KKU ENGINEERING JOURNAL 2016;43(S2):350-353 Research Article

KKU Engineering Journal

https://www.tci-thaijo.org/index.php/kkuenj/index

Improving fermat factorization algorithm by dividing modulus into three forms

Kritsanapong Somsuk and Kitt Tientanopajai*

Department of Computer Engineering, Faculty of Engineering, Khon Kaen University, Khon Kaen 40002, Thailand.

Received April 2016
Accepted June 2016

Abstract

Integer Factorization (IF) becomes an important issue since RSA which is the public key cryptosystem was occurred, because IF is one of the techniques for breaking RSA. Fermat's Factorization Algorithm (FFA) is one of integer factorization algorithms that can factor all values of modulus. In general, FFA can factor the modulus very fast in case that both of prime factors are very close. Although many factorization algorithms improved from FFA were proposed, it is still time – consuming to find the prime factors. The aim of this paper is to present a new improvement of FFA in order to reduce the computation time to factor the modulus by removing some iterations of the computation. In fact, the key of the proposed algorithm is the combination within the three techniques to check the forms of the modulus before making decision to leave some integers out from the computation. In addition, the proposed algorithm is called Multi Forms of Modulus for Fermat Factorization Algorithm (Mn-FFA). The experimental results show that Mn-FFA can reduce the iterations of computation for all values of the modulus when it is compared with FFA and the other improved algorithms.

Keywords: Fermat's Factorization Algorithm (FFA), Modulus, Prime number, Multi Forms of Modulus for Fermat Factorization Algorithm (Mn-FFA)

1. Introduction

RSA algorithm [1] is a public key cryptosystem used to secure the information, which is sent through the insecure channel, by using the different processes between encryption and decryption. This algorithm uses a pair of keys, public key and private key, to encrypt the message and decrypt the encrypted message. In fact, public key is always disclosed to everyone but private key is kept secret by key generator. However, attackers can recover the private key to break RSA whenever the modulus which is equal to the product of two prime numbers is factored. Therefore, the size of the modulus should be larger than 1024 bits [2] to make RSA securely. At present, there are many integer factorization algorithms proposed for attacking RSA. In general, each efficient algorithm is based on the size of the modulus and also the size of two prime factors of the modulus. For example, if the size of one out of two prime factors of the modulus is very small, Trial Division Algorithm (TDA) can factor the modulus very fast, whereas another prime factor is very large. According to this algorithm, the first prime divisor is 3 and it will be increased larger until the real prime factor of the modulus is found. Nevertheless, TDA becomes very slow whenever it is used to factor the modulus that both of two prime factors are very large. However, if the difference between two large primes is very small, Fermat's factorization algorithm (FFA) [3-4] is one of the efficient factorization algorithms for this situation. FFA is a brute force attack method to find two prime factors by rewriting the modulus as the difference of two perfect squares equation. Although many factorization algorithms improved from FFA were proposed to decrease computation time by reducing some processes or iterations of the computation such as [5-9], it is still time – consuming to find two prime factors.

In this paper, time-reducing of FFA is proposed by leaving some integers out from the computation to find two prime factors. The key is from the combination within three techniques [7-9] proposed to remove some steps of FFA to choose only integers in the conditions. The new improvement of Fermat's factorization is called "Multi Forms of Modulus for Fermat Factorization Algorithm (Mn-FFA)". For Mn-FFA, the chosen group of integers is small when it is compared with the others improved from FFA. Therefore, the computation time for factoring the modulus is certainly decreased.

2. Related work

2.1 Fermat's Factorization Algorithm: FFA

FFA is the integer factorization algorithm discovered by Pierre de Fermat. This algorithm can factor the modulus very fast when the difference between two large primes is very small. In general, FFA will rewrite the modulus as the difference of two perfect squares as follows:

$$n = x^2 - y^2$$

Therefore, $$y^2 = x^2 - n$$

*Corresponding author. Tel.: +6696 945 6624
Email address: tonpor007@hotmail.com; kitt@kku.ac.th*
doi: 10.14456/kkuenj.2016.127

Where $x = (p + q)/2$, $y = (p – q)/2$, p and q are primes and n is the modulus

In addition, the initial value of x starts at $\lceil\sqrt{n}\rceil$ and it is always increased by 1 to find the integer of $y = \sqrt{x^2 - n}$ whenever y is still not an integer. Nevertheless, the process will be stopped when the integer of y is found and it implies that $x + y$ and $x – y$ are two large prime factors of n. Furthermore, many factorization algorithms improved from FFA were proposed to decrease computation time as follows:

2.2 Modified Fermat Factorization Version 2: MFFV2

MFFV2 [5] does not compute the square root of y^2 when the least significant digit of this integer is 2, 3, 7 or 8 because it is not a perfect square. From this reason, the square root result is not certainly an integer when the least significant digit of y^2 is 2, 3, 7 or 8. Therefore, the computation time of MFFV2 is decreased when it is compared with FFA.

