BACKTRACKING BASED INTEGER FACTORISATION, PRIMALITY TESTING AND SQUARE ROOT CALCULATION

Mohammed Golam Kaosar

School of Computing and Mathematics
Charles Sturt University Australia
mkaosar@csu.edu.au

ABSTRACT

Breaking a big integer into two factors is a famous problem in the field of Mathematics and Cryptography for years. Many crypto-systems use such a big number as their key or part of a key with the assumption - it is too big that the fastest factorisation algorithms running on the fastest computers would take impractically long period of time to factorise. Hence, many efforts have been provided to break those crypto-systems by finding two factors of an integer for decades. In this paper, a new factorisation technique is proposed which is based on the concept of backtracking. Binary bit by bit operations are performed to find two factors of a given integer. This proposed solution can be applied in computing square root, primality test, finding prime factors of integer numbers etc. If the proposed solution is proven to be efficient enough, it may break the security of many crypto-systems. Implementation and performance comparison of the technique is kept for future research.

KEYWORDS

Information Security, Crypto-system, Factorization, Primality test, Backtracking.

1. INTRODUCTION

Integer factorisation is known as the decomposition of a composite integer number into small divisors. As for example, 91 is a composite integer which is a composition of 7 and 13, i.e. 91 = 7 x 13. Both 7 and 13 are known as the factors of 91. If the factor is a prime number, then it is known as prime factor. In the above example, both of them are prime factors. As the size of the number increases, it becomes very difficult to find its factors. Sometimes, a sophisticated algorithm running in the fastest computer may take hundreds of years to find a factor of a large number. As a matter of fact, many cryptographic algorithms, such as RSA [11], use a big number (1024 or 2048 bits) in generating keys with the assumption that, fastest technique with the help of many computers would not be able to factorise that number within a practically feasible time.

This paper proposes a new technique to decompose a composite integer into two factors using backtracking technique. Repetitive application of the proposed technique will find all possible

David C. Wyld et al. (Eds) : CCSIT, SIPP, AISC, PDCTA, NLP - 2014
pp. 77–83, 2014. © CS & IT-CSCP 2014 DOI : 10.5121/csit.2014.4207

factors of a given integer number. The proposed solution also can be used in testing the primality and finding the square root of an integer (if there exist any).

The impact of the proposed solution can be tremendous, depending upon its performance. If it can factorise a big number (as big as the number used in some crypto-systems) within a practically feasible amount of time, it would make a big change in the field of Cryptography. Further impact, implementation and performance analyses are kept for future research.

The rest of the paper is organised as follows: Section 2 presents existing solutions for factorising integer numbers in brief. Section 3 discusses the proposed solution with an example and Section 4 concludes the paper with some future research directions.

2. EXISTING SOLUTIONS

There have been many efforts proposed to factorize an integer number. The use of prime factors in crypto-systems increased much research interest to finding a practical solution to factorise a big integer. Integer factorization intrinsically leads towards the solution of primality testing and finding square root of an integer number. Some of the existing solutions are as follows:

- Trial Division: Trial division algorithm finds whether a given integer N is divisible by any positive number less than N. This is a simple and brute force approach which is very time consuming to find a solution.

- Wheel Factorisation: This is a graphical method of factorising an integer. In this method, natural numbers are marked around the wheel to form spokes of primes and their multiples.

- Rho Methods: Rho (ρ) method generates $\rho_1, \rho_2, \rho_3\ldots$ where $\rho_{i+1} = \rho_i^2 + 10,$ $\rho_1 = 1$ and parameter 10 is chosen by users. This method finds factors of N by computing gcd(N, $(\rho_2 - \rho_1)$ $(\rho_4 - \rho_2)\ldots.(\rho_{2n} - \rho_n)$). Parameter n is also chosen by user. This method initially was proposed by Pollard [8]. Some variations, improvements and optimisations of Rho methods are proposed in – [9], [2], [1] etc.

- Fermat's and Euler's Factorisation: Fermat's Factorisation technique [7] represents odd integer N as $a^2 - b^2$ where both a and b are positive integer. Thus, N is equal to $a^2 - b^2 = (a + b)(a - b)$ and factors are $(a + b)$ and $(a - b)$. In Euler's Factorisation technique [6], N is represented as $a^2 + b^2$ and $c^2 + d^2$, where a, b, c and d are positive integers.

