Lehman 素因数分解法の RSA 合成数への適用に関する研究

松尾和人研究室　　小林真理子

1　はじめに

近年、ネットワークの普及に伴い通信やメール等の暗号化が重要とされている。現在では公開鍵暗号が多く用いられるようになってきた。公開鍵暗号は公開鍵とペアである秘密鍵がなくては復号出来ないため暗号化は誰でも出来るが、復号出来るのは秘密鍵の所有者のみである。公開鍵暗号には様々な方式が存在する。代表的な公開鍵暗号に RSA 暗号がある。RSA 暗号は、大きな 2 つの素数を掛け合わした合成数を素因数分解することの困難性を利用した暗号方式である。RSA 暗号に用いられる合成数を RSA 合成数という。大きな素数によってできた RSA 合成数の素因数分解は困難である。素因数分解法には、古くから知られている Fermat 法、モンテカルロ法などが存在するが、現在では、代数体を用いる数体ふるい法、楕円曲線の群を用いる楕円曲線法などが使われている。数体ふるい法や楕円曲線法の方が一般の RSA 合成数に対しては効率的であるが、2 つの素数の差が小さいときは Fermat 法の方が効率的になりうる。Fermat 法の改良には Lawrence 法や Lehman 法が存在する。Lehman 法は Lawrence 法のアイディアを基に Fermat 法を一般の合成数に対して効率化した方法である。本研究では RSA 合成数に対する Lehman 法の効率化を検討する。

2　RSA 暗号と RSA 合成数

RSA 暗号は公開鍵暗号方式の一つである。大きな 2 素数同士の合成数は素因数分解することが困難である。RSA 暗号は素因数分解の困難性を用いた暗号方式である。このときに使われる合成数を RSA 合成数という。

p, q を $p < q$ を満足する奇素数とし、$N = pq$ とする。p のビット長と q のビット長が等しいとき N を RSA 合成数と呼ぶ。すなわち、$2^{z-1} \leq p \leq 2^z - 1$ を満足する p のビット長 z に対し $2^{z-1} \leq q \leq 2^z - 1$ を満足するとき N は RSA 合成数である。$N = pq$ が RSA 合成数のとき $q/2 < p < q$ が成立する。本研究ではこの性質を利用して Lehman 法を効率化する。

3　Fermat 法と Lawrence 法

Lehman 法は Fermat 法の改良であるが、Laurence 法で利用されたアイディアに基づいている。ここでは Fermat 法と Lawrence 法について説明する。

3.1　Fermat 法

奇素数 p, q に対して合成数 $N = pq$ とする。このとき、$N = x^2 - y^2$ となるような正整数 x, y があり、N を因数分解すると、

$$N = x^2 - y^2 = (x + y)(x - y)$$

となる。$p < q$ であるとすると、$p = (x-y), q = (x + y)$ となる。このことから x, y の値を見つけることが出来れば、N の因子 p, q を求めることが出来る。x の値を見つけるために N の平方根から順に

$$x = [N^{1/2}], x = [N^{1/2}] + 1, x = [N^{1/2}] + 2, \cdots$$

のように整数値を候補として各々について分解できるかを確認していく。$x^2 - N$ が最初の完全平方となるときの値が y^2 となり、このときの x, y の値から因子 p, q が求められる。したがって、Fermat 法は x の値が $N^{1/2}$ に近いほど効率的であり、y^2 の値が小さいほど効率的となる。したがって、p, q の差が小さいほど効率的となる。

3.2　Lawrence 法

Fermat 法を改良した方法として Lawrence 法がある。Lawrence 法は Fermat 法に対して因子の比を用いて因数分解を行う。奇合成数 $N = pq$ に対して因子 p, q の大体の比を $p : q \approx a : b$ とする。ここで a, b は互いに素な小さな正整数とする。比の一方が奇数でもう一方が偶数である場合には求める x, y の値は整数 $+\frac{1}{2}$ となる。そのため、$4abN$ を合成数として Fermat 法を用いて因数分解する。この際に Lawrence 法では比が因子に近いほど効率的となる。

