# Examples of RSA cipher by Mathematica and Maxima

Koichiro OHTAKE
Department of Mathematics, Faculty of Education, Gunma University

(Accepted on September 30th, 2015)

**Abstract**

*RSA cipher is necessary to hide important information. But its principle is very simple. In this paper we show an practical example of RSA cipher by Mathematica and Maxima. Programs are very short. If we can easily get big prime numbers RSA cipher will be easy and safe to use. We will show how to get big prime numbers. So it will be interesting for readers to use RSA cipher.*

## 1 Big prime numbers as you like

Solovay-Strassen and Miller-Rabin primality tests provides primality test functions such as **PrimeQ[ ]** of Mathematica and **primep( )** of Maxima. Since there are infinitely many prime numbers it is easy to find prime numbers of arbitrary digits by using these primality test functions.

The following is a Mathematica program to produce a prime number of n digits.

```
gp[n_] := Module[{i, a},
    i = 1;
    a = RandomInteger[{1, 9}];
    While[i < n, a = 10 a + RandomInteger[{0, 9}]; i++];
    If[Mod[a, 2] == 0, a -= 1];
    While[! PrimeQ[a], a += 2];
    Return[a]]

gp[100]
```

```
1753867441330041362359753700908022884392952056122773418270141020698661806297727913335068687192840371
```

The next is a Maxima program of the same function.

```
gp(n):=block([a:0,i:1],
    if n=1 then (while a<3 do(a:random(10))) else (while a=0 do(a:random(10))),
    while i<n do(a:10*a+random(10), i:i+1),
    if mod(a,2)=0 then a:a-1,
    while not primep(a) do(a:a+2),
    return(a))$

gp(100);
```

**22454195835506909476993966630129986533784650765472830626674691682022135299169088839063230994457622 87**

# 2 Review of RSA cipher

Let $p$, $q$ be prime numbers and let $n = pq$, $\ell = (p-1)(q-1)$. Let $e > 0$ be an integer such that $(e, \ell) = 1$, where $(x, y)$ denotes the greatest common divisor of $x$ and $y$. Then there exists an integer $d > 0$ such that $de \equiv 1 \pmod{\ell}$. Let $S_n = \{a \in \boldsymbol{Z} \mid 0 \leqslant a \leqslant n-1\}$, where $\boldsymbol{Z}$ is the set of integers. $E : S_n \to S_n$ is defined via $E(a) \equiv a^e \pmod{n}$. Similarly $D$ is defined via $D(a) \equiv a^d \pmod{n}$. Then $E$ and $D$ are relatively inverse functions. This is the principle of RSA cipher.

Let $n_1 = p_1q_1$, $n_2 = p_2q_2$ and suppose $n_1 > n_2$. Let $E_i$, $D_i : S_i \to S_i$ $(i = 1, 2)$, where we put $S_i$ instead of $S_{n_i}$. Then compositions $D_1E_2$ and $E_1D_2$ are allowed but $D_2E_1$ and $E_2D_1$ are not allowed. The reason is the following. Let $a \in S_2$. Then $E_2(a) \in S_2$. Since $n_1 > n_2$ $E_2(a) \in S_1$ holds. Thus $E_1(D_1(E_2(a))) = E_2(a)$ and so $(D_2E_1)(D_1E_2)(a) = a$ holds. On the other hand, if we use $E_2D_1$ then $D_1(a) \notin S_2$ may happen (such an $a$ exists). In this case $D_1(a) \equiv b \pmod{n_2}$, where $0 \leqslant b \leqslant n_2 - 1$ and in particular $D_1(a) \neq b$. Then $E_2(D_1(a)) = E_2(b)$ and $E_1(D_2(E_2(D_1(a)))) = E_1(D_2(E_2(b))) = E_1(b)$. If $E_1(b) = a$ then $b = D_1(E_1(b)) = D_1(a)$. This is a contradiction. This means $E_1D_2$ is not the inverse of $E_2D_1$.