- Other Methods: Various other methods of factorisation algorithms are: group factorisation technique of Pollard's ρ – 1 [8], William's ρ + 1 factorisation [12], Lenstra's elliptic curve factorization [5]; Dixon's factorization [3]; Quadratic sieve factorization [10] etc.

3. PROPOSED SOLUTION

This section presents the proposed solution to factorize an integer with a simple example. The solution is also extended for primality testing and square root calculation.

3.1 Factorising an Integer

Let us consider, a composite m bit integer number R, binary representation of which is $r_m, r_{m-1}, ...r_2, r_1$. Also assume the binary representation of R's two factors A and B are $a_\ell, a_{\ell-1}, ...a_2, a_1$ and $b_\ell, b_{\ell-1}, ...b_2, b_1$ respectively, where both of them are ℓ bit long. Therefore, $R = A \times B$. Their standard multiplication (also known as grade-school multiplication) is demonstrated in Fig.1. The method of multiplication to be correct, n_i has to be equal to r_i for all $i = 1$ *to* m where $m = 2\ell - 1$.

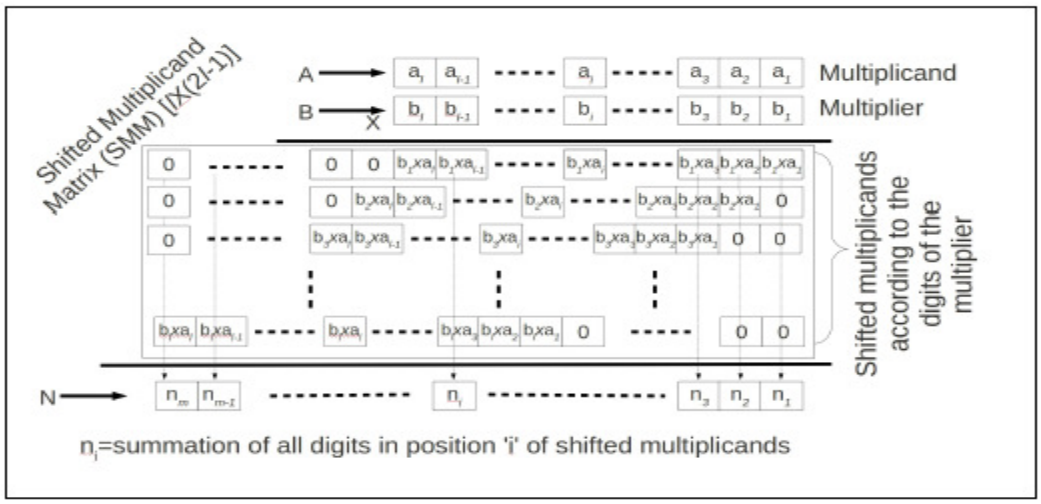

Figure 1: Binary standard multiplication of A and B to produce N

Shifted Multiplicands Matrix (SMM), as shown in Figure 2, has the dimension of $\ell \times (2\ell - 1)$ with entry as follows:

$$SMM[i][j] = 0 \qquad for\ (j < i)\ and\ (j \geqslant (\ell + i))$$
$$or \tag{1}$$
$$SMM[i][j] = b_i \times a_{j-(i-1)}$$

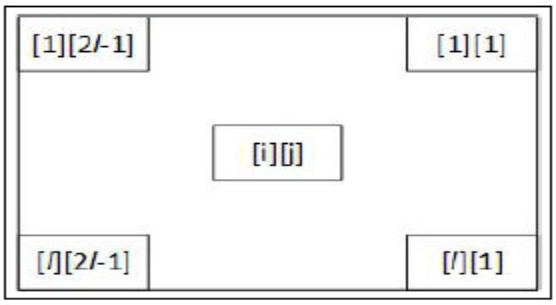

Figure 2: SMM block diagram

Each bit in N is a binary summation of bits of SMM's corresponding positions, as displayed in Equation 2. Where, S_k and C_k are the sum and carry bit of SMM entries at column k.

$$\begin{aligned} S_k &= C_{k-1} + \sum_{j=1}^{\ell} SMM[j][k] \\ C_k &= S_k/2 \\ n_k &= S_k\%2 \end{aligned} \tag{2}$$

Further detail of binary multiplication method can be found in [4].

Now an algorithm is to be developed which would choose a_i and b_i in such a way that, n_k calculated in Equation 2 becomes equal to r_k, for all $k \leq (2l - 1)$. Following example explains how a number is factorised using the proposed technique.

