talenta
PUBLISHER

DATA SCIENCE
Journal of Computing and Applied Informatics

On Factoring the RSA Modulus Using Tabu Search

Ade Candra[1], Mohammad Andri Budiman[2], and Dian Rachmawati[3]

[1]College of Science and Engineering, School of Environmental Design, Kanazawa University
[2,3]Department of Computer Science, Universitas Sumatera Utara, Medan, Indonesia

Abstract. It is intuitively clear that the security of RSA cryptosystem depends on the hardness of factoring a very large integer into its two prime factors. Numerous studies about integer factorization in the field of number theory have been carried out, and as a result, lots of exact factorization algorithms, such as Fermat's factorization algorithm, quadratic sieve method, and Pollard's rho algorithm have been found. The factorization problem is in the class of NP (non-deterministic polynomial time). Tabu search is a metaheuristic in the field of artificial intelligence which is often used to solve NP and NP-hard problems; the result of this method is expected to be close-to-optimal (suboptimal). This study aims to factorize the RSA modulus into its two prime factors using tabu search by conducting experiments in Python programming language and to compare its time performance with an exact factorization algorithm, i.e. Pollard's algorithm. The primality test is done with Lehmann's algorithm.

Keyword: RSA, Tabu search, Pollard's factorization, Prime numbers, Lehmann's primality test, Python.

Abstrak. *Secara intuitif, keamanan sistem kriptografi kunci publik RSA bergantung kepada sulitnya memfaktorisasi sebuah bilangan bulat yang sangat besar menjadi dua buah faktor primanya. Penelitian mengenai faktorisasi bilangan bulat umumnya dilakukan di ranah teori bilangan dan telah menghasilkan beberapa macam algoritma eksak, seperti algoritma faktorisasi Fermat, metode saringan kuadratik, dan algoritma Pollard. Masalah faktorisasi itu sendiri termasuk dalam kelas NP (non-deterministic polynomial time). Tabu search adalah suatu metaheuristik pada ranah kecerdasan buatan yang jamak dipakai untuk menyelesaikan masalah kelas NP dan NP-hard; kategori hasil dari metode ini adalah mendekati optimal (suboptimal). Penelitian ini berupaya untuk memfaktorkan modulus RSA menjadi dua buah faktor primanya dengan menggunakan tabu search dengan pendekatan eksperimental dalam bahasa pemrograman Python dan membandingkan waktu faktorisasinya dengan salah satu algoritma eksak, yaitu algoritma Pollard. Uji keprimaan dilakukan dengan algoritma Lehmann.*

Kata Kunci: *RSA, tabu search, faktorisasi Pollard, bilangan prima, uji prima Lehmann, Python.*

Received 27 April 2017 | Revised 29 May 2017 | Accepted 30 June 2017

*Corresponding author at:
[1]College of Science and Engineering, School of Environmental Design, Kanazawa University, Japan
[2,3]Department of Computer Science, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan 20155, Indonesia
E-mail address: ade_candra@stu.kanazawa-u.ac.jp (Ade Chandra), mandrib@usu.ac.id (Mohammad Andri Budiman), dian.rachmawati@usu.ac.id (Dian Rachmawati)

Journal Homepage: https://talenta.usu.ac.id/JoCAI

1. Introduction

RSA is one of the most widely used algorithms in public key cryptosystems. The RSA cryptosystem was created by Rivest, Shamir, and Adleman [5]. The security of RSA depends on how hard it is to factor its public key, n, into two corresponding prime numbers, p and q, or its private keys. The value of n should be big enough so that any cryptanalyst that attempts to factorize n will not have ample time to do so. In 2007, it was noted that 1039 bit integer could only be factored by four hundreds computer in more than eleven months using special number field algorithm [6]. This is not surprising, since factorization is in the class of NP (nondeterministic polynomial time). Wiener [4] concluded that the modulus n and the public key e are useful to estimate a fraction that involves the private key d.

There are plenty of studies in the field of number theory that resulted numerous exact algorithms such as Fermat's factorization, Pollard-rho, quadratic sieve, etc. However, there are not enough references that could give an experimental figure as to whether or not non-exact methods such as metaheuristics could be useful in factoring large integers into their corresponding prime factors.