If $n_1 = n_2$ then the situation becomes rather simple, because compositions of functions are commutative. In one group it is enough to use one $n$. An administrator will serve $e$ and $d$ to each user. $d$ is like a password. But a user does not need to remember $d$. It is only necessary to keep the function $D$.

# 3 Strings and numbers

Since RSA cipher is to transform a number to another number, it is necessary to convert characters to numbers. Each character has a code number. Thus we can use those code numbers. The most simple way is to repeat the following: [character $\to$ number $\overset{RSA}{\to}$ number $\to$ character]. Actually we need to transform a string to a string by one cycle.

Mathematica has functions to transform a string to a lists of numbers. It is **ToCharacterCode[ ]** and its inverse **FromCharacterCode[ ]**. These functions are available to Japanese letters. We will make RSA functions (enciphering and deciphering functions) E and D as functions from lists of numbers to lists of numbers. If we take RSA functions like this then we can consider strings and numbers separately. This means that if a certain system can use functions to change a character to its code number and vice versa then we can change strings to lists of numbers. We can use RSA functions to these lists of numbers in another system like Mathematica or Maxima. This is very convenient and useful for security.

We explain this method more concretely. Let $str =$ "$c_1c_2\cdots c_r$" be a string. Let $a_i$ be the character code of $c_i$ and put $lst = \{a_1, a_2, \ldots, a_r\}$. Since $c_i$ is a character, $a_i$ is a positive integer. Actually $1 \leqslant a_i < 256^2 = 65536$ (actually $a_i < 60000$). We put $m = 60000$. To $lst$ we associate a number $num$ as $num = a_1m^{r-1} + a_2m^{r-2} + \cdots + a_r$. Let $n$ be a number used in RSA ($n = pq$). Then it is necessary that $num < n$. Since $num < m^r$ it is enough to satisfy $m^r \leqslant n$. $n$ is greater than 200 digits. So it is enough to put $r = 40$. Let $mun$ be a transformed number by RSA of $num$. Let $mun = b_1m^{t-1} + b_2m^{t-2} + \cdots + b_t$ and put $tsl = \{b_1, b_2, \ldots, b_t\}$. Then we decide $E(lst) = tsl$ or $D(lst) = tsl$. If $r > 40$ then we insert

$m$ for every 40 numbers, that is $lst \to \{a_1, \ldots, a_{40}, m, a_{41}, \ldots\}$. We transform the first 40 numbers, the next 40 numbers and so on. By this way we can transform a string of any length.

# 4 Main programs

We divide RSA cipher to two parts. One is the main program which includes the RSA function. And the another is the type of enciphering and deciphering functions. Enciphering and deciphering functions are essentially the same. In this section we show main programs of Mathematica and Maxima.

First we show the Mathematica program.

```
kugiri = 60000; (* 65536, 128 *)
jisuu = 40; (* 90 *)

adic[m_, n_] := Module[{k, r, i, j, a, c},
      k = m;
      i = 0; a[0] = 1;
      While[a[i] <= k, i++; a[i] = n^ i];
      r = i - 1; j = 0;
      While[j <= r,
        c[r - j] = IntegerPart[k/a[r - j]]; k = k - c[r - j] a[r - j]; j++];
      b = {};
      For[i = 0, i ¡= r, i++, b = Append[b, c[r - i]]];
      Return[b]];

exchg[b_] := Module[{l, j, num},
      l = Length[b];
      num = 0;
      For[j = 1, j <= l, j++, num = num + b[[j]]*kugiri^ (l - j)];
      Return[num];];

tomoji[seq_] := Module[{a, i, l},
      a = {}; l = Length[seq];
      For[i = 1, i <= l, i++,
        If[! seq[[i]] == kugiri, a = Append[a, seq[[i]]]]];
      Return[FromCharacterCode[a]]];

tocode[mojiseq_] := Return[ToCharacterCode[mojiseq]];

cleandat[seq_] := Module[{a, i, l},
      a = {}; l = Length[seq];
      For[i = 1, i <= l, i++,
        If[! seq[[i]] == kugiri, a = Append[a, seq[[i]]]]];
      Return[a]];

rsa[n_, e_, lst_] := Module[{i, j, k, l, b, num, mun, seq, mojilst},
      mojilst = lst;
      k = Length[mojilst]; b = {};
      If[MemberQ[mojilst, kugiri], b = mojilst,
```

```
    If[k <= jisuu, b = Append[mojilst, kugiri],
      For[j = 1, j <= k, j++,
        If[Mod[j, jisuu] == 0, b = Append[b, mojilst[[j]]];
      b = Append[b, kugiri],
      b = Append[b, mojilst[[j]]]]]];
  mojilst = b; k = Length[mojilst];
    If[! mojilst[[k]] == kugiri, mojilst = Append[mojilst, kugiri]; k++];
  b = {}; seq = {};
  For[j = 1, j <= k, j++,
    If[mojilst[[j]] == kugiri,
      num = exchg[b];
    mun = PowerMod[num, e, n];
      b = adic[mun, kugiri];
      l = Length[b];
      For[i = 1, i <= l, i++, seq = Append[seq, b[[i]]]];
      seq = Append[seq, kugiri]; b = {},
      b = Append[b, mojilst[[j]]]]];
  Return[seq]];
```