3.1.1 An Example

Say an integer R = 12 and hence, m = 4. We have to find two factors, A and B, or R. Binary representation of R, A, B and S M M, as discussed in Section 3.1, are now showed in step 1 of Fig. 3, where fields of A and B are empty and entries of S M M are set to 0 initially.

We need to consider every bit of R from right to left. In Step 2 we try to choose values for a1, b1 and S M M [1] [1] such that n1 calculated using Equation 2 becomes equal to r1. Therefore, S M M [1] [1] must be 0. Possible values of { a1, b1 } could be { 0,0 } or { 1,0} or { 0,1}, but cannot be {1,1}. Let us say we choose { 0, 0}. Now immediately we can set S M M [2] [2] = S M M [3] [3] = 0, since a_1 would make these values to 0 regardless of the value of b_i, i = 1 to ℓ. Similarly S M M [1] [2] = S M M [1] [3] = 0, since b1 would make these values to 0 regardless of the value of a_i, i = 1 to ℓ.

Now, we move to the next bit of R in step 3. Values of SMM in column 2 are already consistent. We have the option to choose any value for {a_2, b_2}. Say us choose {1, 1}. Therefore, S M M [2][3] can be set to 1. Similarly we consider for r_3 in Step 4. Only one position of S M M in this column is left which has to be 0 to be consistent with r_3. Choose any value for {a3, b3}. Let us say we choose {0,0}. Hence S M M [2][4] and S M M [2][5] have to be 0. But in the next column (column 4), all values become 0. Therefore, a conflict arises, since the summation could not produce a value equal to r4. Hence, a backtracking will be necessary for r3. Now, let us try with the values of {a3, b3} an {1,1} in Step 5. This time it creates conflict too. In next step, let us try with {a3, b3} - {1,0} in step 6. Now the values of A, B and S M M becomes consistent with that of R. Therefore, two factors of R would be A = 6 and B = 2.

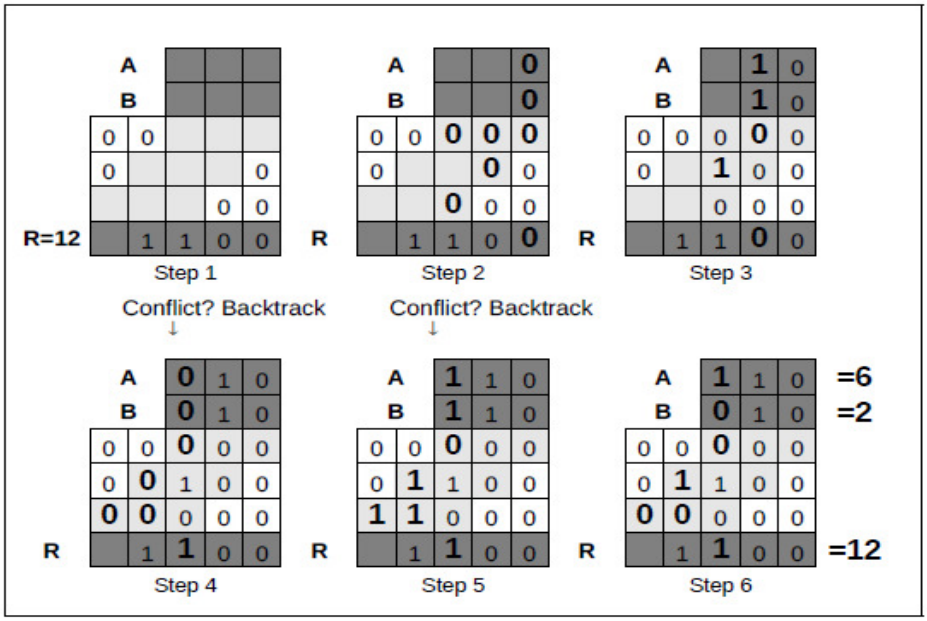

Figure 3: Steps of factorisation for the example

3.1.2 The Algorithm

The proposed solution appears in the following algorithm in a concised manner:

Algorithm 1 Factorisation of a composite integer R

input : $R = r_1, r_2, ..., r_m$
output : A *and* B
Begin
for $(i = 1$ *to* $m)$ **do**
  if $i < \ell$ **then**
    choose a_i, b_i *and* $SMM[1, 2...\ell][i]$ *such that*
    $r_i = n_i$
    if not feasible then
    $i = backtrack(i)$
  else
    compute $n_{l+1...m}$ *from rest of* SMM
     if at any point $r_i \neq n_i$ *then*
    $i = backtrack(i)$
  end if
end for
return A *and* B
End

