

# Factoring Semiprimes and Possible Implications for RSA

João Carlos Leandro da Silva

Via Medole 22, Castiglione delle Stiviere (MN), Italy

**Abstract**—While most of the presently known factorization methods employ advanced mathematical tools in all phases of their respective processes the proposed algorithm is very simple and direct. The technique only applies to semiprimes or the product of two different but equal-sized primes and is based on reversing the decimal digits of the modulus. Since balanced RSA is an example of a semiprime and given that this algorithm requires little memory and is easily parallelized, we outline the basic requirements for a distributed computing experiment to factor a RSA-1024 bit modulus.

**Index terms**—Factorization, semiprimes, RSA, modulus.

## I. INTRODUCTION

The integer factorization problem [1] is a well-known topic of research within both academia and industry. It consists of finding the prime factors for any given large modulus. Currently, the best factoring algorithm is the general number field sieve or GNFS for short. On December 12, 2009 a small group of scientists [2] used the previously mentioned approach to factor a RSA-768 bit modulus, that is, a composite number with 232 decimal digits. Their achievement required more than two years of collaborative work and used many hundreds of computing machines. Hence, factoring large semiprimes is a laborious and complex task.

Other techniques exist such as the Elliptic Curve Method [3] but it is also difficult. In addition, as the numbers to be factored increase in size the respective algorithm runs progressively slower. As an alternative to all these methods we presented new ideas for both factorization and computer search in [4] which have led to the development of the following algorithm.

## II. THE ALGORITHM

The idea behind our algorithm relies on the disturbing notion that although factoring is a hard computational problem it boils down to inverting one of the most elementary operations of mathematics, more specifically, multiplication. Amazing is the fact that there exist an infinite number of integers  $Z$  such that  $\gcd[N, Z] = p$  or  $q$  where  $p$  and  $q$  are the prime factors of the modulus  $N$ , but we are still unable to find

not even one such integer within a reasonable amount of time. The most obvious solution occurs if  $Z$  is a multiple of  $p$  or  $q$ , but that is of no help since we are actually looking for either of these integers. All we know is the numerical value of  $N$ . At this point, the following questions come to mind. Is there some additional information in  $N$  that may help us? Could  $Z$  be somehow related to  $N$ ? Surprisingly, both answers are positive. It was recently noted that  $\gcd[N, (k \times \check{N}) + \Delta]$  and  $\gcd[N, (k \times \check{N}) - \Delta]$  result in non-trivial factors of  $N$  for different values of  $\Delta$  where  $\check{N}$  is the reverse of  $N$  and  $k$  is a positive integer ranging from one to infinity. The same conditions apply to  $\gcd[N, (\Delta \times \check{N}) + k]$  and  $\gcd[N, (\Delta \times \check{N}) - k]$  where  $\Delta$  represents a random number which is limited between  $\sqrt{N}$  and  $\sqrt{\check{N}}$ . Consequently, all four general relations are used simultaneous in our search.

The implementation of our algorithm is done via a single *While* loop which is interrupted whenever any of the above relations outputs a non-trivial factor. We will demonstrate that the locations of the desired factors obey the following equations:  $\alpha + p \times C$  and  $\beta + q \times D$  where  $\alpha$  and  $\beta$  are unknown starting points,  $C$  and  $D$  are both constants and  $p$  (smaller) and  $q$  (larger) are the factors of  $N$ . This shows that all of the potential targets are periodically aligned and since the algorithm searches randomly within the finite parameter space, the probability of striking any one of them is not negligible. Just as an illustrative example, given  $N = 143$  we find its prime factors by using  $\check{N} = 341$ , and evaluate  $\gcd[143, (k \times 341) + \Delta]$  and  $\gcd[143, (k \times 341) - \Delta]$ ;  $\gcd[N, (\Delta \times 341) + k]$  and finally  $\gcd[N, (\Delta \times 341) - k]$  where  $k = 1$  and, only in this particular case, we let  $\Delta$  run from one to fifty. With respect to the first and second relations, the prime factor (13) is located along the equations:  $10 + 13D$  and  $3 + 13D$ , respectively. Likewise, for the third and fourth relations, the same prime factor can be found along the equations:  $4 + 13D$  and  $9 + 13D$ , respectively. In other words, within this range there are a total of sixteen potential targets that our search can hit to find 13. Note that there are four different numerical values for the starting points.

Similarly, for  $k = 2$  the corresponding four equations would be:  $7 + 13D$ ;  $6 + 13D$ ;  $8 + 13D$  and  $5 + 13D$  for a total of fifteen potential targets to locate 13 and, once again, we get another four different numerical values for the starting points.