4　Lehman 法

Lawrence 法では因子の比を利用するが実際にはこの値は分からない。そこで Lehman 法では Lawrence 法で用いた ab に対して $k = ab$ と置き、この値の採りうる範囲 $1 \leq k \leq r$ に対して網羅的な探索を行う。ここで r は探索精度を決めるパラメータである。比の積であるため $p : q \approx 1 : 1$ が因子同士の差では最小となる。Lehman 法では試し割り算との組み合わせで素因数分解を行う。このことから素因子の差の最大は $N^{1/2}$ より小さいことになる。上記のことから r は $1 \leq r < N^{1/2}$ を満たす値で因数分解に用いられる。Lawrence 法と同様に奇合成数 $N = pq$ となるような素数 p, q があるとする。このとき、$p < q$ である。$(N/(r+1))^{1/2} < p \leq N^{1/2}$ のとき、$x^2 - y^2 = 4kN$ となる正整数 x, y, k が存在する。このときに用いられる k は $1 \leq k \leq r$ の範囲で値を順に $k = ab$ としている。Lawrence 法より x の値は $x \equiv k + 1 \pmod 2$ となる。k が奇数のときには x の値は

整数であったので、$x \equiv k+N \pmod 4$ となる。$y^2 = x^2 - 4kN$ より y の値の範囲は以下となる。

$$0 \leq x - (4kN)^{1/2} \leq (1/(4(r+1)))(N/k)^{1/2} \quad (1)$$

このとき $p < q$ より $p = \min(\gcd(x+y,n), \gcd(x-y,n))$ となり、因子が求められる。今回の Lehman 法では RSA 合成数を用いるため試し割り算は省いている。

4.1 Lehman 法のアルゴリズム

Lehman 法の計算の流れをアルゴリズムで表す。本稿で用いた Lehman 法のアルゴリズムを Algorithm1 に示す。

Algorithm 1 Lehman 法

Input: RSA 合成数 $N = pq, r \in \mathbb{N}$
Output: p

```
while 1 ≤ k ≤ r do
    x ← [(4kN)^{1/2}]
    b ← [(1/4(r+1))(N/k)^{1/2} + 2√(kN)]
    while x ≤ b do
        if x ≡ k+1 (mod 2) then
            if k が奇数 then
                if x ≡ k+N (mod 4) then
                    y ← (x^2 − 4kN)^{1/2}
                    if y が整数 then
                        p ← min(gcd(x + y, N), gcd(x − y, N))
                        return p
        x ← x + 1
    k ← k + 1
```

4.2 Lehman 法の計算量

試し割り算法と Lehman 法から素因数分解を行うまでの計算量は全体で

$$O((N/r)^{1/2}) + \textstyle\sum_{1 \leq k \leq r} O((1/r)(N/k)^{1/2} + 1)$$

となる。試し割り算の計算量は Lehman 法の計算量と同じ値となる部分があるため全体の計算量は

$$O((N/r)^{1/2}) + O(r)$$

となるので、効率的な計算量を考える際には $O((N/r)^{1/2})$ と $O(r)$ から考える。このときの効率的な計算量は $(N/r)^{1/2} = r$ となる。このことから最良計算量は、$r = N^{1/3}$ に近いときと思われる。

5 RSA 合成数に対する Lehman 法の改良

RSA 合成数に対する Lehman 法の改良を検討する。RSA 合成数に対して Leheman 法を適用する場合も通常の合成数と同様に $O(N^{1/3})$ で素因数分解可能であり、Fermat 法より効率的である。さらに、通常の合成数に対する場合と異なり試し割り算が必要ないためその分の高速化が期待される。ここでは RSA 合成数に対して特化することで Lehman 法をさらに高速化できないか検討する。

5.1 RSA 合成数に対する改良

RSA 合成数に対して Lehman 法を用いる。RSA 合成数に対して p/q の最小値は $1/2$ である。したがって、Lehman 数列を用いた場合には $1/2$ 以上のみの値を利用すればよい。

