ and $disv$ by using Algorithm 4. In deep, $disu$ is the set of the subtraction results between two adjacent values of $LSG_m(u)$ and $disv$ is the set of the subtraction results between two adjacent values of $LSG_m(v)$.
3: Find the initial values of $U$ and $V$ that $LSG_m(U)$ and $LSG_m(V)$ must equal to one of all possible values of $LSG_m(u)$ and $LSG_m(v)$, in order. In addition, $U$ and $V$ are started at $2\lceil\sqrt{n}\rceil$ and 0 respectively.
4: Assign $i$ is the index of $disu$ that the initial value is based on the values of $LSG_m(U)$ and the position in $LSG_m(u)$ and $disu$.
5: Assign $j$ which is always started at 0 is the index of $disv$.
6: $r = U^2 - V^2 - 4n$
7: while $r$ is not equal to zero do
8:   if $r$ is more than zero then
9:     $r = r - (2 * V * disv(j) + disv(j)^2)$
10:    $V = V + disv(j)$
11:    $j = j + 1$ // $j$ becomes to zero whenever $j$ is equal to the size of $disv$
12:  else
13:    $r = r + (2 * U * disu(i) + disu(i)^2)$
14:    $U = U + disu(i)$
15:    $i = i+1$, // $i$ becomes to zero whenever $i$ is equal to the size of $disu$
16:  end if
17: end while
18: $p = \frac{U+V}{2}$
19: $q = \frac{U-V}{2}$
20: End
```

choosing only the pairs of ($LSG_m(a)$, $LSG_m(b)$) which $LSG_m(LSG_m(a)*LSG_m(b)) = LSG_m(ab)$ is equal to $LSG_m(n)$.

Second is the algorithm for finding all members of $LSG_m(u)$, $LSG_m(v)$, $disu$ and $disv$ before returning these variables to Step 1 of Algorithm 3.

Algorithm 4 Finding $LSG_m(u)$, $LSG_m(v)$, $disu$ and $disv$

```
1: Begin
2: Assign each pair of ($LSG_{(m-1)}(a)$, $LSG_{(m-1)}(b)$) is a group.
3: For each group, find a pair of ($LSG_m(a)$, $LSG_m(b)$) using Algorithm 5 and then leave out a pair of ($LSG_{(m-1)}(a)$, $LSG_{(m-1)}(b)$) from the group.
4: For each group, find all pairs of ($LSG_m(a)$, $LSG_m(b)$) by using theorem 1 whenever $LSG(a)$ is equal to $LSG(b)$. However, theorem 2 is used for the other case.
5: Compute all possible values of $LSG_m(u)$ and $LSG_m(v)$. However, all repeated values of $LSG_m(u)$ and $LSG_m(v)$ will be left out from the sets.
6: Sort the members of $LSG_m(u)$ and $LSG_m(v)$ from the minimum to the maximum.
7: Find $disu$, the set of the subtraction results between two adjacent values of $LSG_m(u)$. Nevertheless, the last member of $disu$ is the subtraction between the minimum and the maximum of $LSG_m(u)$. In addition, the result of the last member must be also increased by $10^m$ for changing as the positive integer.
8: Find $disv$, the set of the subtraction results between two adjacent values of $LSG_m(v)$. Nevertheless, the last member of $disv$ is the subtraction between the minimum and the maximum of $LSG_m(v)$. Moreover, this result must be increased by $10^m$.
9: End
```

However, the results from this algorithm can be applied with all values of $LSG_m(n)$ which $LSG_m(u)$, $LSG_m(v)$, $disu$ and $disv$ had been computed. Therefore, only the first time of the computation will be computed to find all pairs of ($LSG_m(a)$, $LSG_m(b)$) to compute all members of these four variables.

Furthermore, if $disu$ or $disv$ have the repeated patterns, we can leave out them from the sets.

Example 2. *Assume, all members of $LSG_2(u)$ and $LSG_2(v)$ of SFFA-03 are known. Finding disu and disv* *of SFFA-03:*

1) *$LSG_2(u)$ = {04, 16, 24, 36, 44, 56, 64, 76, 84, 96}. Therefore, disu = {12, 8, 12, 8, 12, 8, 12, 8, 12, 8}. Because, disu has the repeated patterns, then, we can reassign disu = {12, 8}.*

2) *$LSG_2(v)$ = {02, 18, 22, 38, 42, 58, 62, 78, 82, 98}. Therefore, disv = {16, 4, 16, 4, 16, 4, 16, 4, 16, 4}. Because, disv has the repeated patterns, then, we can reassign disv = {16, 4}.*

In deep, if the value of all members of disu (or disv) is same, these members can be reduced to be only one member of the set. For example, if disv ={20, 20, 20, 20}, we can reduce as disv = 20 and j will be left out from the algorithm because the value of all members in disv is same.

