DOI: 10.21917/ijct.2016.0192

INTEGER FACTORIZATION IMPLEMENTATIONS

[1]Reza Alimoradi and [2]Hamid Reza Arkian
[1]Department of Mathematics and Computer Science, Faculty of Science, University of Qom, Iran
E-mail: [1]r.alimoradi@qom.ac.ir, alimoradi.r@gmail.com
[2]Research Center of Developing Advanced Technologies, Iran
E-mail: [2]arkian@rcdat.ir

Abstract
One difficult problem of mathematics that forms the basics of some public key cryptography systems like RSA, is finding factors of big numbers. To solve this problem, many factorization algorithms have been offered with different complexities. Many attempts have been made today to implement these factorization algorithms. And these implementations are different from various aspects. Each of these factorization algorithms is efficient for numbers with a specific size. These algorithms can be compared with regard to time and memory complexity. In this paper, some of these implementations are studied and compared and consequently the most appropriate one will be introduced.

Keywords:
Factorization, GMP-ECM, CADO-NFS, NFS, RSA, ECM

1. INTRODUCTION

Public key cryptography based on complexity of hard problem in mathematics. Security in some current cryptography methods, like RSA public key cryptography systems, depends on complexity of factorization of integer numbers to their prime factors. Such algorithms are widely used in order to maintain security in networks and exchange data in a confidential and authentic way through insecure communicational channels. This heightens competition for a more efficient and beneficial attack at the main core of this group of algorithms which is complexity of number factorization. Introduction of new methods and reports of new developments in number factorization bear evidence to this claim. The problem of number factorization has many years history. So, with the passage of time, different algorithms have been introduced each of which has been variously implemented. In most of them, the purpose is reaching a more optimal implementation and a higher speed of factorization. This paper studies some implementations of factorization algorithms. Factorization algorithms and their complexity, size of the appropriate numbers for them, and the records reached by them are included in the second part of the paper. The third section introduces some software implementations. The results will follow as the final part.

2. ALGORITHMS

As mentioned before, in number factorization many algorithms with different features regarding speed and interval of factorization and different functions have been designed [10]; such as trial division, Pollard, ECM [19], quadratic sieve [5][8], [21] and number field sieve [7].

2.1 COMPLEXITY

The most important reason for this variety and as a result, dissatisfaction with the existing algorithms was a decrease in the algorithms' complexity in integer numbers' factorization which is especially important for big numbers. So, as the following Table.1 shows the most significant feature of the more recent algorithms compared to the old ones is their decreased complexity. Of course, it must be pointed out that designing a new algorithm does not necessarily mean the previous algorithms' abandonment. It is because number factorization algorithms have different functions and they keep working in other functions or in factorization of small intervals.

Table.1. Complexity of number factorization algorithms

Name	Inventor	Date	Complexity	Depends on
Trial Division	-	-	P ~ N^(1/2)	Size of p
Fermat	D. Fermat	Circa 1650	N^(1/2)	Size of N
SQUFOF	D. Shanks	1971	N^(1/4)	Size of N
Lehman (Fermat)	R. Lehman	1974	N^(1/3)	Size of N
P-1 P+1	J. Pollard H. Williams	1974 1982	BlogB	Smoothness of factor
Pollard's Rho	J. Pollard	1975	P^(1/2)	Monte carlo
Continued Fractions	Brillhart, Morrison	1975	Ln[1/2, sqrt(2)]	Size of N
ECM	H. Lenstra	1987	Ln[1/2, sqrt(2)]	Size of p
Dixon's	J. Dixon	1981	Ln[1/2, 2 sqrt(2)]	Size of N
Quadratic Sieve	C. Pomerance	1981	Ln[1/2. 1]	Size of N
MPQS	R. Silverman	1987	Ln[1/2, 1]	Size of N
SIQS	P. Montgomery	1993	Ln[1/2, 1]	Size of N
Number Fiels Sieve(Special NFS)	J. Pollard	1993 1988	Ln[1/3, 1.92] Ln[1/3, 1.52]	Size of N

2.2 ALGORITHM INTERVAL

One touchstone for choosing the right algorithm is complexity of their implementation. For example, implementation of the trial division and Pollard is very easy. About ECM and the quadratic sieve, it is hard to reach an implementation with a proper speed. Likewise, implementation of the number field sieve [6][14] is very complex. On the other hand, preprocesses needed for some

algorithms decrease speed of factorization. Thus, each of them is highly efficient in a specific interval. With regard to complexity of factorization algorithms, it is very logical for each of them to be appropriate for factorization of numbers with specific sizes.