この性質を利用し Lehman 法に対し RSA 合成数に対する以下の修正を行った。

1. Lehman 法では比 a/b を 0 から 1 の間で探索していたものを $1/2$ から 1 の間の探索とした。
2. 比 a/b に対して、Lehman 法ではパラメータ $k = ab$ を動かして探索していたが、比 a/b を直接動かして探索を行うこととした。
3. a, b を直接利用することで式 (1) を精密化し、探索範囲をより狭くすることで効率化を図った。

これらを達成するためには比 a/b が必要なため、Lehman 数列 S_r の生成方法が必要となる。

5.2 Lehman 数列の生成

Farey 数列に対する方法の類推として Lehman 数列を降順に生成する方法を求める。以下では、S_r の要素 $s_i = a_i/b_i, s_{i+1} = a_{i+1}/b_{i+1}$ を既知とし、$s_{i-1} = a_{i-1}/b_{i-1}$ を求める。

$$\begin{aligned} ta_i &= a_{i-1} + a_{i+1} \\ tb_i &= b_{i-1} + b_{i+1} \end{aligned}$$

$$t = \left\lfloor \frac{a_i b_{i+1} + a_{i+1} b_i + \sqrt{(a_i b_{i+1} - a_{i+1} b_i)^2 + 4 a_i b_i r}}{2 a_i b_i} \right\rfloor \quad (2)$$

上記の結果から a_{i-1}, b_{i-1} はそれぞれ以下の値として決定可能である。

$$a_{i-1} = t a_i - a_{i+1},\ b_{i-1} = t b_i - b_{i+1} \quad (3)$$

5.3 x のバウンドに対する改良

RSA 合成数に対して改良を行ったアルゴリズムでは $k = a_i b_i$ の値を $\frac{a_i}{b_i}$ として計算を行っている。このため Lehman 法とは違い、比の合成数からそれぞれの a_i, b_i を決める必要がなくなる。このことからそれぞれの担当範囲の a_i, b_i を直接的に用いることが出来る。Lehman 法では x の値の範囲は r を用いて決められている。そのため、x の値の範囲をより限定することが可能である。以下では、a_i, b_i が $N = pq$ に対応する領域である。

$$\left[\frac{a_i + a_{i-1}}{b_i + b_{i-1}} = \frac{a_i}{b_i} - \frac{1}{b_i(b_i + b_{i-1})}, \frac{a_i + a_{i+1}}{b_i + b_{i+1}} = \frac{a_i}{b_i} + \frac{1}{b_i(b_i + b_{i+1})} \right]$$

すなわち、p/q が上記の領域に属しているとする。また、Lehman 法と同じく以下の値となるとする。

$$m = 4kN = 4a_ib_iN,$$
$$\tau = x - \sqrt{m}$$
$$\epsilon = \tau/\sqrt{m} = x/\sqrt{m} - 1$$

上記のことから x, y の値は以下のように置ける。

$$x = (\epsilon + 1)\sqrt{m}$$
$$y = \sqrt{(\epsilon^2 + 2\epsilon)m}$$

以下ではLehman法の解析方法と同様に $p/q \leq a/b$ の場合と $p/q > a/b$ の場合に分けて考える。最初に $p/q \leq a/b$ のときLehman法より以下の関係が成立する。

$$\frac{p}{q} = \frac{a_i}{b_i}(1 + \epsilon - \sqrt{\epsilon^2 + 2\epsilon})^2$$

上記の値から $z \in \mathbb{R}_{\geq 0}$ に対して $\sqrt{1+z} \leq 1 + z/2$ より、$\delta_p = 1/\gamma$ とすると、以下の値となる。

$$\epsilon \leq \frac{1}{2(\delta_p - 2)\delta_p}$$

同様に、以下の値を評価する。

$$(\epsilon^2 + 2\epsilon) - (\epsilon + 1)\sqrt{\epsilon^2 + 2\epsilon} \leq 0$$

