A Fast Factorization Method

Author: Alessandro Boatto
Italy 05/06/2018

Abstract

Integer factorization is one of the mathematical aspects which are largely applied in our daily activities. One of the situations where factoring is applied is in cryptography which is the treasured portion of ecommerce systems. Due improved encryption currently banking institutions, business firms, government institutions, online merchants and many other groups and individuals are using e-commerce platform to send and receive confidential information such as transferring money. Cryptography enhances the integrity and authenticity of the data and communication between legit by assigning specific private keys to participants. The keys are generated by factorizing a large number, and they can only be decrypted by the assigned individuals only. The number under quest is large enough such that finding their factors which are the keys takes a lot of time and becomes costly than the information encrypted by those keys especially when using the old Trial Division method. Therefore it has been a concern of researchers to come up with a better method of finding the Keys.

Keywords: RSA Algorithm, RSA Key Cryptography, Fast Factorization Method, RSA Attacks.

Introduction

Key cryptography which is based on integer factorization is the most applied type of cryptography in the e-commerce systems for ensuring integrity, confidentiality, and authenticity of the information shared via those systems. There are many algorithms available which can be used to generate the keys, but in this discussion, we are going to consider the RSA algorithm in generating cryptography keys [1]. In RSA algorithm there two are two large prime numbers ‘a’ and ‘b’, which are factors of a larger number, n. Let the e-commerce system have person A and person B, according to key cryptography everyone is required to have unique keys assigned to them. Each participant is required to have a public and a private key. A
public key will allow the participants to accessed by whoever wants to reach them out whereas a private key ensures that it is only the intended recipient who has legit access to the information shared. Let person A have $<x, n>$ as his public key and $<y, n>$ as his private key. We have $n$ being a large selected number know to the public with two prime factors ‘a’ and ‘b’ (note that with the two prime factors $xy = 1 \mod (a-1) (b-1)$). This indicates that the security of the private key $<y, n>$ relies on the difficulty of finding the two prime factors of an integer $n$. So in the case where person B wants to communicate to person A, he would need access to $<x, n>$ to send him an encrypted message. Person B will determine $M' = \text{mod n}$ indicated as , thus encrypting message M which can only be decrypted by person A using the specified key. Upon reception of the message person A, will be required to determine $M''$ for him to access the contents of the sent message. In determining $M''$ person A, decrypts the secured message $M$, $M'' = = M \mod n$. Ideally the only person who is in a position to decrypt message $M$ is person A because he is the only one who acknowledges $y$. Since the message to be decrypted it requires $y$ which is only know to person A, and then the message is considered to be secure. Therefore, for the attacker to access the encrypted message he/she needs $x$ and $y$, $x$ is available as the public key of person A but factorization of $n$ should be performed to find $y$ [2]. The security of encrypted messages and the RSA keys depend on the complexity of factoring. For the purpose of intensifying the security of RSA, integer $n$ to be selected should be too large and complex to factor such that the cost of decrypting unauthorized data becomes expensive to scare attackers away [3].

Unauthorized decryption of information

Despite the RSA key cryptography is one of the preferred secure key algorithms attackers have developed more sophisticated methods of getting into systems secure by RSA. Although the attackers being able to work with approaches such as Timing attacks, mathematical attacks, Chosen Ciphertext attacks, and Brute force attacks, there has been a need of coming up with a faster and a less costly approach. Timing attacks work by approximating the time the RSA key cryptography uses to generate the private encrypted key and uses that time to determine $y$. On the hand, mathematical attacks deploy different tactics based on the structures of the RSA cryptography to factor $n$ with the objective of decrypting the encrypted messages. Where Brute Force attack is based on trials, all possible keys are tried to see if they can match that of the encrypted message. The Brute Force attack and others which have been in use still use a lot of time to decrypt information [4]. Concerning the quest of having a faster and less costly approach the Fast Factorization Method was developed and has proven to serve the purpose.