- The ECM algorithm is appropriate for numbers with less than 30 digits.
- The QS algorithm is the best choice for numbers with between 30-60 digits.
- The MPQS is most efficient for numbers with between 60-120 digits.
- The NFS algorithm is proper for numbers with more than 120 digits.

However, improvements of the ECM algorithm and its very optimal implementation i.e. GMP-ECM, in addition to using hardware implementations, have all made it possible for the ECM algorithm to be applied in factorization of numbers as big as 200 bits.

2.3 FACTORIZATION RECORDS

In the area of number factorization, different records have been gathered/compiled with different purposes. About the records one must note that:

- All these records are gathered by the public number field sieve algorithm.
- The sieving stage in this method is very time consuming; in fact, the whole complexity of the factorization is due to this stage. Therefore, the current paper focuses on the time required for this sieving.
- The previous records would compute time based on MIPS standards i.e. the number of years a computer needed to factorize a number by performing 1 million directions per second. But, nowadays, Pentium or AMP processors are mostly used; and time computations based on the number of years a Pentium 1GHz processor needs to complete sieving have been done till now [8].

Table.2. Some factorization records [8]

Number	Digits	Date completed	Sieving time
C158	158	January, 2002	3.4 Pentium 1GHz CPU years
RSA-160	160	March, 2003	2.7 Pentium 1GHz CPU years
RSA-576	174	December, 2003	13.2 Pentium 1GHz CPU years
C176	176	May, 2005	48.6 Pentium 1GHz CPU years
RSA-200	200	May, 2005	121 Pentium 1GHz CPU years
RSA-768	232	Dec, 2009	3300 Opteron 1GHz CPU years

As mentioned earlier and is clear in the above Table.2, to factorize RSA cryptography systems, number field sieve algorithm must be used. This is because this algorithm has the sub exponential complexity [1], [2], [3], [4], [11], [12]. However, efficient implementation of this algorithm requires great cost and time [13], [16], [24]. To reduce them, different software implementations for NFS have been offered; some of which are introduced below.

3 SOME IMPLEMENTATIONS

There are various implementations in number factorization; each designed with a specific purpose and includes one or more instance(s) of number factorization algorithms.

3.1 GMP-ECM

It is a free, open source implementation of ECM method which is used for number factorization. The main goal of this project is finding factors with 50 or more digits through the ECM method. Significance of this method is because it is the fastest one introduced for factorization of almost less than 200 bits numbers. It is worth noting that number factorization in small intervals is not independently important. They are used in algorithms such as quadratic sieve and number field sieve in which millions of small numbers get factorized in a big number's factorization process. That is why efficiency of small numbers factorization has attracted much attention and many attempts have been made to make these factorizations faster. The GMP-ECM library is considered the fastest and the most authentic existing library in this field. Experiments already have done bear evidence to this claim. The experiments have been performed on various number intervals. In this library, implementations existed in the LiDIA [15] - a library for working with the elliptic curves, and the implementations in the NTL library [18] - a library about number theory - alongside independent implementations.

Table.3. Records of the ECM implementations in our work

Size Bit/Digit	GMP ECM	LiDIA ECM	NTL ECM	Personal ECM
16 /5	<1ms	<1ms	<1ms	NO RESULT
32 / 10	<1ms	0.030	0.030	NO RESULT
50 / 15	0.001	16.540	13.250	0.001
64 / 19	0.01	4647.71	7278.22	0.01
80 / 24	0.028	~	~	0.04
100 / 30	0.188	~	~	0.55
128 / 39	3.112	~	~	4.09
150 / 45	12.86	~	~	14.19
170 / 51	155.26	~	~	173.6

The main problem with using LiDIA and NTL was their weakness against number factorization of the RSA numbers (the numbers included two factors of p and q) which made it practically impossible to factorize these numbers.

Of course, our implementation and the GMP-ECM [7, 23] have been separately compared. The results are shown in the Table.3, Fig.1 and Fig.2. It must be noted that these numbers have specific features; thus, the ECM algorithm will not be able to factorize every composite number as big as the mentioned numbers.