2.3 Modified Fermat Factorization Version 3: MFFV3

The key of MFFV3 [6] uses the table called Difference's Least Significant Digit Table (DLSDT) for analyzing the least significant digit of y^2 without computing all digits of this integer directly. Therefore, if the result of the least significant digit of y^2 analyzed in DLSDT is 2, 3, 7 or 8, it is not time-consuming to compute y^2 and its square root. In general, MFFV3 is faster than MFFV2 because MFFV2 must compute y^2 for all iterations.

2.4 Modified Fermat Factorization Version 4: MFFV4

MFFV4 [7] uses Y2MOD20 which is the improved table of DLSDT for analyzing the result of y^2 % 20 without computing all digits of y^2 directly. In general, y^2 will be computed only if the result of y^2 % 20 in Y2MOD20 is 0, 1, 4, 5, 9 or 16. This reason implies that total iterations to compute y^2 by using MFFV4 are always less than total iterations to compute y^2 by using MFFV3.

Table 1 The results of y^2 modulo 20 (Y2MOD20)

LSG(*x*)	***n* modulo 20 (*n* % 20)**							
	1	**3**	**7**	**9**	**11**	**13**	**17**	**19**
0	19	17	13	11	9	7	3	1
1	0	18	14	12	10	8	4	2
2	3	1	17	15	13	11	7	5
3	8	6	2	0	18	16	12	10
4	15	13	9	7	5	3	19	17
5	4	2	18	16	14	12	8	6
6	15	13	9	7	5	3	19	17
7	8	6	2	0	18	16	12	10
8	3	1	17	15	13	11	7	5
9	0	18	14	12	10	8	4	2

From Table 1, LSG(x) is represented as the least significant digit of x. In fact, the result of y^2 % 20 can be considered from the relation between LSG(x) and the result of n % 20. For example, if LSG(x) is 0 and the result of n % 20 is 1, then Y2MOD20 shows that the result of y^2 % 20 is always equal to 19. Therefore, the results in Y2MOD20 can be used for making decision to compute y^2.

2.5 Xiang's method

In 2004, Xiang proposed the improvement of FFA [9] by considering n as the form $4k$. He found that there were only two forms of n as follows:

Form 1 ($n = 4k - 1$): x is always an even number.
Form 2 ($n = 4k + 1$): x is always an odd number.

Therefore, the computation time is decreased because the values of x can be increased by 2 for the both forms. Although the form of n must be checked for choosing the pattern of x, this process is always finished at the first step out of loop of the computation.

2.6 Possible Prime Modified Fermat Factorization: P²MFF

In 2014, P²MFF [8] was proposed by considering n as the form $6k$. We know that when all prime numbers are rewritten as the form $6k$, they are only $6k – 1$ or $6k + 1$ [8]. However, there are only two prime numbers that cannot be rewritten as these forms. First is $2 = 6(0) + 2 = 6k + 2$ and the second is $3 = 6(0) + 3 = 6k + 3$. Therefore, if both of them are left out from the decision, n can be divided as only two forms as follows:

Form 1 ($n = 6k - 1$): x is always divided by 3.
Form 2 ($n = 6k + 1$): x is not always divided by 3.

In fact, the key of P²MFF is to check the form of n to find the pattern and the increment value of x. Moreover, the technique of MFFV2 is also chosen to implement with P²MFF to decrease more iterations.

3. The Proposed method

In this paper, the hybrid forms of n are proposed to leave out some values of x which are not the expected values to find p and q. The key is that, n will be transformed as both of the forms $4a$ and $6b$, a and b are any integers, to choose only some values of x in the conditions of the both forms. Therefore, all iterations of the computation by using the hybrid forms are certainly less than all iterations of the computation by using single form. Due to, both of the mentioned forms are divided into two cases, the hybrid forms must be divided into four cases as follows:

Case 1: the forms of n are $4a – 1$ and $6b - 1$
x has to be an even number and it also must be divided by 3.
Case 2: the forms of n are $4a + 1$ and $6b – 1$
x has to be an odd number and it also must be divided by 3.
Case 3: the forms of n are $4a - 1$ and $6b + 1$
x has to be an even number but it must not be divided by 3.
Case 4: the forms of n are $4a + 1$ and $6b + 1$
x has to be an odd number but it must not be divided by 3.