In the above program we explain the meaning of each function.

**adic[m_,n_]**: To express the number m to an n-adic integer as a list of numbers.

**exchg[b_]**: To express b to an integer, where b is a list of numbers expressed as a kugiri-adic integer.

**tomoji[seq_]**: To transform a list of numbers to a string.

**tocode[mojiseq_]**: To transform a string to a list of character codes.

**cleandat[seq_]**: To delete a break mark (= kugiri).

**rsa[a_,e_,lst_]**: To transform a list of numbers to another list of numbers by RSA cipher.

Next we show the Maxima program.

```
kugiri :128/* 65536 */
jisuu : 90/* 40 */

adic(m,n):=block([a,b,c,i,j,k,r],
       k:m,
       i:0, a[0]:1,
       while a[i]<=k do(i:i+1, a[i]:n^ i),
       r:i-1, j:0,
      while j<=r do(c[r-j]:quotient(k,a[r-j]),k:k-c[r-j]*a[r-j],j:j+1),
       b:[ ],
       for i:0 thru r do(b:endcons(c[r-i],b)),
       return(b))$

exchg(b):=block([l,j,num],
       l:length(b),
       num:0,
       for j:1 thru l do(num:num+b[j]*kugiri^ (l-j)),
```

```
        return(num))$

tocode(s):=block([str,b,i,k],
        str:charlist(s),
        k:length(str),b:[ ],
        for i:1 thru k do(b:endcons(cint(str[i]),b)),
        return(b))$

tomoji(b):=block([s,i,k],
        k:length(b),
        s:"",
       for i:1 thru k do(if b[i]#kugiri then s:concat(s,ascii(b[i]))),
        return(s))$

cleandat(b):=block([a,k],
        k:length(b),
        a:[ ],
        for i:1 thru k do(if b[i]#kugiri then a:endcons(b[i],a)),
        return(a))$

rsa(n,e,lst):=block([b,i,j,k,l,mlst,num,mun,seq],
        mlst:lst,
        k:length(mlst), b:[ ],
        if member(kugiri,mlst) then b:mlst else
          if k<=jisuu then b:endcons(kugiri,mlst) else
              for j:1 thru k do(
                  if mod(j,jisuu)=0 then (b:endcons(mlst[j],b), b:endcons(kugiri,b))
                     else b:endcons(mlst[j],b)),
        mlst:b, k:length(mlst),
        if mlst[k]#kugiri then (mlst:endcons(kugiri,mlst), k:k+1),
        b:[ ],seq:[ ],
        for j:1 thru k do(if mlst[j]=kugiri then
              (num:exchg(b),
              mun:power_mod(num,e,n),
              b:adic(mun,kugiri),
           l:length(b),
              for i:1 thru l do(seq:endcons(b[i],seq)),
              seq:endcons(kugiri,seq),
              b:[ ]) else b:endcons(mlst[j],b)),
        return(seq))$
```

# 5 Enciphering and deciphering functions

In this section we proceed to exchange texts between takahashi and ootake. For generality we take different numbers of $n$. For takahashi we take primes of 80 and 120 digits and for ootake we take primes of 81 and 121 digits. First we make takahashi's functions.