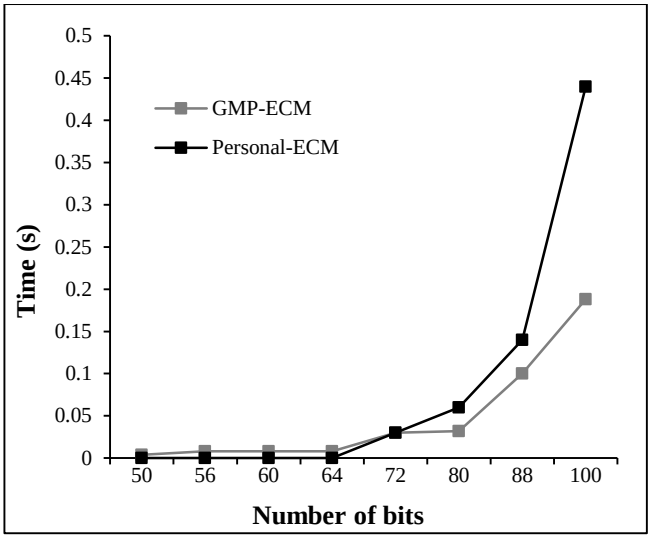

Fig.1. A comparison between our implementation of ECM and GMP-ECM in an interval smaller than 100 bits

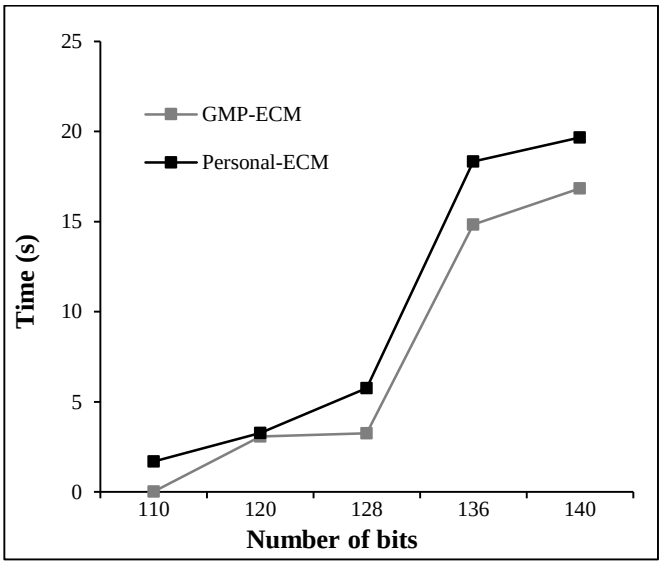

Fig.2. A comparison between our implementation of ECM and GMP-ECM in a 110-140 bits interval

Note: However contrary to Fig.1 and Fig.2, Personal ECM has proved to be better in specific instances. In fact GMP-ECM is more efficient most of the times.

In addition to these tests, many records are published by the GMP-ECM that will come in Table.4.

Table.4. Records of finding the factor by GMP-ECM [25]

Digit	Factor	From	Date	Who
69	474942339376967475871960321762614254290810243038880418971061342256529	2^1822+1	20 Feb 2011	B. Dodson
67	30433922592068709674825000822339863904244	2^1151+1	06 May 2011	J. Bos, T. Kleinjung, A. Lenstra, P. Montgomery
	27895732225880158451433430 7			
64	721382090243496336415118656055915842092158633984503832846830 5677	3^634+1	27 Mar 2011	S. Wagstaff
64	309911794558115751747256081735354877428245029539903688747366609	2^964+1	04 Apr 2011	J. Bos, T. Kleinjung, A. Lenstra, P. Montgomery
64	298570534025587797438445775521787580015202118215146887548612 1347	2^1051+1	12 May 2011	J. Bos, T. Kleinjung, A. Lenstra, P. Montgomery
63	844419612452198561310720080038799341972811718390811373415844 777	3^723-1	24 Apr 2011	S. Wagstaff
63	767749587188080891246822731496641644423848612695929371145177 219	2^1049+1	10 Mar 2011	J. Bos, T. Kleinjung, A. Lenstra, P. Montgomery
63	112985563755005263987526248018869431962485492728920337098962 673	2^1109+1	21 Mar 2011	J. Bos, T. Kleinjung, A. Lenstra, P. Montgomery
62	550466955462395093013756031588759767837419833205721663507213 91	11^289-1	02 Apr 2011	B. Dodson

3.2 MIRACL

It is a complete package dealing with big numbers. At present, its 5.5 version is published. It is consisted of a quite complete set of number factorization algorithms. This package has fine examples of working with number factorization algorithms. Unfortunately it has a low speed and cannot be practically used in big numbers factorization [17].