The cryptanalysis techniques using metaheuristic have been largely done to attack classical ciphers. For example, Garg [1] experimented with genetic algorithms, tabu search, and simulated annealing to attack transposition cipher and recommended tabu search as the most powerful method of all the three methods. Barnes and Laguna [3] suggested that genetic algorithm and simulated annealing may be combined with tabu search in a hybrid system in order to solve hard optimization problems. However, as noted before, studies that used metaheuristics to cryptanalyze public key cryptosystems (such as RSA) are not abundantly available.

In this study, we attempted to compare experimentally a metaheuristic method called tabu search and an exact algorithm named Pollard factorization algorithm in order to give some ideas about their comparative time performance in factorizing RSA public key. The experiment was done in Python programming language. The primality test being used was Lehmann's algorithm.

2. Method

Tabu search is a metaheuristic procedure to solve optimization problems that can be embedded into other heuristic procedures to avoid the trap of local minima; more about tabu search can be found in a tutorial by Glover [2].

RSA public key is n, and its private keys are p and q. From [5], we know that n = pq, so that the RSA security is based on how hard it is to factor n into p and q.

Our method is as follows. First, we experiment the factorization of RSA public key n with tabu search, keeping records of its time performances. Second, we experiment it again with Pollard's factorization algorithm. Third, we make a conclusion of whether or not tabu search is as good as Pollard's algorithm to factorize the RSA public key.

Our Python source code for the tabu search is as follows.

```
def tweak(X):
      p = getRandomPrime(X[0], X[0] + pr)
      q = getRandomPrime(n // p - qr, n // p + qr)
      if p * q > n:
            p = getRandomPrime(X[0] // 2, X[0] // 2 + pr)
            q = getRandomPrime(n // p - qr, n // p + qr)
      return [p, q]

def delta(Q):
      return abs(n - Q[0] * Q[1]) * 1.0

print "n =", n

l = 100
N = 100
max_time = 3

p = int(math.sqrt(n))
q = int(math.sqrt(n)//2)
S = [p, q]
best = S
L = []
L.append(S)

x = []
y = []
exploration = 0
x.append(exploration)
y.append(delta(S))

x2 = []
y2 = []
walk = 0
x2.append(walk)
y2.append(delta(S))

start = time.time()
stop = False
while not(stop):
      if len(L) > l:
            del L[0]
      R = tweak(S)
      exploration += 1
      x.append(exploration)
      y.append(delta(R))
      print "L =", L
      print "R =", R
      for i in range(N - 1):
            W = tweak(S)
            walk += 1
            x2.append(walk)
            y2.append(delta(W))
            print "W =", W
            if W not in L and (delta(W) < delta(R) or R in L):
                  R = W
            if delta(R) == 0.0:
                  stop = True
                  break
            if time.time() - start > max_time:
                  print "time's up"
                  stop = True
                  break
      if R not in L:
            S = R
            L.insert(0, R)
      if delta(S) < delta(best):
            best = S
      if delta(best) == 0.0:
            print "success"
            print best
            print "running time", time.time() - start, "secs"
            break
      if time.time() - start > max_time:
            print "time's up"
            break

pyp.subplot(2, 1, 1)
pyp.title('Factoring RSA Modulus n = ' + str(n) + ' with Tabu 
Search')

pyp.ylabel('delta')
pyp.plot(x, y, 'r.-')
pyp.legend(['exploration'])
pyp.grid(True)

pyp.subplot(2, 1, 2)
pyp.ylabel('delta')
pyp.plot(x2, y2, 'b.-')
pyp.legend(['walk'])
pyp.grid(True)

pyp.show()

pyp.savefig('ts-rsa.png')
```

Our Python source code for Lehmann’s algorithm used for the primality test is as follows. More about Lehman’s algorithm can be found in [7].

```
def rnd(mini, maxi):
      return random.randint(mini, maxi)

def Lehmann(p):
      k = 10
      for i in range(k):
            a = rnd(2, p - 1)
            L = pow(a, (p - 1) / 2, p)
            if L != 1 and L - p != -1:
                  return False
      return True

def getRandomPrime(mini, maxi):
      p = rnd(mini, maxi) // 2 * 2 + 1
      while not Lehmann(p):
            p = rnd(mini, maxi) // 2 * 2 + 1
      return p
```