**gp[80]**

**887331847455955278887272018132621464421454574826206725549599045647288815297238 19**

**gp[120]**

**714883838653680459247271545778397954219942101943941729295281104484257505899126529817419739510198086770386074352392068043 9**

So let $p$ = 887331847455955278887272018132621464421454574826206725549599045647288 81529723819 and $q$ = 714883838653680459247271545778397954219942101943941729295281104 482575058991265298174197395101980867703860743523920680439. Then $n = pq$ and $\ell = (p-1)(q-1)$. Let $d = 1628293$. We need to find $e$ such that $de \equiv 1 \pmod{\ell}$.

**PowerMod[1628293,-1,l]**

**274099894010065012005003992947148769666714210935259324649460341741824625835460611795857493210014536304762356469466698923049793539199524364231212693622023259015850811340622467245126544104099618928461 67**

This value is $e$. Now takahashi's open key function is the following.

```
takahashiE[lst_] := Module[{n, e, mojilst},
    n = 6343391972689753352770321361979946835928571763422225703727522902
    211844123430869141509682489536928942833279393508601792610796205016
    141013509127016366199827632242288416545943031567145567458394542256
    76541;
    e = 2740998940100650120050039929471487696667142109352593246494603417
    418246258354606117958574932100145363047623564694666989230497935391995
    919952436423121269362202325901585081134062246724512654410409961892
    84616;
     mojilst = rsa[n, e, lst];
     mojilst];
```

If we replace $e$ by $d$ and the value of $e$ by $d = 1618283$ then we get takahashi's secret key function **takahashiD**[s_].

For ootake's functions we take $n$ = 210931373164800187001569675988206586394008830364 409783939709263581769267116922924636766290657295098024875641854685018517362905546553622807138846700028891601576291553884138363051714600358081503652347923, $e$ = 148462 973396254282883288761183066006435307838396639762699880270255592808417377028606225385249567073443407718849076824215887918162607733345445319055633685803555428498680989968042326651848795611645379837 7 and $d$ = 138976433.

Next we show Maxima programs. Values are the same as the cases of Mathematica.

```
takahashiE(lst):=block([n,e,codelst],
    n:6343391972689753352770321361979946835928571763422225703727522902
    118441234308691415096824895369289428332793935086017926107962050161
    410135091270163661998276322422884165459430315671455674583945422567
    6541,
```

```
e:27409989401006501200500399294714876966671421093525932464946034174182462583546061179585749321001453630476235646946669892304979353919952436423121269362202325901585081134062246724512654410409961892846167,
 codelst:rsa(n,e,lst),
 return(codelst))$
```

We omit to show other functions.

# 6 Results of run

Since takahashi's $n$ < ootake's $n$, we should use **ootakeE[takahashiD[s]]** or **ootakeD[takahashiE[s]]** when we need to encipher. On the other hand we should use **takahashiD[ootakeE[s]]** or **takahashiE[ootakeD[s]]** when we neet to decipher.

Since Mathematica is available to Japanese, we can encode Japanese sentence by Mathematica. Next we transform its sequence by RSA cipher in Maxima. Finally we make Japnese sentence by decoding a sequence in Mathematica.