上記の値を計算すると、$\epsilon \geq 0$ の範囲が得られる。

次に $p/q > a/b$ のとき $p/q \leq a/b$ と同様に以下の値を用いる。

$$\frac{p}{q} = \frac{a_i}{b_i}(1 + \epsilon + \sqrt{\epsilon^2 + 2\epsilon})^2$$

γ の値を以下のように置く。

$$\gamma = \frac{1}{2a_i(b_i + b_{i+1})}$$

上記の値を用いると、$z \in \mathbb{R}_{\geq 0}$ に対して $\sqrt{1+z} \leq 1 + z/2$ より、$\delta_s = 1/\gamma$ とすると、以下の値が得られる。

$$\epsilon \leq \frac{1}{2(\delta_s + 2)\delta_s}$$

$p/q \leq a_i/b_i$ の場合と同様に計算を行うと、$\epsilon \geq 0$ の範囲が得られる。

5.4 パラメータ x の範囲

これまでの結果より以下の範囲が求められる。

$$0 \leq \epsilon \leq \max\left(\frac{1}{2(\delta_p - 2)\delta_p}, \frac{1}{2(\delta_s + 2)\delta_s}\right)$$

$$\max\left(\frac{1}{2(\delta_p - 2)\delta_p}, \frac{1}{2(\delta_s + 2)\delta_s}\right) = \frac{1}{2\min\left((\delta_p - 2)\delta_p, (\delta_s + 2)\delta_s\right)}$$

ここで、$\delta_p = 2a(b + b_p)$, $\delta_s = 2a(b + b_s)$ である。上記の結果より $x = (\epsilon + 1)\sqrt{m}$ からパラメータ x の取りうる値の範囲は以下となる。

$$\sqrt{m} \leq x \leq \sqrt{\frac{(1 + 2\delta_m)^2 kn}{\delta_m^2}} \quad (4)$$

上記で用いた δ_m^2 の値は以下の値である。

$$\delta_m = \min\left((\delta_p - 2)\delta_p, (\delta_s + 2)\delta_s\right) \quad (5)$$

5.5 改良アルゴリズム

Algorithm 2 Lehman法の改良

Input: RSA合成数 $N = pq, r \in \mathbb{N}$
Output: p

$a \leftarrow 1$
$b \leftarrow 1$
$a_1 \leftarrow 1$
$b_1 \leftarrow 1$
$a_2 \leftarrow [\sqrt{r}] - 1$
$b_2 \leftarrow [\sqrt{r}]$
if $a_2b_2 \leq r$ **then**
  $a_2 \leftarrow a_2 - 1$
  $b_2 \leftarrow b_2 - 1$
while $a_1 \neq 1$ and $b_1 \neq 2$ **do**
  $k \leftarrow ab$
  $x \leftarrow [2\sqrt{kN}]$
  $d_1 \leftarrow 2a(b + b_1)$
  $d_2 \leftarrow 2a(b + b_2)$
  if $b = 1$ **then**
    $d \leftarrow (d_2 - 2)d_2$
  else if $b_1 = 2$ **then**
    $d \leftarrow (d_1 + 2)d_1$
  else
    $d \leftarrow min((d_2 - 2)d_2, (d_1 + 2)d_1)$
  while $x \leq [\sqrt{\frac{(1+2d)^2kN}{d^2}}]$ **do**
    if $x \equiv k + 1 \pmod 2$ **then**
      if $k \equiv 0 \pmod 0$ or $x \equiv k + N \pmod 4$ **then**
        $y \leftarrow [\sqrt{x^2 - 4kN}]$
        if $y^2 = x^2 - 4kN$ **then**
          $p \leftarrow min(gcd(x + y, N), gcd(x - y, N))$
          return p
    $x \leftarrow x + 1$
  $a_1 \leftarrow a$
  $b_1 \leftarrow b$
  $a \leftarrow a_2$
  $b \leftarrow b_2$
  $t \leftarrow [\frac{ab_1 + a_1b + \sqrt{(ab_1 - a_1b)^2 + 4abr}}{2ab}]$
  $a_2 \leftarrow ta - a_1$
  $b_2 \leftarrow tb - b_1$
  return false