The last algorithm is for computing the pair of ($LSG_m(a)$, $LSG_m(b)$) when a pair of ($LSG_{(m-1)}(a)$, $LSG_{(m-1)}(b)$) is known. The idea of this algorithm is to find only the value of $LSG_m(b)$ while $LSG_m(a)$ is always equal to $0LSG_{(m-1)}(a)$. For example, assume $LSG_3(a) = 841$, then $LSG_4(a) = 0841$.

Algorithm 5 Finding Pair of $LSG_m(a)$ and $LSG_m(b)$

```
1: Begin
2: $x = LSG_{(m-1)}(a) * LSG_{(m-1)}(b)$
3: $n_m = \lfloor \frac{n\%10^m}{10^{m-1}} \rfloor$
4: $x_m = \lfloor \frac{x\%10^m}{10^{m-1}} \rfloor$
5: $a_1$ = LSG$_{(m-1)}(a)$ % 10
6: Assign $i = 0$
7: $j = (i * a_1 + x_m)\%10$
8: while $j$ is not equal to $n_m$ do
9:     $i = i + 1$
10:    $j = (i * a_1 + x_m)\%10$
11: end while
12: $LSG_m(a) = 0LSG_{(m-1)}(a)$
13: $LSG_m(b) = LSG_{(m-1)}(b) + i * 10^{m-1}$
14: End
```

Example 3. *Assign $LSG_3(a) = 233$ and $LSG_3(b) = 897$, $LSG_3(ab) = 001$. Find a pair of ($LSG_4(a)$, $LSG_4(b)$) that $LSG_4(ab) = 2001$ by using Algorithm 5.*

In this example, $LSG_4(ab) = 2001$ can be represented as the value of $LSG_4(n)$.

$$\begin{aligned} x &= 233 * 897 = 209,001 \\ n_4 &= \lfloor \frac{2001\%10^4}{10^3} \rfloor = 2 \\ x_4 &= \lfloor \frac{209,001\%10^4}{10^3} \rfloor = 9 \\ a_1 &= 233\%10 = 3 \end{aligned}$$

where $i = 0, j = (0 * 3 + 9)\%10 = 9, j$ is not equal to 2. $i = 1, j = (1 * 3 + 9)\%10 = 2, j$ is equal to 2.

Then, $LSG_4(a) = 0233$ and $LSG_4(b) = LSG_3(b) + i * 10^3 = 897 + 1000 = 1897$. However, for some values of n, the implementation of SFFA-X should be divided into 2 groups, based on the relation between LSG(a) and LSG(b). In fact, there are 2 cases of SFFA-X as follows:

Case 1: *The algorithm is always divided into 2 groups whenever there may be the pairs of a and b that LSG(a) is equal to LSG(b).*

Group 1: *The algorithm for all pairs of a and b that LSG(a) is equal to LSG(b), the value of LSG(v) is always equal to 0.*

Group 2: *The algorithm for all pairs of a and b that LSG(a) is not equal to LSG(b), the value of LSG(u) is always equal to 0.*

Case 2: *There is only one group whenever there is no pairs of a and b that LSG(a) is equal to LSG(b), both of LSG(u) and LSG(v) are not always equal to 0.*

For the case of n that must be divided into 2 groups, there is only one solution in only one group. Therefore, the process is ended when the corrected solution in one out of two groups is found.

Example 4. *Factoring $n = 1287901$ using SFFA-X with $m = 2$.*
This example is assigned to use SFFA-X with m = 2 to find two prime factors of n. Because $LSG_2(n)$ = 01, therefore SFFA-01 is the algorithm used for the implementation. However, all steps to find two prime factors of n = 1287901 using SFFA-01 are as follows:
Assumption: *All pairs of (LSG(a), LSG(b)) that LSG(ab) is equal to LSG(n) = 1 are known.*

Algorithm 3:

Step 1: *Find all of $LSG_2(u)$, $LSG_2(v)$, disu and disv by using Algorithm 4.*
Algorithm 4:

Step 1: *Group 1: (1, 1); Group 2: (3, 7); Group 3: (9, 9).*

Step 2: *Group 1: (01, 01); Group 2: (03, 67); Group 3: (09, 89). This process is computed by using Algorithm 5.*

Step 3:

Group 1: *(01, 01), (11, 91), (21, 81), (31, 71), (41, 61), (51, 51);*

Group 2: *(03, 67), (13, 77), (23, 87), (33, 97), (43, 07), (53, 17), (63, 27), (73, 37), (83, 47), (93, 57). In this case, one of all possible pairs of (k_1, k_2) which are (1, 1), (3, 3), (7, 7) or (9, 9) is chosen.*

Group 3: *(09, 89), (19, 79), (29, 69), (39, 59), (49, 49), (99, 99).*

Steps 4 - 5: *Because there are some pairs of a and b that LSG(a) is equal to LSG(b), SFFA-01 must be divided into 2 group. Group 1 is from the combining between Group 1 and Group 3, LSG(a) is equal to LSG(b). The other is that LSG(a) is not equal to LSG(b). Therefore,*

Group 1: $LSG_2(u) = \{02, 98\}$; $LSG_2(v) = \{00, 20, 40, 60, 80\}$.