Before we run the main program of Maxima we change the value of **kugiri** to 60000 and **jisuu** to 40. Then we run the main program. Next we run **ootakeE(takahashiD(lst0))** and get the following sequence of numbers. Here **lst0** is a sequence of numbers applied **tocode[ ]** to a string in Mathematica ({ } is changed to [ ] for the sake of Maxima).

lst:[1,33846,40768,42785,10655,949,6485,33723,10532,7501,61,14085,14020,26125, 36820,29160, 31765,33732,50786,18999,29028,13881,20816,10532,50075,29093,22077, 48215,24606,35886,437 74,19262,59668,13231,19482,4647,51706,34903,29594,42075, 57027,34512,37164,60000,4,17913, 33227,44613,53229,33082,9458,4595,28731,36166 ,51081,4315,50950,44772,7291,27478,40176,3 4437,48519,1483,27664,197,7975,1757,3966,32249,22990,45241,29407,22714, 45178,40516,34334, 4833,22169,33360,53640,9487,20105,24640, 25895,27384,51745,60000,2,24189,625,21307,27021, 26147,43793,14099,59078,15324,10526,5569,47271, 23628,28000,57370,16668,28324,57707,665, 13471,52654,58283,50027,9191,17036,40368,35344,7556,36913 ,3846,47357,7241,33131,8425 ,16 970,18364,47251,19882,40015,58914,47143,49830,60000,2,193,57678, 16280,49699, 6260,47185, 7366,33449,2384,46628,12808,16157,40641,11856,33412,21320, 53545,18627,18209 ,27417,52641 ,31616,12054,9511,44961,53931,4636,17013,29428,31804,19626 ,8520,43111,97,54261,57250, 702 7,35215,2368,4815,48712,14921,60000,1,33586,24898 ,34542,12679,16720,52744,20317,13011,14 81,43443,37316,36429,20305,3784,42771,28 440,29282,5540,7625,25448,16874,15189,45668,2010 0,53008,44923,6189,23706,20223, 15563,38099,4959,16137,2937,18923,1400,27903,52436,49039, 30173,46283,33279,60000]

We run **cleandat(takahashiE(ootakeD(lst)))**.

Then we get the following sequence (we changed [ ] to { } for the sake of Mathematica).

lst2:{49,57,55,55,24180,12395,83,111,108,111,118,101,121,12392,83,116, 114,97,115,115,101,110, 12395,12424,12387,12390,32032,25968,12398,21028,23450,12395, 38306,12377,12427,30011,2639 9,30340,12394,35542,25991,12364,30330,34920, 10,12373,12428,12414,12375,12383,12290,12381, 12398,24460,82,97,98,105,110, 12395,12424,12387,12390,12424,12426,31934,24230,12398,39640, 12356,32032, 25968,21028,23450,27861,12364,30330,34920,12373,12428,12414,12375,12383, 1229 0,10,24444,12425,12398,32080,26524,12395,12424,12387,12390,31169,12383, 12385,12399,20309, 30334,26689,12398,32032,25968,12418,31777,21336,12395, 35211,12388,12369,12425,12428,1242

7,12424,12358,12395,12394,12426,12414, 12375,10,12383,12290,12371,12371,12391,20351,12431, 12428,12390,12356, 12427,82,83,65,26263,21495,12399,12371,12398,12424,12358,12395,22823, 12 365,12356,32032,25968,12434,20351,12358,12371,12392,12395,12424,12387, 12390,23433,10,2084 0,24615,12364,20445,35388,12373,12428,12427,12398, 12391,12377,12290}

Finally we get the Japanese plaintext in Mathematica by the following.

**tomoji[lst2]**

1977年に Solovay と Strassen によって素数の判定に関する画期的な論文が発表されました。その後 Rabin によってより精度の高い素数判定法が発表されました。彼らの結果によって私たちは何百桁の素数も簡単に見つけられるようになりました。ここで使われている RSA 暗号はこのように大きい素数を使うことによって安全性が保証されるのです。

We translate this to English.

In 1977 Solovay and Strassen published an epoch-making paper about primality test. After that Rabin proved more improved result. By their results we can find arbitrary digits of prime numbers. The safety of RSA cipher is guaranteed by using such big prime numbers.

