International Journal of Theoretical and Applied Mathematics
2017; 3(6): 199-202
http://www.sciencepublishinggroup.com/j/ijtam
doi: 10.11648/j.ijtam.20170306.14
ISSN: 2575-5072 (Print); ISSN: 2575-5080 (Online)

Methodology Article

New Pragmatic Algorithms to Improve Factoring of Large Numbers

Mohamed Zaki Abd El-Mageed[1], Hassan Hussein[2]

[1]Department of Computer Since, Faculty of Engineering, Al-Zahra University, Cairo Egypt

[2]Research Development Center, National Defense Council, Cairo, Egypt

Email address:
mzaki.azhar@gmail.com (M. Z. A. El-Mageed), hassan.m.hussein@gmail.com (H. Hussein)

To cite this article:
Mohamed Zaki Abd El-Mageed, Hassan Hussein. New Pragmatic Algorithms to Improve Factoring of Large Numbers. *International Journal of Theoretical and Applied Mathematics*. Vol. 3, No. 6, 2017, pp. 199-202. doi: 10.11648/j.ijtam.20170306.14

Received: September 28, 2017; **Accepted**: November 13, 2017; **Published**: December 5, 2017

Abstract: Rivest, Shamir, Adleman, RSA algorithm is a popular public key cryptosystem and is known to be secure, however, this fact relies on the difficulty of factoring large numbers. No algorithm has been published that can factor all integers in polynomial time. This paper proposes a new function that can be used to improve the process of factoring integer numbers. It gets the factor faster than known methods. By making use of such proposed function, corresponding two algorithms are proposed and pseudocoded. The utilization of these algorithms along with the basics of the theory of numbers led to three other new factoring algorithms. The five algorithms are implemented and verified using Python Language. The tabulated results that represent the time of factorization versus the number of digits of the large number have indicated the applicability of the last three algorithms.

Keywords: RSA, Large Number Factorization, Number Field Sieve, Greatest Common Divisor (GCD)

1. Introduction

Integer factorization is a classical problem in computer science and number theory [2], [3]. When the numbers are very large, no efficient, integer factorization algorithm is known. Several cryptographic systems are based on the hardness of factorization problems have been proposed. Among them, the RSA system is the most famous and widely used [2]. Actually, not all numbers of a given length are equally hard to factor. This fact is obvious from Table 2 (Sec. 4). The hardest instances of that problem occur when the corresponding to prime numbers are both large, randomly chosen and about the same size (but not too close).

Zaki and Hussein [1], proposed a function that has been used as fitness function in genetic algorithms to improve the heuristic search for factoring integer numbers. The fastest general-purpose factorization algorithm is the Number Field Sieve (NFS), [4]-[7]. This paper is organized as follows. In section 2, related works are discussed and a mathematical background is given in section 3. The proposed function in [1] is extended in section 4. Additional new algorithms for factoring integers will be discussed in section 5. Finally, the paper is concluded in section 6.

2. Related Work

Many references, [4]-[7] have explained methods of factoring integers, to attack the public key [3], such as Elliptic Curves, Continued Fraction, Quadratic Sieve and Number Field Sieve (NFS). Implementation of any of these methods, takes impractical factoring time.

Abd El-Mageed and Hussein [1] have proposed function that used as fitness function to improve the heuristic search and gave genetic algorithms in order to factor the integer number. However, the time complexity of a genetic algorithm is $O(a^3)$ where a is the number of the constituents prime factors of the underlying integers. Here, the proposed algorithms, depend on a new function that can optimize memory and time of implementation for factoring integers.

It can be noticed that, on the research level several quantum systems have solved the quantum factorization problem [9] [10] and achieved noticeable success [11]. However, in practice, RSA is still popular and widely spread.

3. Mathematical Background

Suppose we want to factor the composite number N. We choose a bound, and identify the factor base, P, the set of all primes less than or equal to the chosen bound. Next, we search for positive integers z such that both z and (z+N) have all of their prime factors in P. We can therefore write,

{\displaystyle z=\prod_{p_{i}\in P}p_{i}^{a_{i}}}

$$z = \Pi_i \, p_i{}^{a_i} \tag{1}$$

$p_i \in$ P and i = 1, 2,…, number of primes in P

and similarly:

$$z + N = \Pi_i \, p_i{}^{b_i} \tag{2}$$

{\displaystyle z+n=\prod _{p_{i}\in P}p_{i}^{b_{i}}}

But z and (z+N) are congruent modulo N, and each integer z yields a multiplicative relation (mod N) among the elements of *P*, i. e.

$$\Pi_i \, p_i{}^{a_i} = \Pi_i \, p_i{}^{b_i} \; (mod \; N) \tag{3}$$

{\displaystyle \prod _{p_{i}\in P}p_{i}^{a_{i}}\equiv \prod _{p_{i}\in P}p_{i}^{b_{i}}{\pmod {n}}}

Where a and b are nonnegative integers. When we have generated enough of these relations one can use the methods of linear algebra to multiply together these various relations in such a way that the exponents of the primes are all even [8]. This will give us a congruence of squares of the form $a^2 \equiv b^2$ (mod N), which can be turned into a factorization of N.