Furthermore, Y2MOD20 is also used with the proposed method for decreasing other iterations of the computation, because only y^2 that y^2 % 20 is 0, 1, 4, 5, 9 or 16 will be chosen to compute the square root. In fact, the proposed method combined with Y2MOD20 is called Multi Forms of Modulus for Fermat Factorization Algorithm (Mn-FFA). In addition, the algorithm of Mn-FFA is as follows:

Algorithm: Mn-FFA
Input: n
Step1: $a = n$ % 4
Step2: $b = n$ % 6
Step3: $c = n$ % 20
Step4: Find the pattern of x based on the value of a, b and c, in Table 2
Step5: $x_t = \lceil\sqrt{n}\rceil$
Step6: While x_t is not the pattern of x do
Step7: $x_t = x_t + 1$
Step8: EndWhile
Step9: $y = \sqrt{x_t^2 - n}$
Step10: While y is not an integer do

Table 2 The pattern of x from the combination within three techniques

Case	a	b	c	Pattern of x
1	1	1	1	Odd number which is not divided by 3 and the last digit is 1, 5 or 9
2	1	1	9	Odd number which is not divided by 3 and the last digit is 3, 5 or 7
3	1	1	13	Odd number which is not divided by 3 and the last digit is 3 or 7
4	1	1	17	Odd number which is not divided by 3 and the last digit is 1 or 9
5	1	5	1	Odd number divided by 3 and the last digit is 1, 5 or 9
6	1	5	9	Odd number divided by 3 and the last digit is 3, 5 or 7
7	1	5	13	Odd number divided by 3 and the last digit is 3 or 7
8	1	5	17	Odd number divided by 3 and the last digit is 1 or 9
9	3	1	3	Even number which is not divided by 3 and the last digit is 2 or 8
10	3	1	7	Even number which is not divided by 3 and the last digit is 4 or 6
11	3	1	11	Even number which is not divided by 3 and the last digit is 0, 4 or 6
12	3	1	19	Even number which is not divided by 3 and the last digit is 0, 2 or 8
13	3	5	3	Even number divided by 3 and the last digit is 2 or 8
14	3	5	7	Even number divided by 3 and the last digit is 4 or 6
15	3	5	11	Even number divided by 3 and the last digit is 0, 4 or 6
16	3	5	19	Even number divided by 3 and the last digit is 0, 2 or 8

Table 3 The computation time of P^{2}MFF, MFFV4 and Mn-FFA

Modulus (n)	Form of n			Time (Seconds)		
	a	b	c	P^{2}MFF	MFFV4	Mn-FFA
13579987811093858723	3	5	3	13.96	21.9	**7.4**
9190349761567687607	3	5	7	10.39	16.8	**5.7**
12304702995418616831	3	5	11	4.21	7.7	**2.6**
11160812689425299939	3	5	19	45.3	117.5	**29**
11613506411700676661	1	5	1	16.66	80.68	**10.5**
9689684844042557429	1	5	9	19.57	96.12	**12.2**
9862358417775834713	1	5	13	3.13	12.7	**1.6**
7879325961791987357	1	5	17	15.81	64.3	**8.4**
10971944694657500683	3	1	3	64.26	52.55	**34.1**
6671971776553728907	3	1	7	26.81	21.03	**14.2**
11277658827780482311	3	1	11	33.79	31.15	**20.5**
6995000962812320239	3	1	19	75.07	71.5	**18.3**
9468901990375452721	1	1	1	39.43	36.23	**24**
9958622145744545569	1	1	9	7.68	7.07	**4.9**
10694303699556447013	1	1	13	9.79	8.05	**5.3**
8001373904877499297	1	1	17	49	38.9	**27.1**
Average				27.18	42.76	**14.11**

Step11: Increase x_t, the increment value of x_t is based on the pattern of x in Step4

Step12: $y = \sqrt{x_t^2 - n}$

Step13: EndWhile

Step14: $p = x_t + y$

Step15: $q = x_t - y$

Output: p, q

In general, for the combination within three techniques, n is divided into thirty two cases because there are two cases of a, two cases of b and eight cases of c. However, some cases are impossible to occur. For example, if the results of a and c are 3 and 1 respectively, the value of x must be an even number and LSG(x) must be 1, 5 or 9 that is the contradiction. In fact, there are only sixteen cases of n that can be occurred. Table 2 shows all possible cases of n to generate the pattern of x.

Example 1: Factoring n = 337123 by using Mn-FFA

Step1: a = 337123 % 4 = 3 ($4k$ + 3 or $4k$ - 1)

Step2: b = 337123 % 6 = 1 ($6k$ + 1)

Step3: c = 337123 % 20 = 3

Step4: Finding the pattern of x

Because of the results of a = 3, b = 1 and c = 3, the pattern of x must be in case 9. That means the value of x must not be divided by 3 and it is always an even number. Furthermore, LSG(x) must be only 2 or 8.