# 7 Appendix

Recently we have various IDs and passwords. It is hard to remember all passwords. If we transform these passwords by RSA cipher then it is safe to put them closely to us. So we show an easy way to transform these passwords by RSA cipher. We do not use character codes. We use only 91 characters which can be input from keyboard. We number these characters orderly. Suppose a string **s** is given. Let **str** be the character list of **s**. Let **str** $= [b[1], \ldots, b[\ell]]$. And let $a[i]$ =**(the order of** $b[i]$**) − 1**. Let **num**$= a[1] \times 91^{\ell-1} + \cdots + a[\ell]$. It should be noted that $a[1] \neq 0$. In other words the head of **s** must not be the first letter (here "`"). **mun** = **power_mod(num,e,n)** is an RSA transformation. **mun** corresponds to a string if we pursue the opposite direction of **string → number**. Maxima program is the following.

```
batchload("~/Documents/passwdkanri/SecretNumbers.wxm")$

/* Max string length is 50. Don't forget this. */

lst:[" ` ", " ", "=", "^", "~", "_", ".", ",", ";", ":", "A", "B", "C", "D",
     "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R",
     "S", "T", "U", "V", "W", "X", "Y", "Z", "a", "b", "c", "d", "e", "f",
     "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t",
     "u", "v", "w", "x", "y", "z", "1", "2", "3", "4", "5", "6", "7", "8",
     "9", "0", "+", "-", "*", "/", "!", "#", "$", "%", "&",
     "[", "]", "{", "}", "<", ">", "?", "","(",")"]$

adic(m):=block([k,r,c,i,a,j],
     if m>=0 then k:m else k:-m,
     if k=0 then (r:0,c[0]:0,go(express)),
     i:0, a[0]:1,
     while a[i]¡=k do(i:i+1, a[i]:91^i),
     r:i-1, j:0,
     while j<=r do(c[r-j]:quotient(k,a[r-j]),k:k-c[r-j]*a[r-j],j:j+1),
```

```
        express,
        return(makelist(c[r-j],j,0,r)))$

position(a):=block([i:1],
        while a#lst[i] do(i:i+1),
        return(i-1))$

exchg(b):=block([l,a,j,num],
        l:length(b),
        for j:1 thru l do(a[j]:position(b[j])),
        num:0,
        for j:1 thru l do(num:num+a[j]*91^ (l-j)),
        return(num))$

rsa(f,s):=block([str,j,k,l,b,num,mun,seq,mojilst,mojiseq],
        str:charlist(s),
        k:length(str),
        b:str,
        num:exchg(b),
        mun:power_mod(num,f,n),
        seq:adic(mun),
        l:length(seq),
        mojilst:[],
        for j:1 thru l do(mojilst:endcons(lst[seq[j]+1],mojilst)),
        mojiseq:"",
        for i:1 thru l do(mojiseq:concat(mojiseq,mojilst[i])),
        return(mojiseq))$

E(s):=block([mojiseq],
        mojiseq:rsa(e,s),
        return(mojiseq))$

D(s):=block([mojiseq],
        mojiseq:rsa(d,s),
        return(mojiseq))$
```

SecretNumbers.wxm is the following (this is only an example).

```
n:1340313730112877046697364106498934884822960177619634860931401366409365122979559520000957018272697 49$
e:3790137091$
d:42160500414096803256448652387901470020936326479111529589645154076046991787644043342350444837177323$
```

If a password is "ILike2.718" then run E("ILike2.718"). The result is

" p1IVhR]VD[WF+H2_,['A80y![¡votFvR,$!2d3BLf_m{m"

If we want to know the original password just run

D(" p1IVhR]VD[WF+H2_,['A80y![¡votFvR,$!2d3BLf_m{m").

# References

[1] S.C. Coutiho, "The Mathematics of Ciphers: Number Theory and RSA Cryptography", 1999, A K Peters, Ltd.

[2] K. Ohtake, Some remarks on primality tests, Science Report of The Faculty of Education Gunma University, Vol 63 (2015),1-8.

[3] K. Ohtake, "A first course of the theory of integers" (in Japanese), 2015(revised), unpublished textbook.

[4] M.O. Rabin, Probabilistic algorithm for testing primality, J. Number Theory 12 (1980), 128-138.

[5] R. Solovay and V. Strassen, A fast Monte-Carlo test for primality, SIAM J. Comput. 6(1977), 84-85.