### III. IMPLEMENTATION

Our previous example can be implemented through the usage of a single *While* loop as shown below

```

r=1;
While[GCD[143,341+r]==1&&GCD[143,2*(341)+r]==1
&&GCD[143,341-r]==1&&GCD[143,2*(341)-r]==1
&&GCD[143,341*r+1]==1&&GCD[143,341*r+2]==1
&&GCD[143,341*r-1]==1&&GCD[143,341*r-2]==1,
r=RandomInteger[50]];
Print[GCD[143,341+r]];Print[GCD[143,2*(341)+r]];
Print[GCD[143,341-r]];Print[GCD[143,2*(341)-r]];
Print[GCD[143,341*r+1]];Print[GCD[143,341*r+2]];
Print[GCD[143,341*r-1]];Print[GCD[143,341*r-2]]

```

The corresponding *Mathematica* output is

```

1
13
1
1
1
1
1
1
1

```

where the second entry above or 13 results in one of the prime factors of 143 as expected. The other seven entries represent trivial factors all equal to one. Please note that our factoring algorithm contains all eight relations where the letter “r” stands for  $\Delta$  which is the random number limited between  $\sqrt{N}$  and  $\sqrt{N}$ . In this particular case, the numerical value of “r” is any number between one and fifty.

Several considerations must be made when trying to factor larger semiprimes on a single computer. First, the number of relations should be as large as possible, but always much smaller than either  $N$  or  $\sqrt{N}$  since the value of  $k$  is proportional to the number of potential targets. Consequently, the greater the number of potential targets, the higher is the probability that our algorithm will find one of the prime factors. Second, the range of  $\Delta$  is of fundamental importance in our search. Ideally, we would prefer that  $\Delta$  be as small as possible in order for the running time of the algorithm to be minimal. Unfortunately, by doing so we are taking the risk of using an interval with zero potential targets and therefore no possibility of finding any prime factor. If this is case the algorithm will go into an infinite loop and cause the computer never to halt. In order to avoid such situation the range for  $\Delta$  should be kept between  $\sqrt{N}$  and  $\sqrt{N}$  because within such interval we are certain that the number of potential targets is at least two thus assuring that the algorithm will terminate. Third, in general, by

increasing the number of relations we reduce the running time of the algorithm. Yet, for very large semiprimes the required number of relations is so big that a single machine no longer can deal with it. In such case, the only option is to factor using distributed computing.

### IV. DISTRIBUTED COMPUTING

Due to the fact that the present algorithm consists of basic mathematical operations such as addition, subtraction and multiplication together with the classical greatest common denominator it requires little memory. Yet, it is intensely CPU and bus bound. Another relevant feature of this method is its scalability. In other words, by adding more lines of code the efficiency of the algorithm increases making it suitable for parallel processing. Successful factorizations with higher-order semiprimes were achieved independently on both single-core Intel and double-core AMD processors without any network connection.

At this point, we outline the basic requirements for a distributed computing experiment to factor RSA-1024 bit or a 309 decimal digit modulus. Suppose we have a network with about one thousand computers connected to a main server. Each computer will run the present algorithm with one million relations for one full calendar year. In other words, the first computer will have  $k = 1$  to  $k = 1,000,000$  while the second computer ( $k = 1,000,001$  to  $k = 2,000,000$ ) and so on. This means that in total we have one billion relations being tested every minute. We are assuming that the latest multiple core processor is capable of running the given algorithm with one million relations just under one minute. Since every day has 1,440 minutes then one year is approximately 525,000 minutes. At the end of the year our network would have already calculated more than  $10^{14}$  different GCD’s (greatest common denominator) and if any resulted in a non-trivial factor, a flag would be raised and the main server would command all units to halt the parallel computation.

### V. CONCLUSIONS

In this paper we proposed a factoring algorithm specially designed for semiprimes based on new mathematical ideas. Since this method is relatively simple and scalable, we strongly believe it can be suitable for parallel processing. As a result, we outline the basic requirements for a distributed computing experiment to factor RSA-1024 bit in one full calendar year.

### REFERENCES

- [1] A. K. Lenstra, *Integer factoring*, Designs, Codes and Cryptography 19 (2000), 101-128.
- [2] Factorization of a 768-bit RSA modulus.
- [3] W. Stallings, *Cryptography and Network Security: Principles and Practices*, Pearson Educational International, New Jersey, 2003.
- [4] J. C. L. da Silva, *The Rainbow of Primes*, Freund Publishing House, Tel- Aviv, 2009.