3.3 PARI/GP

This package is a complete algebraic system whose 2.3.5 versions have been introduced. It includes Pollard, ECM, MPQS and SQUFOF algorithms. The codes related to small numbers have been properly implemented. But about the MPQS algorithm, it is not well developed. So, it is a good package only for numbers with about 47 digits [20].

3.4 FLINT

It is an algebraic library including trial division, Quadratic sieve, MPQS and SQUFOF methods. In this library, the focus is on the logic of polynomials. Nevertheless, its efficiency is limited to the interval of numbers with about 55 digits.

3.5 YAFU

It is an instrument to factorize numbers. Its 1.12 version includes almost all factorization algorithms except for the number field sieve. It can properly select the appropriate algorithm in different intervals. The SIQS method is quite optimally implemented. This leads to the package's very high speed in factorizing numbers with between 50-100 digits [22].

3.6 MSIEVE

It is a complete package of number factorization consisting of almost all existing algorithms. When the inputs are less than 25 digits, small current routines perform factorization. For bigger numbers, it applies GMP-ECM library which makes use of p+1 and p-1 and ECM method with an amount chosen by the user. If all these methods did not lead to the complete factorization of the input, the library shifts to a stronger method i.e. the quadratic sieve algorithm with internal primary numbering. Msieve also includes a complete implementation of the number field sieve algorithm which can be applied in bigger numbers' factorization. At present, it covers numbers with about 275 digits. However, programmers of the library do not expect the package to perform a complete factorization of numbers bigger than 120 digits independently. Notice that it has a low speed in the sieving stage, in addition to an optimal implementation of choosing the polynomial stage which is related to the number field sieve algorithm.

3.7 GGNFS

It is a package specifically focused on the number field sieve algorithm and offers an appropriate implementation of it. At present, its 0.77.1 version is available which can be used in factorization of specific numbers with about 180 digits and general numbers with an interval of 140 digits. Of course, bigger numbers have also been factorized with the same quality. But there are some points about the software which make the bigger factorizations somehow harder. In other words, for using its different parts, special techniques are required. Making use of basic functions of the GMP, ability for parallel running and it's very easy application are some other features of this library [16].

3.8 CADO-NFS

It is a complete implementation of the number field sieve algorithm. Corresponding to different phases of the algorithm, it has different programs. A script has also been prepared to run them. In addition, there is the possibility of under-web parallel running of the computers. Its primary development is based Linux x86-64 with gcc 4.4 which has been run on the processor Intel. On other 64-bit systems, it also gets regularly checked. Nevertheless, there is the possibility of its implementation on systems other than the main platform. Of course, it may have some errors too. For instance, it can be run on the 32-bit Linux, but not on the windows. Anyway, this package is able to compete with the best implementations available to MPQS (i.e. Msieve, as is generally agreed) for numbers bigger than 95 digits. According to its programmers, factorization of a number with120 digits by this software takes 3 or 4 days on the core of a typical computer. Factorization of a 140- digits number requires a month on a core. Also, a 160-digits number needs 6 or 7 months on one core to be factorized. The integer number entered must not be very small, i.e. not less than 60 digits (CADO is efficient for numbers with more than 85 digits) and it is best to remove its small factors [24].

Below is a table of the results of testing these implementations and the algorithms applied in them.

Table.5. Results of the algorithms [25]

Name	0.02 second	1 second	1 minute	1 hour
Trial Division	15	18	21	24
Fermat	17	19	23	27
Pollard's Rho	18	25	33	39
SQUFOF	19	25	32	40
MPQS	32 (YAFU) 33 (FLINT) 26 (Msieve)	50 (YAFU) 52 (FLINT) 54 (Msieve)	68 (YAFU) 70 (FLINT) 71 (Msieve)	80 (FLINT) 92 (Msieve)
SIQS	22 (YAFU)	55 (YAFU)	77 (YAFU)	97 (YAFU)
Number Fiels Sieve	-	-	-	98 (GGNFS + ms)
MIX	25 (MIRACL) 26 (Msieve) 32 (PARI) -- (YAFU)	35 (MIRACL) 48 (PARI) 53 (Msieve) 50 (YAFU)	62 (MIRACL) 66 (PARI) 71 (Msieve) 76 (YAFU)	75 (MIRACL) 83 (PARI) 91 (Msieve) 97 (YAFU)

4. CONCLUSION

Existing implementations and algorithms in integer numbers' factorization have different functions. However, according to the comparisons, one can conclude that for 200 bits numbers, using GMP-ECM which is based on the ECM algorithm is very efficient. Moreover, for extra big numbers, implementation of CADO-NFS which is based on the NFS algorithm is highly applicable.