Group 2: $LSG_2(u) = \{10, 30, 50, 70, 90\}$. $LSG_2(v) = \{36, 64\}$.

Steps 6-7 *Group 1:* $disu = \{96, 4\}$ *and* $disv = \{20, 20, 20, 20, 20\} = 20$, *do not assign* j. *Group 2:* $disu = \{20, 20, 20, 20, 20\} = 20$, *do not assign* i, *and* $disv = \{28, 72\}$.

End of Algorithm 4.

Steps 2-4: *Find the initial value of* U, V, i *and* j *for each group. First, compute* $U = 2\lceil\sqrt{n}\rceil = 2270$.

Group 1: *Because* $LSG_2(U) = 70$ *is not a member in the set of* $LSG_2(u)$, U *must be changed as 2298, the nearest value which is more than 2270. That means the initial value of* i *must be equal to 1. However, the initial value of* V *is 0 because the minimum value of* $LSG_2(v)$ *is 0.*

Group 2: *Because* $LSG_2(U) = 70$ *is already a member in the set of* $LSG_2(u)$, $U = 2270$ *can be used as the initial value. However, the variable,* i, *is not used in this group. In addition, the initial value of* V *is 36 because the minimum value of* $LSG_2(v)$ *is 36.*

Step 5:

Group 1: $r = 2298^2 - 0^2 - 4(1287901) = 129200$.

Group 2: $r = 2270^2 - 36^2 - 4(1287901) = \boldsymbol{0}$.

In Step 5, the expected solution is found in Group 2, $r = 0$. *Therefore, the process in the loop, Steps 6 - 16, will not be implemented. However, the two prime factors can be computed from* $p = \frac{U+V}{2} = \frac{2270+36}{2} = 1153$ *and* $p = \frac{U-V}{2} = \frac{2270-36}{2} = 1117$.

Moreover, for the value of n *in Example 3, assume SFFA-901,* $m = 3$, *is chosen to factor* n *instead of using SFFA-01. All pairs of* ($LSG_2(a)$, $LSG_2(b)$) *which are found in this example will be used in Step 1 of SFFA-901 in order to find all pairs of* ($LSG_3(a)$, $LSG_3(b)$) *that* $LSG_3(LSG_3(a) * LSG_3(b))$ *is equal to* $LSG_3(n) = 901$.

Example 5. *Factoring* $n = 133901$ *using SFFA-X with* $m = 2$.
This example is assigned $LSG_2(n) = 01$. *That means SFFA-01 is chosen for this example. Because* $LSG_2(u)$, $LSG_2(v)$, *disu and disv of SFFA-01 had been already computed in Example 4, it is not time-consuming to calculate them again and we can start the process at Step 2 of Algorithm 3.*

Algorithm 3:

Steps 2 - 4: *Find the initial value of* U, V, i *and* j *for each group. First, compute* $U = 2\lceil\sqrt{n}\rceil = 732$.

Group 1: *Because* $LSG_2(U) = 32$ *is not a member in the set of* $LSG_2(u)$, U *must be changed as 798. That means the initial value of* i *must be equal to 1. However, the initial value of* V *is 0 because the minimum value of* $LSG_2(v)$ *is 0.*

Group 2: *Because* $LSG_2(U) = 32$ *is a not member in the set of* $LSG_2(u)$, U *must be changed as 750. However, the variable,* i, *is not used in this group. In addition, the initial value of* V *must be 36 because the minimum value of* $LSG_2(v)$ *is 36.*

Step 5:

Group 1: $r = 798^2 - 0^2 - 4(133901) = 101200$.

Group 2: $r = 750^2 - 36^2 - 4(133901) = 25600$.