Timing attacks

As mentioned above, the timing attacks are time approximation based approaches which can be used to attack RSA key cryptography messages. During encryption, the RSA algorithm computes $M' = \text{mod n}$ which is the security key for the data to be transmitted. Timing attacks estimate the time taken to compute $M'$ and uses that time to determine $y$ and uses it to decrypt the message.

Mathematical attacks

The RSA key algorithm bases its security on the difficulty of finding the two prime factor of an integer $n$. Therefore, if anyone comes up with a way of determining those factors (‘a’ and ‘b’) then he/she has access to whatever data which is encrypted with that algorithm. Mathematical attacks deploy different approaches based on the RSA key functionality with the aim of factoring $n$ into ‘a’ and ‘b’. After determining the prime factor, it would be easy to find $y$ and $M''$ hence decrypting the message [5].
**Brute Force attacks**

As indicated above, Brute Force attacks are based on trial mechanism; the attackers attempt to decrypt given information by the use of all possible combination of private and public keys. Based on the trial and error approach, picking the correct key is possible with small sized numbers, but it becomes difficult to guess a key if the number under quest is very large. Therefore large integers are recommended in the RSA algorithms to outwit Brute Force attackers. On the other hand, very large integers are not efficient as the algorithm will require a lot of time for internal computations thus inconveniencing the transmission and reception of information therefore not a right approach of scaring brute force attackers away. For that reason, integer \( n \) is kept reasonable but then attackers bewildered from the data [6].

**Factorization Method**

The integer \( n \) in the RSA key cryptography is a large size number such that the other approaches which have been in use before have not been able to factor it fast enough within the utility time. The two large prime factors \( x \) and \( y \) are the major concern of an attacker [7]. From the above, the information sent to person A is encrypted in \( y \), \( x \), and \( n \) are public therefore the task is to find \( y \) which is only possible in determining the factors of \( n \). The other existing approaches can determine the factors ‘\( a \)’ and ‘\( b \)’ but then the time it takes is the only major drawback. We are looking at factorization methods which is less costly and which can realize the factors fast enough when the information is still valuable [8].

**Trial Division**

Trial division approach is the simplest to apply in determining the prime factor of an integer \( n \), in this case, \( n \) is subjected to division by small prime integers. The small prime number applicable starts with 2, 3, 5, 7 so on up to any reasonable small prime integer. The division process is continuously conducted up to a point it cannot further be divisible. A remainder of zero at the end of the division indicates that \( n \) is divisible by the prime number used. The process aims at determining a prime number which is equal to or close to but not more than the square root of \( n \).

**Example:**

In determining the factors of 91, we can use small prime numbers below it such as 2, 3, 5, 7. Remember we can’t try with a prime beyond seven because we need a prime number which is close or equal but not more than its square root. When 91 which is our \( n \), in this case, is divided by 7 we get a reminder of 0 implying we got one of our factors at that point. Since \( a \times b \) should be equal to 91, then our factors are 7 and 13 [9].

Compared to other factorization methods available, Trial Division is the simplest and easiest to follow. It is efficient in determining prime factors of small numbers ranging from 0 to 9 digits numbers. Applying trial division on the number with more than 9 digits can take extremely long trying to guess possible prime factors thus it will be costly and time-consuming.

**Pollard’s P-1**

The Pollard’s \( P-1 \) algorithm uses Fermat’s Little Theorem in determining prime factors of \( n \).

**Pollard’s Rho Method**

Looking at the weakness of Trial Division Method, John M Pollard decided to improve on it. The
improved method enables the user to find the factor of numbers of up to doing the equivalent to factoring integer \( n \) using the trial division method.

**Quadratic Sieve**

Quadratic Sieve factoring algorithm is an improvement on the Dixon’s random square method and the Fermat’s method. The algorithm was developed objectively to work on integers with digits of between 50 and 100. Between the years of 1980’s and 1990’s Quadratic Sieve method was considered the most effective. The time taken to work only depended on the size of integer \( n \). The Quadratic Sieve states that which meant A non-trivial factor of \( n \) was determined by computing \( \text{GCD}((x-y), n) \) and that of \( \text{GCD}((x+y), n) \) [10].