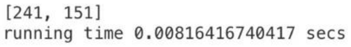
Our Python source code for Pollard. More about Pollard's factorization algorithm can be found in [8].

```
#title: Pollard's Factorization Algorithm
#purpose: Factorization of Large Numbers
#author: Mohammad Andri Budiman
#version: 0.99
#date: May 20th 2017
#time: 10:33

import math, random, time

def modexp(x, y,n):
      binary = dec2bin(y)
      z = 1
      for i in binary:
            if i == 0:
                  z = z * z % n
            else:
                  z = x * z * z % n
      return z

def dec2bin(d):
      binary = []
      while d != 0:
            binary.append(d % 2)
            d = d // 2
      binary.reverse()
      return binary

def rnd(min, max):
      return random.randint(min, max)

def gcd(m, n):
      r = m % n
      if r == 0:
            return n
      return gcd(n, r)

def Pollard(n):
      a = 2
      i = 2
      factor = [1]
      while (n % 2 == 0):
            factor.append(2)
            n = n // 2
      while (n != 1):
            if Lehmann(n):
                  factor.append(n)
                  factor.sort()
                  return factor
            a = modexp(a, i, n)
            d = gcd(a - 1, n)
            if 1 < d < n:
                  factor.append(d)
                  n = n // d
                  i = 1
            i += 1

n = 206957

start = time.time()
```

```
factor = Pollard(n)
print "n =", n
print factor[1:]
print "running time", time.time() - start, "secs"
result = 1
for i in factor:
      result = result * i
```

3. Result and Discussion

The results of factoring small digits of n with tabu search are as follows.



```
[241, 151]
running time 0.00816416740417 secs
```

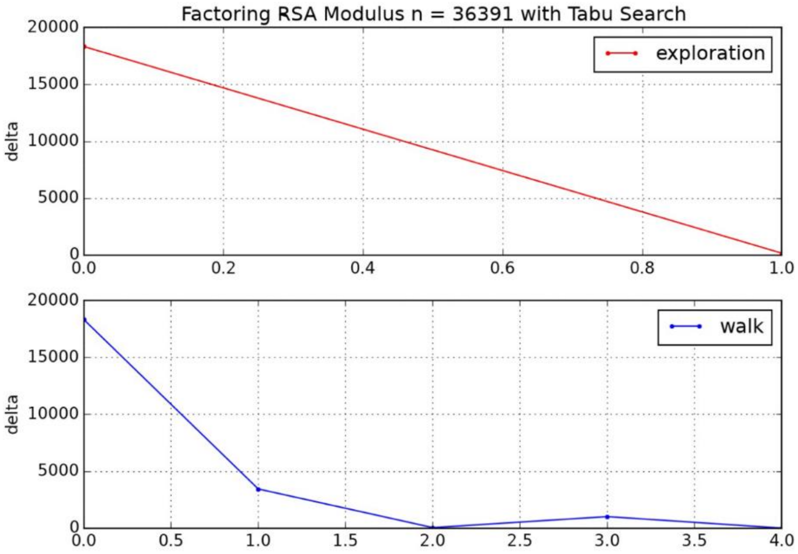
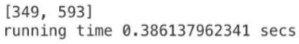


Figure 1. Factoring RSA Modulus n = 36391 with Tabu Search



```
[349, 593]
running time 0.386137962341 secs
```