Steps 6 - 16: *Process in loop:*

Iteration 1:

Group 1: $(r > 0, V = 0)$:

$$\begin{aligned} r &= r - (2 * V * disv + disv^2) = 100800 \\ V &= V + disv = 20. \end{aligned}$$

Group 2: $(r > 0, V = 36, j = 0)$:

$$\begin{aligned} r &= r - (2 * V * disv(0) + disv(0)^2) = 22800. \\ V &= V + disv(0) = 64, j = 1. \end{aligned}$$

Iteration 2:

Group 1: $(r > 0, V = 20)$:

$$\begin{aligned} r &= r - (2 * V * disv + disv^2) = 99600. \\ V &= V + disv = 40. \end{aligned}$$

Group 2: $(r > 0, V = 64, j = 1)$:

$$\begin{aligned} r &= r - (2 * V * disv(1) + disv(1)^2) = 8400. \\ V &= V + disv(1) = 136, j = 0. \end{aligned}$$

Iteration 3:

Group 1: $(r > 0, V = 40)$:

$$\begin{aligned} r &= r - (2 * V * disv + disv^2) = 97600. \\ V &= V + disv = 60. \end{aligned}$$

Group 2: $(r > 0, V = 136, j = 0)$:

$$\begin{aligned} r &= r - (2 * V * disv(0) + disv(0)^2) = \boldsymbol{0}. \\ V &= V + disv(0) = 164. \end{aligned}$$

Because the expected value of r *which is equal to 0 is found in Group 2 of the* 3^{rd} *iteration, the process is stopped and then two prime factors can be computed from* $p = \frac{U+V}{2} = \frac{750+164}{2} = 457$ *and* $q = \frac{U-V}{2} = \frac{750-164}{2} = 293$.

4 Results and Discussion

In this work, the computer specifications for the implementation are Intel(R) Core(TM) i3 CPU M380 2.53 GHz, 4.00 GB RAM Memory, and Microsoft Windows 8.1 Pro Operating System. Java Programming Language is chosen to develop the algorithms. Moreover, we use BigInteger class which is the class of Java as the data type because this class can be represented as the unlimited data type. The experiment is distinguished as 4 parts. Each bits size in each experiment is the average result of 50 values of n chosen randomly. In addition, for Figure 1 and Figure 2, only the difference of bits size between two prime factors equal to 2 is chosen. However, all algorithms of SFFA-X in these experiments are started at Step 2 of Algorithm 3 because the process of Step 1 is always implemented only the first time of SFFA-X, when X is fixed.

The experiment in Figure 3 is for the same size of two prime factors of n and the values of n that SFFA-X must be divided into 2 groups, $LSG_4(n) = 0001$ is the representative of this experiment. That means only 4 algorithms of SFFA-X in which X is equal to 1, 01, 001 and 0001 are chosen for the implementation. The experiment shows that SFFA-0001 is the fastest algorithm. Nevertheless, the average computation time of FFA-2, SFFA-1, SFFA-01, SFFA-001, SFFA-0001, MFFV4 and P^{2}MFF are about 30.27, 17.17, 4.09, 2.05, 1.82, 5.78 and 15.15 seconds respectively. Furthermore, the information in this figure implies that SFFA-X begins to factor n faster than MFFV4 when X which is equal to 01 is chosen.

The difference between the experiment in Figure 3 and Figure 1 is that the size of two large prime factors in Figure 1 is different while the other in Figure 3 is same. Therefore, all algorithms of SFFA-X for this experiment are still SFFA-1, SFFA-01, SFFA-001 and SFFA-0001. However, the experiment in Figure 1 shows that SFFA-0001 is still the fastest integer factorization algorithm. Nevertheless, the average computation time of FFA-2, SFFA-1, SFFA-01, SFFA-001, SFFA-0001, MFFV4 and P^{2}MFF are about 243.2, 135.76, 29.4, 19.02, 10.73, 227.4 and 341.9 seconds respectively. Furthermore, this figure implies that SFFA-X begins to factor n faster than MFFV4 when X is equal to 1 is chosen.

Whereas, SFFA-X in Figure 4 will not be divided into 2 groups because all values of $LSG_4(n)$ in this experiment is 0003. However, the size of two prime factors of n is same. That means only 4 algorithms of SFFA-X in which X is equal to 3, 03, 003 and 0003 are chosen to implement. The experiment shows that SFFA-0003 is the fastest algorithm. Nevertheless, the average computation time of FFA-2, SFFA-3, SFFA-03, SFFA-003, SFFA-0003, MFFV4 and P^{2}MFF are about 26.4, 9.35, 3.38, 3.2, 1.16, 4.78 and 13.08 seconds respectively. Furthermore, the information in this figure implies that SFFA-X begins to factor n faster than MFFV4 when X which is equal to 03 is chosen.