**The Fast Factorization Method**

When dealing with the information coded with the RSA key algorithm you need a versatile and convenient factoring method. We are looking at the factorization method which is less costly and which can realize the factors fast enough when the information is still valuable.

**Methodology**

In finding ‘\( a \)’ and ‘\( b \)’ a fast factorization method is conducted, it includes two sections.

**Section 1**

1. First, you have to know the exact number of digits of one of the two factors.
2. If you do not know you have to try it by trial.
3. Compare the smaller rest of each unit from 9 to 0, and then choose the digit with the smaller rest.
4. Start with a digit from left to right.
5. If the rest is 0, you know the two factors.
6. A and A1 serve to determine the exact digit/unit and the smallest rest.

**Section 2**

1. Every final section 1 at point 3. Compare the smaller rest of each unit from 9 to 0, and then choose the digit with the smaller rest.

Example:

```
-----------------------------Starting First Digit
N = 101*3 = 303

A:
303/999 = 0.3
0.3 * 999 = 299
303-299 = 4
A1:
303/900 = 0.3
0.3 * 900 = 270
```
303-270 = 33

---

A:
303/899 = 0.3
0.3 * 899 = 269
303-269 = 34
A1:
303/800 = 0.3
0.3 * 800 = 240
303-240 = 63

---

A:
303/799 = 0.3
0.3 * 799 = 239
303-239 = 64
A1:
303/700 = 0.3
0.3 * 700 = 280
303-280 = 23

---

A:
303/699 = 0.4
0.4 * 699 = 279
303-279 = 24
A1:
303/600 = 0.5
0.5 * 600 = 300
303-300 = 3

---

A:
303/599 = 0.5
0.5 * 599 = 299
303-299 = 4
A1:
303/500 = 0.6
0.6 * 500 = 300
303-300 = 3

---

A:
303/499 = 0.6
0.6 * 499 = 299
303-299 = 4
A1:
303/400 = 0.7
0.7 * 400 = 280
303-280 = 23

---

A:
303/399 = 0.7
0.7 * 399 = 279
303-279 = 24
A1:
303/300 = 1.01
1 * 300 = 300
303-300 = 3

------------------------
A:
303/299 = 1
1 * 299 = 299
303-299 = 4
A1:
303/200 = 1
1 * 200 = 200
303-200 = 103

------------------------
A:
303/199 = 1
1 * 199 = 199
303-199 = 104
A1:
303/100 = 3.03
3 * 100 = 300
303-300 = 3

------------------------

FOUNDAMENTAL NOTE

There is an important Note called Foundamental Note:

Ok there are four equal (Rest) R=3 at 600, 500, 300, 100 called FIRST because are equal and its are the smallest , and there are three (Rest) R=4 at 999, 599 , 299 called SECOND because are the most near to FIRST .

Take FIRST and divide by (Rest) R=3

600/3 = 200
500/3 = 166,6
300/3 = 100
100/3 = 33,3

Take SECOND and divide by (Rest) R=4

999/4 = 249,7
599/4 =149,7
299/4 =74,7

The smaller result is 33,3 and so 100 and so the initial digit is 1 ( the first digit of factor number 101).
That is all.