N=gcd (a-b, N)×gcd (a+b, N).

4. The Proposed Extended Function

The new proposed function *gf (x, N)* is defined as follows:

$$gf_N(x) : Z_N \to \{a : a \equiv 0 (\text{mod } N)\} \cup \{1\}$$

i.e. $gf_N(x) \equiv \begin{cases} 1 \\ a \end{cases} (\text{mod } N)$, where a is factor of N.

Proposition 1.

Let p be a factor of N, then, by repeating $k*p \; (mod \; N)$, $k>0$, the result is p.

Because $k*p$ is congruence of p.

The basic procedure of implementing *gf (x, N)*, as such, is given algorithm that can be pointed out as follows:

Algorithm 1 *gf (x, N)* (implementation of the function, given factor)

Input: the number N to be factored and an element x in the interval [2, N-1]

Output: the factor p of N or *1*.

1- Set $x \equiv x (\text{mod } N)$

2- If $x = 0$ or $x = N$, then return *1*

3- While $x>1$,

Set $b \leftarrow x$, and $x \equiv b (\text{mod } N)$.

4- If $x = 0$, then return b

5. else, return 1

Note that this algorithm is repeated subtraction of x from N.

Theorem 1.

Let 1< x <N. If gf (x, N) > 1, then gf (N-x, N) = gf (x, N) > 1.

Proof

Let *gf (x, N)* produces a factor of *N,*.

$y \equiv N - x$, $x = y$, $y \equiv N - x$ and repeat until $y = 0$.

i. e. N-x = 0 means N mod x = 0..

Then, last x is factor of N.

Examples

1-N = 17*37 = 629,

x = 18, 35, 36, 47, 54. 66, 97, or 99, give gf (x, N) =17

2-N = 7*11 = 77.

x = 10, or 67 give gf (x, N) = 7

3-N = 5*7 = 35.

x==6, or 29 give gf (x, N) = 5

Corollary

Let N = p*q. then # {1<x<N: gf (x, N)>1} ≥ (p+q)

Conjecture:

Let $N = p*q$, $1<p<q<n$. if q-1 is multiple of prime number (not equal p). Then $|gf\ (i, N)>1| >> |gcd\ (i, N)>1|$, for i in *[1, N-1]*.

If $q-1$ is multiple of p, then $|gf\ (i, N)>1| = |gcd\ (i, N)>1|$, for i in *[1, N-1]*.

and vice versa with respect to *p, q*

Also, this function egf (x, N) can be used to get algorithm that extends the co-domain of the *gf* function.

Algorithm 2 e*gf (x, N)* (implementation of the function, has factor)

Input: the number N that factoring and an element x in the interval *[2, N-1]*

Output: the factor p of N or *1*.

1- Set $p \leftarrow gf\ (x, N)$.

2- If $N>p>1$, then return p

3- Else

3.1. Set $b \leftarrow (a^* (a^2-1)) \; (\text{mod } N)$

3.2. Return *gf (b, N)*.

Example

N = 5*7 = 35

Table 1. *Values of egf(x,N).*

x	2	4	6	8	9	11	13	16	19	22	24	26	27	29
egf (x, N)	5	5	5	7	5	5	7	5	5	7	5	5	7	5

This clearly shows that *egf (x, N)* exceeds the co-domain of *gf (x, N)* which exceeds the co-domain of *gcd (x, N), Table 1.* Actually, any heuristic can use one of these algorithms, gf (x, n), or egf (x, n) as a fitness function to get the target factor

5. Other Three New Algorithms

Here, three proposed new factoring algorithms are

presented. The following algorithm uses the base of the integer n.

Algorithm 3 basefact (n):

Input: the integer n which equal $p*q$

Output: p, q

1- set $r \leftarrow \lfloor \sqrt{n} \rfloor + 1$ and $m \leftarrow \lfloor \sqrt[3]{n} \rfloor + 1$

2- make r odd integer.

3- set $q \leftarrow 2*r+1$. and $i \leftarrow 0$

4- while $r>m$:

4.1- if q divides n, then return $(n/q, q)$.

4.2- exceed i by 1, and decrease r and q by 2.

4.3- calculate e as a base of n with respect to r.

4.4- if number of digits of e equal *3* and its last digit is *0*, i. e. $e_2 = 0$, then return $(r, r*e_0 + e_1)$

4.5- if number of digits of e greater than 3, then return *(1, n)*

The following algorithm for factoring, uses the iteration method to solve equation of degree 3 that equals to the integer $n = p * q$.

$$\text{The equation: } a*x^2 + b*x - n = 0 \qquad (4)$$

take $a = -1$.