Algorithm2に改良アルゴリズムを示す。RSA合成数に対して x のバウンドに対する改良を行う。中間値 $s_i = a/b$ とし、前項 $s_{i-1} = a_2/b_2$、後項 $s_{i+1} = a_1/b_1$ とする。降順に計算を行うため中間値を 1/1 とする。そのため 1,2 で中間値の値の代入を行う。後項は 1 以上の値がないため、3,4 で値を 1 としておく。前項は r の値により求めることが可能である。$ab \leq r$ のとりうる値の組み合わせで 1 に近い値は $[\sqrt{r} - 1]/[\sqrt{r}]$ である。そのため、5,6 で前項

の値は $a_2 = [\sqrt{r}] - 1, b_2 = [\sqrt{r}]$ とおける。7 で前項の値が範囲内であることを確認する。仮に 7 で前項が $r > a_2 b_2$ となってしまった場合には 8,9 でそれぞれの値を減らす。前項、中間値、後項の値から降順に Lehman 法の計算を行う。降順で計算を行うため 10 でループ計算を行う。このとき、中間値は 1/2 まで計算を行う。そのため、後項が 1/2 となった場合にはすでに中間値として計算を行ったことになる。このことから 10 で $a_1 = 1, b_1 = 2$ のときは計算を終了する。10 で範囲内である場合は 11 で中間値を $k = ab$ として、Lehman 法を用いて因数分解を行う。11 から 20 は Lehman 法のアルゴリズムと同じである。その結果、因数分解出来ない場合は次の項を試す。降順であるため前項を中間値とし、用いた中間値を後項とする。そのため 29,30 で先に中間値を後項として値を格納する。次に 31,32 で前項を中間値として格納する。前項の値は Lehman 数列の生成の式 (2)、(3) より中間値と後項から計算可能である。式 (2) の値 t を 33 で計算する。33 で求めた値を用いて 34,35 で前項の計算を式 (3) を用いて計算する。前項を計算した後にまた同様に中間値を用いて 10 に戻って Lehman 法を試す。この計算を $s_i = 1/2$ の値まで試していく。Lehman 法を行うループでは 11,12 で用いる値の計算を行う。Lehman 法との x の値の範囲とは違い、式 (4) の x の範囲を用いる。x の値の範囲を決めるために δ_m の値が必要である。そのため 13 から 20 で δ_m の計算を行う。δ_m の値を求めるには式 (5) を用いる。13,14 で式 (5) で用いられる δ_p, δ_s の計算を行う。式 (5) より δ_m の値は最小値をとるため 15 から 18 までは最小値となる値のみの計算を行う。15,17 の値でない場合は 20 で式 (5) の計算を行う。15 から 20 までの計算で求められた δ_m の値から式 (4) を用いて x の値の範囲を計算する。21 から 28 までは決定した x の値の範囲内で Lehman 法を行う。そのため、22 から 28 は Lehman 法と同様である。

6 実験

Lehman 法の RSA 合成数に対する改良を行った。それぞれ RSA 合成数に対しての改良を行った結果を改良 1 とし、さらに x のバウンドに対する改良を加えた結果を改良 2 とする。実際に効率化となるかプログラムを実装し、計測結果を比較する必要がある。Sage 上で Lehman 法、改良 1、改良 2 を実装する。実装には、Core i7-3820 3.6GHz 4Core,64GB-RAM, Sage バージョン 7.4 を利用した。効率化されているかを検討する上でループ回数と計算時間をそれぞれ計測した。ループ回数は主な計算量となっているため回数により計算量の比較が行える。さらに計算時間の計測も行い、計算にかかる時間の比較も行った。尚、比較には最悪の計算量で比較を行った。そのため素因数分解が出来たとしても、計算を終了せずに最後まで計算を行っている。