------------------------Starting Second digit

A:
303/199 = n
n * 199 = n1
303-n1 = R
A1:
303/190 = n
n * 190 = n1
303-n1 = R
-----------------------------
A:
303/189 = n
n * 189 = n1
303-n1 = R
A1:
303/180 = n
n * 180 = n1
303-n1 = R
-----------------------------
A:
303/179 = n
n * 179 = n1
303-n1 = R
A1:
303/170 = n
n * 170 = n1
303-n1 = R
-----------------------------
A:
303/169 = n
n * 169 = n1
303-n1 = R
A1:
303/160 = n
n * 160 = n1
303-n1 = R
-----------------------------
A:
303/159 = n
n * 159 = n1
303-n1 = R
A1:
303/150 = n
n * 150 = n1
303-n1 = R
-----------------------------
A:
303/149 = n
n * 149 = n1
303 - n1 = R
A1:
303/140 = n
n * 140 = n1
303 - n1 = R
-----------------------------------
A:
303/139 = n
n * 139 = n1
303 - n1 = 13
A1:
303/130 = n
n * 130 = n1
303 - n1 = R
-----------------------------------
A:
303/129 = n
n * 129 = n1
303 - n1 = R
A1:
303/120 = n
n * 120 = n1
303 - n1 = R
-----------------------------------
A:
303/119 = n
n * 119 = n1
303 - n1 = R
A1:
303/110 = n
n * 110 = n1
303 - n1 = R
-----------------------------------
A:
303/109 = n
n * 109 = n1
303 - n1 = R
A1:
303/100 = n
n * 100 = n1
303 - n1 = R
-----------------------------------

Repeat the Fundamental Note (FIRST and SECOND and find a smaller result) and find the second digit.

(second digit is 0)

-----------------------------------------------Starting Third digit
Last digit use A or A1 because A = A1

A = 109 to 100 (for example 109, 108, 107, 106, 105, 104, 103, 102, 101, 100)

A1 = 109 to 100 (for example 109, 108, 107, 106, 105, 104, 103, 102, 101, 100)

-------------------------------------
A:
303/109 = n
n* 109 = n1
303-n1 = R
-------------------------------------
A:
303/108 = n
n * 108 = n1
303-n1 = R
-------------------------------------
A:
303/107 = n
n* 107 = n1
303-n1 = R
-------------------------------------
A:
303/106 = n
n* 106 = n1
303-n1 = R
-------------------------------------
A:
303/105 = n
n* 105 = n1
303-n1 = R
-------------------------------------
A:
303/104 = n
n* 104 = n1
303-n1 = R
-------------------------------------
A:
303/103 = n
n* 103 = n1
303-n1 = R
-------------------------------------
A:
303/102 = n
n* 102 = n1
303-n1 = R
-------------------------------------
A:
\[
\begin{align*}
303/101 &= n \\
n*101 &= n1 \\
303-n1 &= R
\end{align*}
\]

\[
\begin{align*}
A: \\
303/100 &= n \\
n*100 &= n1 \\
303-n1 &= R
\end{align*}
\]

That's all.

**Program in Language C**

THAT PROGRAM NOT USE FLOAT NUMBER AND THE CONSEQUENCE IF NUMBER A OR A1 IS GREATER THAN THE PRODUCT THE RESULT IS ZERO.

This factorization method is backed up by a C language code. Using the code you can verify the results obtained using the method. The code is set to work on the order of 10,000 digits.

The code uses the GMP library, to use it:

1. Create a new folder on your desktop and give it a label which you can remember
2. In the created folder insert the following files; main.c, steps1.h, and steps2.c.
3. Create a library subfolder in the same folder and download gmp.h and libgmp.a from the internet.
   4. Compile the program by using gcc and give the following command:

\[
gcc -o nomefile.exe main.c -L./lib -lgmp
\]

//main.c

```c
#include <stdlib.h>
#include <stdio.h>
#include "lib/gmp.h"
#include "steps1.h"

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1
#define MP_OKAY 0
#define MAX_DIGITS 10000

int main ( int argc , char **argv)
{
```
char io_string[MAX_DIGITS +1];

int result;
mpz_t c1 , c2 , c3 , c4, c5;
mpz_inits(c1,c2,c3,c4,c5,NULL );
mpf_set_default_prec(512);

printf("\nInsert product of two prime numbers (for example 101*3 = 303 ): ");
scanf("%s", io_string);
result = mpz_set_str(c1,io_string , 10);
if(result != MP_OKAY)
{
    printf("\nImpossible read the number!\n");
    return EXIT_FAILURE;
}