Algorithm 4 iterfact (n):

Input: the integer n which equal $p*q$

Output: p, q

1- set $r \leftarrow \lfloor \sqrt{n} \rfloor + 1$ and $p \leftarrow 1$ and $q \leftarrow n$.

2- for b = 1 to b = r with step 2 make the following:

2.1- set $A \leftarrow \sqrt{b * b + 4 * n}$

2.2- if A is integer number, then p equal $(A-b) / 2$ and q equal $(A+b)/2$

3- return (p, q)

This algorithm assume the equation $-r^2 + b*r + N = 0$, to make the square root A positive.

Algorithm 5 propfact (n):

Input: the integer n which equal $p*q$

Output: p, q

1- set $m \leftarrow \lfloor \sqrt[3]{n} \rfloor$, j = number of digits of m

3- set v← 1, c← 0, p← 1and q ← 1.

4- for $a = -m/j$, to $a = m/j$:

4.1- for b = 1 to b = m/j:

4.1.1- if gcd (a, b) = 1:{greatest common divisor}

4.1.1.1- set t← v and c← c+1.

4.1.1.2- calculate $x = (a+b*m)$, and $v = (v*x)\%n$

4.1.1.3- if v = 0, then set p ← egf (x, n), q ←e gf (t, n), and v ← t.

5- if p = q = 1, then set p = e gf (v, n) and q = n/p

6- return p, q

Table 2, shows the relation between the number of decimal digits of N and time of implementation of basefact, iterfact, and propfact algorithms.

Table 2. *Relation between implementation time and number of digits.*

No. of digits of N	N	Time of propfact	Time of Iterfact	Time of basefact
2	21	0.000	0.000	0.000
3	407	0.000	0.000	0.000
4	2279	0.003	0.008	0.057
5	10807	0.006	0.005	0.043
6	782303	0.010	0.005	0.026
7	1104143	0.011	0.015	0.025
8	42757439	0.035	0.005	0.025
9	105951673	0.049	0.013	0.031
10	1459321343,	0.261	0.031	0.052
11	15262008803	0.380	0.048	0.169
12	391604285407	1.243	0.184	0.512
13	1125754310689	3.003	0.154	0.314
14	24366140091809	21.627	0.048	0.16
15	106565228726479	66.141	1.923	5.059
16	3156627667094141	109.374	21.894	45.141

6. Conclusion

Integer factorization is an important computational problem, and it is the foundation of the RSA cryptosystem. Since the invention of the general number field sieve in 1993, there is no substantial progress on this problem.

The method that has been proposed here is actually new and it depends on the large co-domain of *gf* an *egf* functions. By extending these functions co-domain three algorithms are given.

These algorithms have been programmed in Python and implemented on a PC. This result puts a doubt on the conclusion of Rivest which states that factoring a 200 digit number requires four billion years on a computer that forms one instruction per micro second

The promising results of the presented algorithms encourage us to enhance their implementation by making use of long integer architecture packages that can use *gf* or e*gf* to decrease time of factorization. In the previous tables, time of implementation depends on the difference of two factors of factored number. In conclusion, it is clear that the proposed method is relatively simple, fast and scalable when compared to existing methods.

References

[1] Mohammed Z. Abd El-Mageed and Hassan M. H. Hussein, An Effective GA Fitness Function To Guide Heuristic Search Integer Factorization, Paper Accepted for GJTAMS Paper Code: 5183.

[2] Song Y. Yan, Computational Number Theory, Higher Education Press, WILY, 2013.

[3] R. L. Rivest, A. Shamir, L. Adleman: A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21 (1978).

[4] H. Cohen, A Course in Computational Algebraic Number Theory, Springer-Verlag, Berlin, Heidelberg, New-York, 1996.

[5] A. K. Lenstra and H. W. Lenstra, *Algorithms in Number Ttheory,* Handbook of theoretical computer science, J. Van Leeuwen, A. Mayer, M. Nivat, M. Patterson and D. Perrin (eds.), Elsevier, 1990.

[6] Song Y. Yan, Cryptanalytic Attacks on RSA, Springer Science+Business Media, 2008.

[7] Song Y. Yan, computational number theory, Higher Education Press, WILY, 2013.

[8] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, The Factorization of the Ninth Fermat Number, Math. Comp. 61 (1993), 319-349.

[9] J. Chu, The beginning of the end for encryption schemes, MIT News, March, 2016.

[10] L. Zyga, Quantum physics offers new way to factor numbers, Phys.org, November, 2016.

[11] R. Dridia and H. Alghassib, Prime factorization using quantum annealing and computational algebraic geometry, NCBI, 2017.