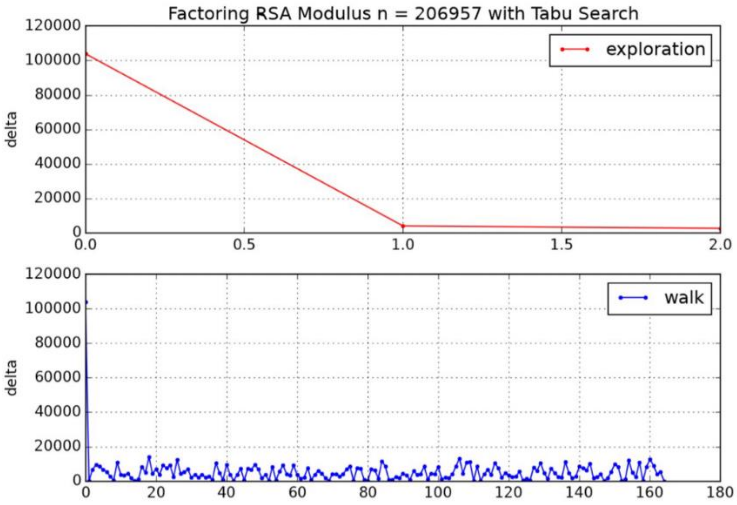
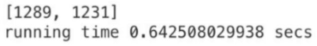


Figure 2. Factoring RSA Modulus n = 206957 with Tabu Search



```
[1289, 1231]
running time 0.642508029938 secs
```

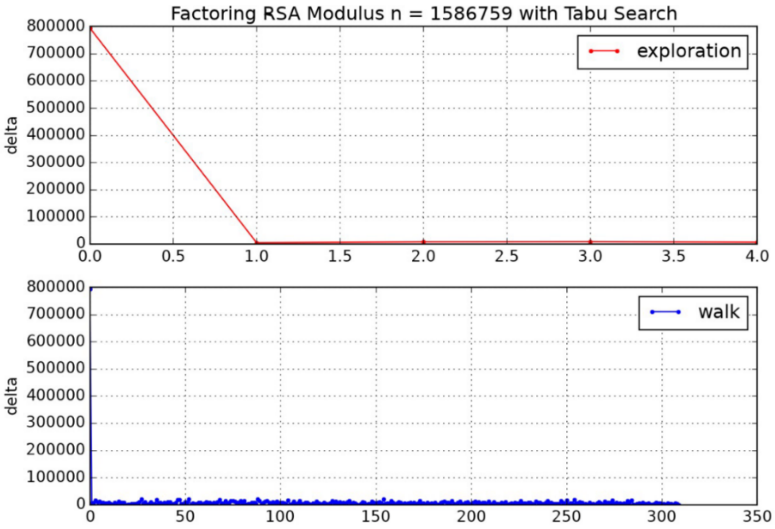


Figure 3. Factoring RSA Modulus n = 1586759 with Tabu Search

Meanwhile, to factorize n = 36391 = 151 * 241, Pollard's algorithm only needed 0.00563 second, to factor n = 206957 = 349 * 593 only 0.043162 second is needed, and to factor n = 1586759 = 1231 * 1289 only 0.012964 second is needed.

Moreover, we conduct a test on n = 56385344634735953, and Pollard's algorithm quickly determined that it is 1993 * 28291693243721 in only 0.017551 second. This very large value of n was failed to factorize with tabu search.

4. Conclusion

From our experimental findings, it is acceptable to conclude that tabu search is not a good candidate to factorize the RSA public key n into its corresponding private keys, p and q. Tabu search needed more time to factorize even small digits of n as compared to Pollard's algorithm which could factorize n with larger digits for smaller amounts of time.

REFERENCES

[1] P. Garg, "Genetic algorithms, tabu search and simulated annealing: A comparison between three approaches for the cryptanalysis of transposition cipher." *Journal of Theoretical & Applied Information Technology*, vol. 5, no. 4, 2009.

[2] F. Glover, "Tabu search: A tutorial," *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

[3] J. W. Barnes and M. Laguna, "A tabu search experience in production scheduling," *Annals of Operations Research*, vol. 41, no. 3, pp. 139–156, 1993.

[4] M. J. Wiener, "Cryptanalysis of short rsa secret exponents," *IEEE Transactions on Information theory*, vol. 36, no. 3, pp. 553–558, 1990.

[5] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[6] J. Kirk, “Researcher: Rsa 1024-bit encryption not enough,” PC World, IDG News Service, 2007.

[7] D. J. Lehmann, “On primality tests,” *SIAM Journal on Computing*, vol. 11, no. 2, pp. 374–375, 1982.

[8] J. M. Pollard, “Theorems on factorization and primality testing,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 76, no. 3. Cambridge University Press, 1974, pp. 521–528.