while (mpz_cmp(c1,c1) >= 0)
{
    printf("\n\n-------INSERT NUMBER FOR DIVISION Example (Insert number A or A1 (for example 999 or 900)): ");
    scanf("%s", io_string);
    result = mpz_set_str(c2,io_string , 10);
    if(result != MP_OKAY)
    {
        printf("\nImpossible read the number!\n");
        return EXIT_FAILURE;
    }

    mpz_div(c3,c1,c2);
    mpz_mul(c4,c3,c2);
    mpz_sub(c5,c1,c4);

    printf("\nNumero Iniziale Prodotto : %s", mpz_get_str(io_string , 10 , c1));
    printf("\nA or A1\n");
    printf("\nResult DIV : %s", mpz_get_str(io_string , 10 , c3));
}
printf("\nResult MUL : %s", mpz_get_str(io_string , 10 , c4));

printf("\nResust REST: %s", mpz_get_str(io_string , 10 , c5));

}

system("pause");
printf("\n\n");
mpz_clears(c1,c2,c3,c4,c5,NULL);

return EXIT_SUCCESS;
}

@end main.c

//steps1.h

#ifndef __STEPS__
#define __STEPS__

#define MP_OKAY 0
#define MP_FAIL -1

int step111(mpz_t c3 , mpz_t c1 , mpz_t c2 );
int step222(mpz_t c4 , mpz_t c3 , mpz_t c2 );
int step333(mpz_t c5 , mpz_t c1 , mpz_t c4 );

#include "steps2.c"

#endif

@end steps1.h

//steps2.c

int step111(mpz_t c3 , mpz_t c1 , mpz_t c2 )
{
    mpz_div(c3,c1,c2);
}
int step222(mpz_t c4, mpz_t c3, mpz_t c2)
{
    mpz_mul(c4, c3, c2);
}
int step333(mpz_t c5, mpz_t c1, mpz_t c4)
{
    mpz_sub(c5, c1, c4);
}

//end steps2.c

**Conclusion**

The fast factorization method preferred in this paper seems to cater for the concerns which have been in the center of RSA key cryptography decryption. The approach factorizes any given size of number $n$ in the shortest time possible, unlike the other methods which have been in use before. For instance, the Quadratic Sieve method is limited to integers of digits between 50 and 100, the Trial Division cannot handle numbers with more than 9 digits, and others being time insensitive. Also, to add to its efficiency in determining the two prime numbers, the fast factorization method is backed up by a C language program for confirmation purposes. Therefore, this is probably the long-sought fast and simple factorization approach.

**References**

[1] Hung-Min Sun, Mu-En Wu, Wei-Chi Ting, and M. Jason Hinek, “Dual RSA and Its Security Analysis”, IEEE Transactions on Information Theory, Vol. 53, No. 8, Aug. 2007.
[2] Joao Carlos Leandro da Silva, “Factoring Semi primes and Possible Implications”, IEEE in Israel, 26th Convention, pp.182-183, Nov. 2010.
[3] Arjen K. Lenstra, “Integer Factoring”, *Designs, Codes and Cryptography*, 19, 101–128, 2000.
[4] Sattar J. Aboud, “An efficient method for attack RSA scheme”, ICADIWT Second International Conference, pp.587-591, 4-6 Aug 2009.
[5] L. Scripcariu, M.D. Frunza, "A New Character Encryption Algorithm", ICMCS 2005, pp. 83 - 86, Sept., 2005.
[6] Mathew E. Briggs, *An introduction to the General Number Field Sieve*, a master thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University.
[7] Hans Riesel, *Prime numbers and computer methods for factorization*, Second Edition, Birkhauser, 1994.
[8] Song Y. Yan, *Number Theory for Computing*, Second edition, Springer, 2000.
[9] J. Pollard, "Theorems on factorization and primality testing", Proc. Cambridge Philos. Soc., Vol. 76, pp.521-528, 1974.
[10] J. Pollard, "Monte Carlo methods for index computation (mod p)", Math. Comp., Vol. 32, pp.918-924, 1978.