6.1 ループ回数による比較

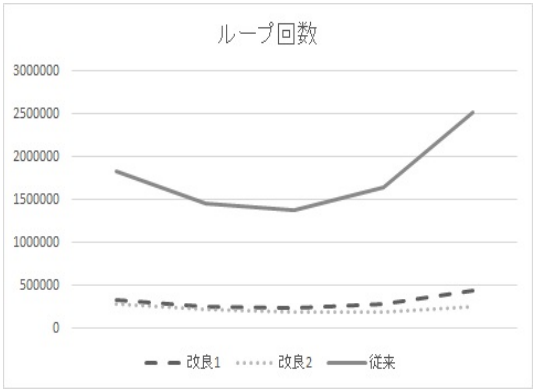

図 1: 64 ビットの合成数に対する Lehman 法と改良法のループ回数の比較

Sage 上で実装した Lehman 法と改良の結果の比較を行う。比較の際に最適な r を求める必要がある。合成数を固定し、ループ回数が少ない r の値を求める。今回の結果をグラフとしてまとめた。図 1 では Lehman 法、改良 1、改良 2 の 3 つに対して比較を行った。図の表記上 Lehman 法は従来としている。それぞれランダムに 5 回計測を行い、その平均値を用いている。本章でのグラフの見方として縦軸はループ回数である。横軸の中心に最適な値が来るように r の値を用いて計測している。そのため、図 1 では Lehman 法の r の値は 2^{17} から 2^{21} の値を計測し、改良 1 の r の値は 2^{18} から 2^{22} の値を計測し、改良 2 の r の値は 2^{17} から 2^{21} の値を計測した結果である。今回は計測の一部として 64 ビットの合成数に対しての結果を示す。

6.2 最適な r による比較

Lehman 法では合成数の約 $\frac{1}{3}$ 乗が最適な r の値であるとしている。そのため最初に合成数を固定し、r の値を変化させ最適な r の値を検討を行った。そのうえで、最適な r の値を用いて合成数のビット数を変更させ比較を行った。ループ回数の比較した結果を図 2 に示す。