The experiment in Figure 2 is similar to the other in Figure 1 but LSG$_4(n)$ in Figure 2 is equal to 0003. Therefore, all algorithms of SFFA-X for this experiment are SFFA-3, SFFA-03, SFFA-003 and SFFA-0003. The experimental in Figure 2 shows that SFFA-0003 is the fastest integer factorization algorithm. Nevertheless, the average computation time of FFA-2, SFFA-3, SFFA-03, SFFA-003, SFFA-0003, MFFV4 and P^{2}MFF are about 270.17, 109.7, 49.38, 43.56, 26.5, 167.25 and 448.28 seconds respectively. Furthermore, this figure implies that SFFA-X begins to factor n faster than MFFV4 when X which is equal to 3 is chosen.

Moreover, if we consider the information in Figure 3 and Figure 4, the average computation time of SFFA-0001 and SFFA-0003 are faster than MFFV4 by about 68.56% and 75.75%, respectively. On the other hand, if the information in Figure 1 and Figure 2 is considered, the average computation time of SFFA-0001 and SFFA-0003 becomes faster than MFFV4 by about 95.28% and 84.15%, respectively.

These results imply that SFFA-X is the better choice to factor n when compared with MFFV4 especially when bits size of X is large and the size of two large prime factors is different. The reason is as follows.

Due to MFFV4 and SFFA-X are modified chronologically from FFA-1 and FFA-2, we will compare the iterations of the computation between FFA-1 and FFA-2 instead of their improvements.

The total iterations in the loop of u, in FFA-2, are $(p+q) - 2\lceil\sqrt{n}\rceil$. However, the increment value of u is always 2, but the increment value of x, in FFA-1, is always 1. That means the total iterations of u are equal to the total iterations of x which are equal to $(p+q)/2 - \lceil\sqrt{n}\rceil$. Nevertheless, the total iterations in the loop of v, in FFA-2, are $p-q$ and are more than the total iterations in the loop of y, in FFA-1, the reason are as follows: Assume A and B are represented as the total iterations in the loop of y and v, respectively. Then,

$$\begin{aligned} A &= (p+q) - 2\lceil\sqrt{n}\rceil \\ &\approx (p+q) - 2\sqrt{n} \\ &= (p+q) - 2\sqrt{p}\sqrt{q} \\ &= (\sqrt{p}-\sqrt{q})^2 \\ &= (\sqrt{p}-\sqrt{q})(\sqrt{p}-\sqrt{q}) \qquad (5) \\ B &= p-q \\ &= \sqrt{p}^2 - \sqrt{q}^2 \\ &= (\sqrt{p}-\sqrt{q})(\sqrt{p}+\sqrt{q}) \qquad (6) \end{aligned}$$

From Equations (5) and (6), all possible results are divided into 2 conditions:

Condition 1: (p is close to q)