REFERENCES

[1] Aoki Kazumaro, Yujim Kida, Takeshi Shimoyama and Hiroki Ueda, "GNFS Factoring Statistics of RSA-100, 110, . . ., 150", *IACR Cryptology ePrint Archive*, pp. 1-9, 2004.

[2] Friedrich Bahr, Jens Franke, Thorsten Kleinjung and M. Bohm, "*RSA-640 e-mail Announcement*", 2005.

[3] Friedrich Bahr, Jens Franke, Thorsten Kleinjung, M. Lochter and M. Böhm, "*RSA-160 e-mail Announcement*", 2003.

[4] Friedrich Bahr, Jens Franke, Thorsten Kleinjung, M. Lochter and M. Böhm, "*RSA-200 e-mail Announcement*", 2005.

[5] Henk Boender and Herman Te Riele, "Factoring Integers with Large-prime Variations of the Quadratic Sieve", *Experimental Mathematics*, Vol. 5, No. 4, pp. 257-273, 1996.

[6] Matthew Briggs, "An Introduction to the General Number Field Sieve", Ph.D Dissertation, Department of Mathematics, Virginia Polytechnic Institute and State University, 1998.
[7] T. Charron, T. Daminelli, T. Granlund, P. Leyland and P. Zimmermann, "The ECMNET Project", Available at: http://mathforum.org/library/view/17102.html.
[8] General Purpose Factoring Records, Available at: http://www.crypto-world.com/FactorRecords.html.
[9] R.M. Elkenbracht Huizing, "Factoring integers with the Number Field Sieve", Ph.D Dissertation, Leiden University, 1997.
[10] Dana A. Jacobsen, "Methods and Implementations for Integer Factorization", CS567 Cryptology I, Boise State University, 2009.
[11] James Cowie, Bruce Dodson, R. Marije Elkenbracht Huizing, Arjen K. Lenstra, Peter L. Montgomery and Jorg Zayer, "A Worldwide Number Field Sieve Factoring Record on to 512 Bits", *Proceedings of International Conference on the Theory and Applications of Cryptology and Information*, pp. 382-394, 1996.
[12] Jens Franke, "On the factorization of RSA200", *Workshop on Special Purpose Hardware for Attacking Cryptographic Systems*, 2006.
[13] Per Leslie Jensen, Pgnf, Available at: http://pgnfs.org/
[14] A.K. Lenstra, H.W. Lenstra Jr., M.S. Manasse and J.M. Pollard, "The Number Field Sieve", *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pp. 564-572, 1990.
[15] LiDIA: A C++ Library For Computational Number Theory, Available at: http://www3.cs.stonybrook.edu/~algorith/implement/lidia/implement.shtml
[16] GGNFS - A Number Field Sieve Implementation, Available at: http://www.math.ttu.edu/~cmonico/software/ggnfs/
[17] Multiprecision Integer and Rational Arithmetic C/C++ Library, Available at: http://www.shamus.ie/
[18] NTL: A Library for doing Number Theory, Available at: http://www.shoup.net/ntl/
[19] R.P. Brent, "Some integer factorization algorithms using elliptic curves", *Australian Computer Science Communications*, Vol. 8, pp. 149-163, 1986.
[20] Pari/Gp, Available at : http://pari.math.u-bordeaux.fr/
[21] Carl Pomerance, "The Quadratic Sieve Factoring Algorithm", *Advances in Cryptology, Proceedings of Eurocrypt 84*, Vol. 209, pp. 169-182, 1985.
[22] Yet Another Factorization Utility, Available at: http://yafu.sourceforge.net/
[23] Paul Zimmermann and Bruce Dodson, "20 Years of ECM", Available at: https://hal.inria.fr/inria-00070192v2/document
[24] CADO-NFS: An Implementation of The Number Field Sieve, Available at :http://cado-nfs.gforge.inria.fr
[25] http://www.loria.fr/~zimmerma/records/top50.html.