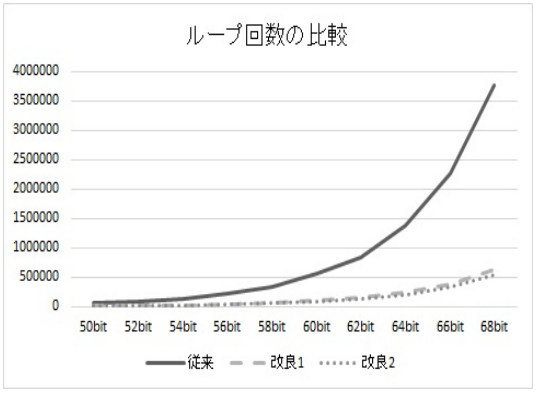

図 2: Lehman 法と改良法のループ回数の比較

2 つの改良法と比較すると、従来の Lehman 法の方が圧倒的にループ回数が多くなっている。計算時間の比較は図 3 にて示す。

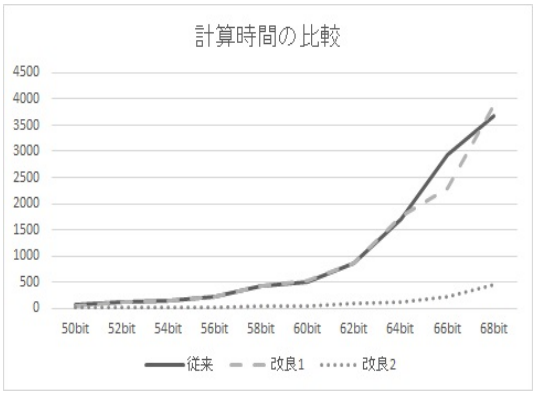

図 3: Lehman 法と改良法のループ回数の比較

6.3 考察

Lehman 法、改良 1、改良 2 を実装し比較を行った。比較には合成数のビット数を変更して計測した。その際にそれぞれの合成数のビット数毎に最適な r の値で比較を行う必要がある。最適な r を選択するために合成数を固定して r の値を変えて計測を行った。その結果、図 1 の合成数では Lehman 法の中心の値は $r = 2^{19}$、改良 1 の中心の値は $r = 2^{20}$、改良 2 の中心の値は $r = 2^{19}$ であったためこの値で最適値となった。その結果から Lehman 法、改良 1、改良 2 はほぼ同様の r の値で最適となった。この結果からさらに合成数のビット数を変更して比較を行う必要がある。そのため、合成数毎に最適な r の値で計測して Lehman 法と比較した。同ビット数の合成数を複数計測を行い、ループ回数と計算時間を比較たした。図 2 よりループ回数は改良 1、改良 2 ともに Lehman 法よりも減少した。改良 1 と Lehman 法を比較すると、50 ビットで Lehman 法が 50037.6 に対し、改良 1 は 8546.4 となり約 1/6 となった。同様に他のビット数で比較を行っても約 1/6 程度となった。改良 2 と Lehman 法を比較すると、50 ビットで Lehman 法が 50037.6 に対し、改良 2 は 7270 となり約 1/7 となった。同様に他のビット数で比較を行っても約 1/7 程度となった。次に計算時間を図 3 より検討する。図 3 の結果より、計算時間では改良 1 が Lehman 法とほぼ同じ時間となってしまった。改良 2 は計算時間も Lehman 法より減少し、Lehman 法と比較すると、50 ビットで Lehman 法は $53.9s$ に対し、改良 2 は $3.45s$ となり約 15 から 16 分の 1 程度となった。計算時間に関してはビット数毎に異なる結果となった。60 ビットでは Lehman 法は $513s$ に対し、改良 2 は $41.46s$ となり約 12 分の 1 程度となった。改良 1 が Lehman 法とほぼ同様の計算時間となったことには次の項の探索に時間がかかっていると考えられる。改良 2 に関してループ回数及び計算時間の両方に関して減少したため改良出来たと考えられる。また、ループ回数に関しては Lehman 法と改良 1 では 1 : 6 となり、Lehman 法と改良 2 では 1 : 7 となったため改良の結果から Lehman 法の大体のループ回数を予測可能と考えられる。

7 まとめ

実際の Lehman 法の改良には RSA 合成数に対して検討を行った。RSA 合成数の際に計算の必要のない領域を省くことで改良を行った。実際に改良した方法を実装した。実装により Lehman 法からどの程度改良されたか検討することによって改良を検討した。最適な r の値を用いて合成数のビット数を変化させ、Lehman 法と改良法の比較を行った。合成数のビット数の変化により部分的な改良ではなく、どのビット数の合成数においても改良出来たことを確認した。比較の結果、改良していくに連れてループ回数は減少した。しかしながら、改良 1 に関しては計算時間が本来の Lehman 法とほぼ同じとなった。このため、計算回数は減少したが一つの計算にかかるコストが多くなってしまった。改良 2 においてはループ回数も計算時間も減少することが出来た。これには改良 1 においての無駄な計算を省いたことによると考えられる。なお、本来の Lehman 法は途中で計算結果が出力しなくなった。これには膨大なループ回数とそれに伴う計算時間によるものと考えられる。改良 1 に関してはループ回数が減っているため計算可能であるが時間がかかってしまう。そのため改良 2 が RSA 合成数に対しては最も効率的である。

参考文献

[1] R. Lehman,“Factoring Large Integers,” Math. Comp., 28,1974, 637-646

[2] R. Crandall, C. Pomerance, 和田 秀夫 (監訳),“素数全書,” 朝倉書店, 2010, 248,249

[3] F. Lawrence,"Factorisation of numbers," Messenger of Math., 24, 1895, 100-109.

[4] G. Hardy, E. Wright,"An Introduction to the Theory of Numbers, 4th ed," Oxford Univ. Press, London, 1960.

[5] R. Graham, D. Knuth, O Ptashinik,"Concrete Mathematics," Addison Wesley, 2nd edition, 1994.