It is obvious that the iterations of FFA-2 are greater than FFA-1. Therefore, the condition of SFFA-X which is faster than MFFV4 is that the digits of X in this case must be large enough, the digits of X in Figure 3 and Figure 4 must be at least 2.

Condition 2: (p is far from q)

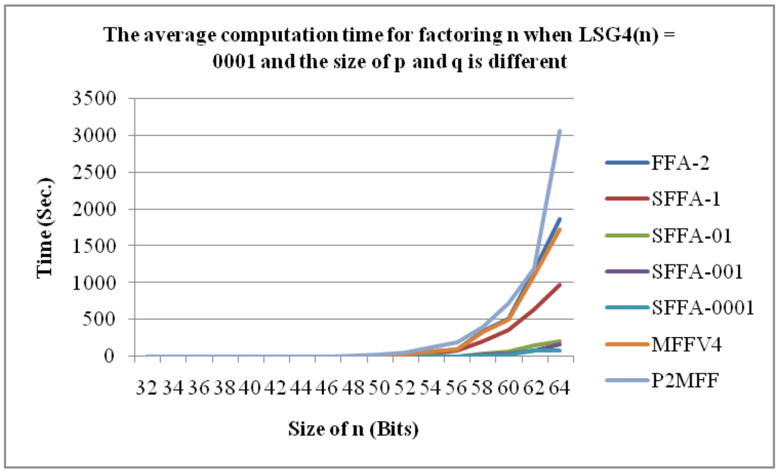

Figure 1: Average computation time for factoring $LSG_4(n) = 0001$ and the size of p and q is different

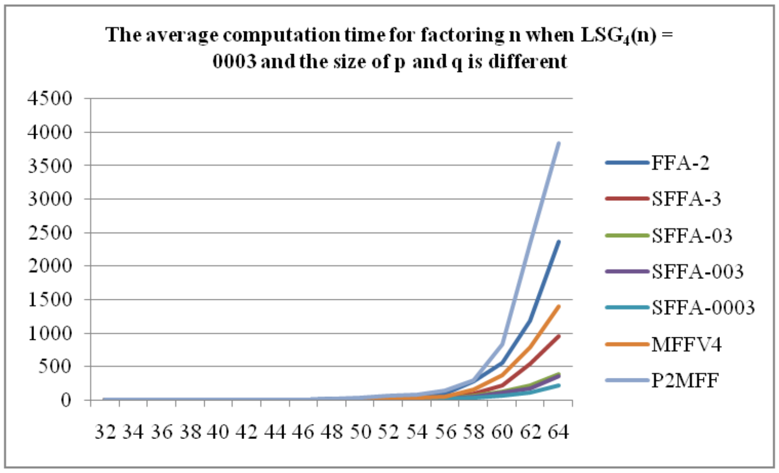

Figure 2: Average computation time for factoring $LSG_4(n) = 0003$ and the size of p and q is different

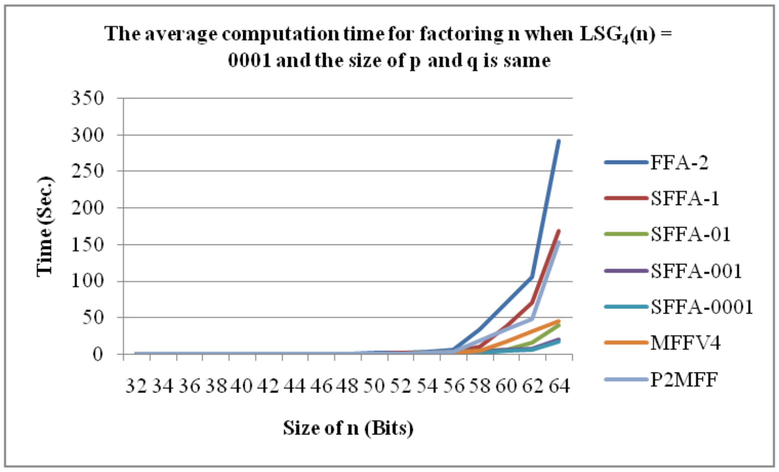

Figure 3: Average computation time for factoring $LSG_4(n) = 0001$ and the size of p and q is same

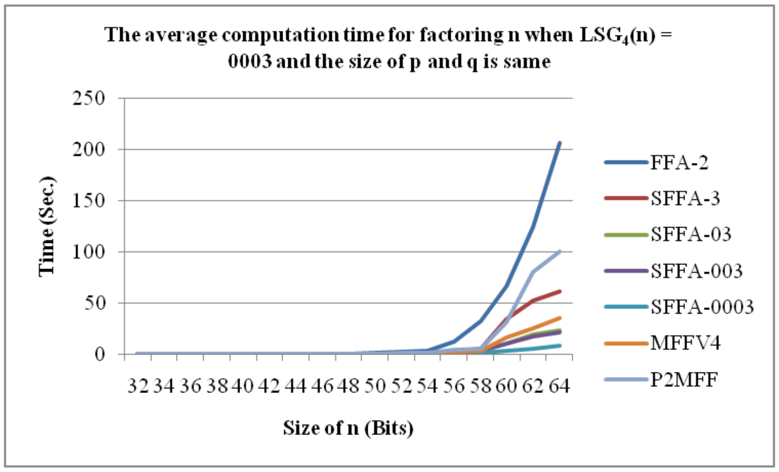

Figure 4: Average computation time for factoring $LSG_4(n) = 0003$ and the size of p and q is same

If p is very larger than q, then q may be left out from the equation to estimate total iterations. That means the iterations of FFA-1 are close to FFA-2. However, FFA-1 is time-consuming to compute the square root of integer while FFA-2 does not to do. This reason indicates that FFA-2 is certainly faster than FFA-1. That means most of SFFA-X can factor n faster than MFFV4 although the size of X is smaller than the other in Condition 1, the digit of X in Figure 1 and Figure 2 is only 1.

From both of two conditions above, we concluded that most of SFFA-X can factor n faster than MFFV4 especially when the size of X is large enough.

Moreover, total iterations of SFFA-X can be found from the following equation:

$$t = (j * \lfloor \frac{A - u_x}{\sum_{i=0}^{j-1} disu(i)} \rfloor + c_u) + (k * \lfloor \frac{B - LSG_m(v_0)}{\sum_{i=0}^{k-1} disv(i)} \rfloor + c_v).$$

Where,

1) t = total iterations of SFFA-X in main loop;
2) $A = (p + q) - 2\lceil\sqrt{n}\rceil$;
3) $B = p - q$;
4) u_x = the increment value of $2\lceil\sqrt{n}\rceil$ to get the initial value of U;
5) j = size of $disu$;
6) k = size of $disv$;
7) $LSG_m(v_0)$ = the minimum value of $LSG_m(v)$;
8) c_u is the remainder iterations that are more than 1 but less than size of $disu$ when there is the remainder of $\frac{A-u_x}{\sum_{i=0}^{j-1} disu(i)}$. However, c_u is equal to 0 when there is not the remainder;
9) c_v is the remainder iterations that are more than 1 but less than size of $disv$ when there is the remainder of $\frac{B-LSG_m(v_0)}{\sum_{i=0}^{k-1} disv(i)}$. However, c_v is equal to 0 when there is not the remainder.