Step 5 - 8:

$x_t = \lceil \sqrt{337123} \rceil = 581$

However, the value of x_t is not in the conditions of case 9. Therefore, x_t has to be increased until the new value in the conditions is found. In fact, the minimum value of x_t which is larger than 581 and is in the conditions is 592.

Step 9:

$y = \sqrt{592^2 - 337123} = 115.5$

Step 10 – 13: Process in Loop

Iteration 1: x_t = 598

$y = \sqrt{598^2 - 337123} = 143.11$

Iteration 2: x_t = 602

$y = \sqrt{602^2 - 337123} = 159$

Step 14: p = 602 + 159 = 761

Step 15: q = 602 – 159 = 443

Therefore, 443 and 761 are two prime factors of 337123.

According to the example, it implies that a lot of x_t not in the conditions can be left out from the computation when it is compared with other algorithms choosing only one out of three forms for the computation.

In fact, Mn-FFA is always faster than P^{2}MFF and MFFV4 for all values of n. The reason is that only one technique is chosen for P^{2}MFF or MFFV4 to decrease

computation time, the technique about checking n % 6 for P^{2}MFF and the technique about checking n % 20 for MFFV4, but Mn-FFA is from the combination within three techniques which one out of three techniques is chosen for P^{2}MFF or MFFV4.

4. Results and discussion

Table 3 shows all possible cases of 64 bits size of n chosen randomly. In addition, all algorithms in Table 3 are implemented by using Biginteger Class of java programming language, because this class can be represented of the data type which is unlimited in size. The experimental results show that Mn-FFA can factor n faster than MFFV4 and P^{2}MFF. Furthermore, the average computation time of P^{2}MFF, MFFV4 and Mn-FFA are about 27.18, 42.76 and 14.11 seconds respectively.

Nevertheless, the disadvantage of Mn-FFA is that the algorithm cannot finish factoring n when one out of two prime factors is equal to 2, 3 or 5. There are two reasons. First is that 2 and 3 cannot be rewritten as the form $6k - 1$ or $6k + 1$. Second is that LSG(n) = 5 is not considered in Y2MOD20. However, it is impossible that one out of two prime factors is 2 or 5 because a prime factor is known suddenly when LSG(n) is 2 or 5. Moreover, if the form of n is $6k + 3$, then 3 is certainly a factor of n. Therefore, if one of the mentioned prime numbers is chosen as a prime factor of n, it can be found before using Mn-FFA.

5. Conclusion

The aim of this paper is to propose Mn-FFA by combining the results of n%4, n%6 and n%20 together. In fact, if these techniques are considered together, many values of x that not include in the conditions are left out from the computation. Therefore, the computation time is reduced. The experimental results show that Mn-FFA can factor n faster than P^{2}MFF and MFFV4. However, the disadvantage of Mn-FFA is that it cannot factor n when a prime factor is 2, 3 or 5. However, these values are not suitable to be chosen as a prime factor because of low security.

6. References

[1] Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public key cryptosystems. Communications of ACM 1978; 21:120-126.

[2] Geiselmann W, Steinwandt R. Special-Purpose Hardware in Cryptanalysis: The Case of 1,024-Bit RSA. IEEE Security & Privacy 2007;5:63-66.

[3] Bishop D. Introduction to Cryptography with java Applets. United States: Jones and Bartlett Publisher; 2003.

[4] Wu ME, Tso R, Sun HM. On the improvement of Fermat factorization using a continued fraction technique. Future Generation Computer Systems 2014; 30:162-168.

[5] Somsuk K, Kasemvilas S. MFFV2 And MNQSV2: Improved Factorization Algorithms. Proceedings of the 4th International Conference on Information Science and Applications; 2013 Jun 24-26; Pattaya, Thailand: New Jersey: IEEE; 2013.

[6] Somsuk K, Kasemvilas S. MFFV3: An Improved Integer Factorization Algorithm to Increase Computation Speed. Proceedings of the 3rd The KKU International Conference; 2014 Mar 27-28; KhonKaen, Thailand: Germany: AMR; 2014.

[7] Somsuk K. A New Modified Integer Factorization Algorithm Using Integer Modulo 20's Technique. Proceedings of the 18th International Computer Science and Engineering Conference; 2014 Jul 30-Aug 1; KhonKaen, Thailand: New Jersey: IEEE; 2014.

[8] Somsuk K, Kasemvilas S. Possible Prime Modified Factorization: New Improved Integer Factorization to Decrease Computation Time for Breaking RSA. Proceedings of the 10th International Conference on Computing and Information Technology; 2014 May 6-8; Phuket, Thailand: Springer; 2014.

[9] Xiang G. Fermat's Method of Factorization. Applied Probability Trust 2004;36:34-35.