Algorithm 2 backtrack(i)

```
input : i
output : j
Begin
for (j = i − 1 down to 1) do
      choose different values for a_j, b_j and SMM[1, 2...ℓ][j] such that
      r_j = n_j
  if r_j = n_j then
    break
  else
       j = backtrack(j)
  end if
end for
return j
End
```

3.2 Primality Test and Square Root

The proposed factorisation technique can easily be extended for the following solutions:

Square Root

If we set or require, A and B must be equal, then we get the square root of R (if it is a perfect square) in A and B. In this case, the computation even would be quicker, since more values of SMM would be pre-set.

Primality Test

If the proposed algorithm finds no solution after finishing all steps then, (R) is a prime number.

4. CONCLUSION AND FUTURE WORK

In spite of the existence of many factorisation techniques, cryptographic algorithms are still in work with the assumption that, factorising a very large number would take too long time to be practical for the crypto-system to be insecure. Therefore, a new solution in the factorisation family should draw much attention to the crypto-community. In this paper, only the algorithm is discussed in brief without considering its implementation and performance evaluation. If the proposed solution performs such that, it becomes capable to factorise those big numbers, then it may break the security of some crypto-systems, such as RSA.

In future, the algorithm can be implemented to measure its performance. It is also important to see whether this solution can factorise very big numbers used in some crypto-systems. If fails, still its performance can be compared with other existing factorisation solutions. A hybrid approach also can be thought to engage this solution with others. The algorithm itself can be improved by introducing some pruning and optimisation techniques too.

REFERENCES

[1] Richard P. Brent. An improved monte carlo factorization algorithm. BIT Numerical Mathematics, 20:176-184, 1980. 10.1007/BF01933190.

[2] Richard P. Brent. Factorization of the tenth fermat number. MATH.COMP, 68:429-451, 1999.

[3] J. D. Dixon. Asymptotically fast factorization of integers. Math. Comp.,36 (153):255260, 1981.

[4] Harris and David. Digital Design and Computer Architecture : From Gates to Processors. Elsevier, Burlington 2007, 2007.

[5] A.K. Lenstra. Fast and rigorous factorization under the generalized riemann hypothesis. Indagationes Mathematicae (Proceedings), 91(4):443-454, 1988.

[6] James Mckee. Turning euler's factoring method into a factoring algorithm. Bulletin of the London Mathematical Society, 28, 1996.

[7] James McKee. Speeding fermat's factoring method. Math. Comput.,68(228):1729-1737, October 1999.

[8] J. M. Pollard. Theorems on factorization and primality testing. Mathematical Proceedings of the Cambridge Philosophical Society, 76:521-528,1974.

[9] J. M. Pollard. A monte carlo method for factorization. BIT Numerical Mathematics, 15:331-334, 1975. 10.1007/BF01933667.

[10] Carl Pomerance. The quadratic sieve factoring algorithm. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, Advances in Cryptology, volume 209 of Lecture Notes in Computer Science, pages 169-182. Springer Berlin / Heidelberg, 1985. 10.1007/3-540-39757-417.

[11] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21(2):120-126, 1978.

[12] H. C. Williams. A p + 1 method of factoring. Math. Comput., 39:225-234,1982.

AUTHOR

Dr. Mohammed Kaosar is one of the faculty members in the School of Computing and Mathematics, Faculty of Business, Charles Sturt University, Australia. He has wide experience of teaching various ICT courses in several universities in Australia, KSA and Bangladesh. Previously, he used to work as a post-doc research fellow after his completion of PhD from the School of Engineering and Science, Victoria University Melbourne, Australia. Prior to that, Dr. Kaosar finished his MS in Computer Engineering and BSc in Computer Science and Engineering in the year of 2006 and 2001 from KSA and Bangladesh respectively. He also has experience of working in many academic, research and commercial projects. He has published good number of research papers in high quality journals and conferences including, IEEE Transactions on Knowledge and Data Engineering (TKDE), Data & Knowledge Engineering (DKE), Computer Communications, IEEE International Conference on Data Engineering (ICDE- 2012) etc. He is an active member of various professional associations including IEEE, EAI.