However c_u and c_v can be found by using Algorithm 6 and Algorithm 7 respectively.

Algorithm 6 Calculating c_u
```
1: Begin
2: $size\_disu$ = size of $disu$ that the repeated patterns are removed
3: $count\_u$ = the index of $LSG_m(u)$ that is equal to the initial value of $LSG_m(U)$
4: $count\_u = count\_u \% size\_disu$
5: $s\_u = \sum_{i=0}^{j-1} disu(i)$
6: $r\_u = (A - u_x) \% s\_u$
7: while $r\_u$ is not equal to zero do
8:     $r\_u = r\_u - disu(count\_u)$
9:     $count\_u = count\_u + 1$
10:    if $count\_u$ is equal to $size\_disu$ then
11:        $count\_u = 0$
12:    end if
13: end while
14: $c_u = count\_u$
15: End
```

In addition, for some value of n that SFFA-X must be divided into 2 group, t is computed by using only all parameters in the solution group.

Example 6. *Find total iterations in Example 5.*

Because the solution group is in Group 2, we have to use parameters in this group as follows:

1) $A = (457 + 293) - 732 = 18$;
2) $B = 457 - 293 = 164$;
3) $j = 1$, *size of* $disu$;

Algorithm 7 Calculating c_v

```
1: Begin
2: $count\_v = 0$
3: $s\_v = \sum_{i=0}^{k-1} disv(i)$
4: $r\_v = (B - \text{LSG}_m(v_0))\%s\_v$
5: while $r\_v$ is not equal to zero do
6:     $r\_v = r\_v - disv(count\_v)$
7:     $count\_v = count\_v + 1$
8: end while
9: $c_v = count\_v$
10: End
```

4) $\sum_{i=0}^{0} disu(i) = disu = 20$*;*

5) $k = 2$*, size of disv;*

6) $\sum_{i=0}^{1} disv(i) = disv(0) + disv(1) = 28 + 72 = 100$*;*

7) $u_x = 18$*, the initial value of U is 750, then* $U - 2\lceil\sqrt{n}\rceil = 750 - 732 = 18$*;*

8) $LSG_2(v_0) = 36$*;*

9) $\frac{A - u_x}{\sum_{i=0}^{j-1} disu(i)} = \frac{18-18}{20} = 0$*, the remainder is 0, then* $c_u = 0$*;*

10) $\frac{B - LSG_2(v_0)}{\sum_{i=0}^{k-1} disu(i)} = \frac{164-36}{100} = 1$*, the remainder is 28 (*$r\_v = 28$*), then* c_v *can be computed by using Algorithm 7 as follows:*

a. $count\_v = 0, s\_v = 100, r\_v = 28$*;*

b. $r\_v = r\_v - disv(0) = 28 - 28 = 0$*;*

c. $count\_v = count\_v + 1 = 1$.

Because $r\_v = 0$*, then* $c_v = count\_v = 1$. *Therefore, total iterations in main loop is*

$$t = ((1)(0) + 0) + ((2)(1) + 1) = 3.$$

In general, both of $\frac{j}{\sum_{i=0}^{j-1} disu(i)}$ and $\frac{k}{\sum_{i=0}^{k-1}(i)}$ are always less than $\frac{1}{2}$. Therefore, t is always less than total iterations of FFA-2 that is equal to $\frac{A+B}{2}$. Furthermore, t can be decreased when the value of m is larger because the values of $\frac{j}{\sum_{i=0}^{j-1} disu(i)}$ and $\frac{k}{\sum_{i=0}^{k-1}(i)}$ are certainly smaller.

5 Conclusion

The aim of this paper is to propose a new technique to speed up the 2^{nd} method of Fermat's Factorization Algorithm (FFA-2) by considering the last m digits of n to leave out some values of u and v which are not in the condition. This technique is called Specific Fermat's Factorization Algorithm Considered from X (SFFA-X) where X is the last m digits of n. Furthermore, the concept of this technique implies that the computation time of SFFA-X will be reduced more whenever the bigger size of m is considered because more details of u and v are known. Therefore, the unrelated values of u and v are increased and they should be left out from the computation.

Moreover, SFFA-X can be also applied with Estimated Prime Factor (EPF) [18] using the technique of continued fractions [5, 12] to estimate the new initial values of U and V to reduce more iterations of the computation. However, the applying SFFA-X with EPF can be used to factor only the unbalanced modulus in order to get the high accuracy.

References

[1] A. R. Ambedkar, A. Gupta, P. Gautam, and S. S. Bedi, "An efficient method to factorize the RSA public key encryption," in *Proceedings of Communication Systems and Network Technologies (CSNT'11)*, pp. 108–111, Jummu, Katra, June 2011.

[2] D. Bishop, *Introduction to Cryptography with Java Applets*, Germany: Jones and Bartlett Publisher, 2003.

[3] D. M. Bressoud, *Factorization and Primality Testing*, United states: Springer-Verlag, 2013.

[4] Q. Huang, Y. T. Li, Y. Zhang, and C. Lu, "A modified non-sieving quadratic sieve for factoring simple blur integers," in *Proceedings of Multimedia and Ubiquitous Engineering (MUE'07)*, pp. 729–732, Seoul, South Korea, Apr. 2007.

[5] N. Kobiltz, *A Course in number theory and cryptography*, United states: Springer, 1994.

[6] J. McKee, "Speeding fermat' factoring method," *Mathematics of Computation*, vol. 68, pp. 1729–1738, 1999.

[7] N. A. Moldovyan, "Short signatures from difficulty of factorization problem," *International Journal of Network Security*, vol. 8, no. 1, pp. 90–95, 2009.

[8] J. Pollard, "Monte carlo methods for index computation (mod p)," *Mathematics of Computation*, vol. 32, no. 7, pp. 918–924, 1978.

[9] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, United states: Birkhuser Boston, 1994.

[10] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communications of ACM*, vol. 21, no. 7, pp. 120–126, 1978.

[11] P. Sharma, A. Gupta, and A. Vijay, "Modified integer factorization algorithm using v-factor method," in *Proceedings of Advanced Computing & Communication Technologies (ACCT'12)*, pp. 423–425, Haryana, India, Jan. 2012.

[12] J. H. Silverman, *A friendly introduction to number theory*, United states: Pearson Education International, 2012.

[13] K. Somsuk, "A new modified integer factorization algorithm using integer modulo 20's technique," in *Proceedings of International Computer Science and Engineering Conference (ICSEC'14)*, pp. 312–316, Khon Kaen, Thailand, July 2014.

[14] K. Somsuk and S. Kasemvilas, "Mffv2 and mnqsv2: Improved factorization algorithms," in *Proceedings of International Conference on Information Science and Applications (ICISA'13)*, pp. 1–3, Pattaya, Thailand, June 2013.

[15] K. Somsuk and S. Kasemvilas, "Mvfactor: A method to decrease processing time for factorization algorithm," in *Proceedings of International Computer Science and Engineering Conference (ICSEC'13)*, pp. 339–342, Bangkok, Thailand, Sept. 2013.

[16] K. Somsuk and S. Kasemvilas, "Mffv3: An improved integer factorization algorithm to increase computation speed," in *Proceedings of The KKU International Conference (KKU-IENC'14)*, pp. 1432–1436, Khon Kaen, Thailand, Mar. 2014.

[17] K. Somsuk and S. Kasemvilas, "Possible prime modified fermat factorization: New improved integer factorization to decrease computation time for breaking rsa," in *Proceedings of International Conference on Computing and Information Technology (IC2IT'14)*, pp. 325–334, Phuket, Thailand, May 2014.

[18] 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, pp. 162–168, 2014.

[19] J. Zalaket and J. H. Boutros, "Prime factorization using square root approximation," *Computers and Mathematics with Applications*, vol. 61, no. 9, pp. 2463–2467, 2011.

Kritsanapong Somsuk is a Ph.D. student of the department of Computer Engineering in Faculty of Engineering, Khon Kaen University, Khon Kaen, Thailand. He received M.Eng. (Computer Engineering) degree from Khon Kaen University in 2009 and M.Sc. (Computer Science) degree from Khon Kaen University in 2013. His research interests are cryptography and integer factorization algorithms.

Kitt Tientanopajai is a lecturer of the department of Computer Engineering in Faculty of Engineering, Khon Kaen University, Khon Kaen, Thailand. He received his M.Eng. (Computer Science and Information Management) degree from Asian Institute of Technology in 1998 and his D.Eng. (Computer Science and Information Management) degree from Asian Institute of Technology in 2005. His research interests include Free/Open Source Software, Information Security, Quality of Service Routing, Computer Networks and Educational Technology.