アルゴリズム論特講 （塩田）

2006年6月1日の課題

課題

- 関数定義部 crypto060601.py と、RSA暗号の攻撃法（作戦A、作戦B、作戦C）のサンプルプログラム strategyA.py, strategyB.py, strategyC.py をダウンロードせよ。
- パラメータ（素数の大きさなど）を取り替えて実行してみよ。

提出期限：未定 （ 512号室ポストまで ）

サンプルプログラム（作戦A）

```
#!/bin/env python
#
# strategyA.py
# RSA 暗号の解析実験
# 作戦 A : p と q が近いときに有効な方法（Fermat 法）

from sys import *
from math import *
from random import *
from time import *
from crypto060601 import *       # crypto060601.py を引用

PrimeSize = 101                  # 素数 p, q のビット数を設定
DifferenceSize = 50              # p - q のビット数を設定
p0 = getprimebit(PrimeSize)      # 素数 p を生成
q0 = p0 + randbit(DifferenceSize)
while solovaystrassen(q0) == 0:  # p との差が小さい素数 q を生成
    q0 += 1
n = p0 * q0                      # RSA の鍵を生成
print 'n = ',n,' (',bitlen(n),'bits )'
print

print 'Fermat Method start'
t1 = time()                      # 開始時刻を計測
c = 0                            # ステップ数を計測
t = longsqrt(n) + 1              # t の初期値は sqrt(n) の切り上げ
x = t * t - n
s = longsqrt(x)
while s * s != x:                # t^2 - n が平方数になったら目的達成
    t += 1
    c += 1
    x = t * t - n
    s = longsqrt(x)

p = t - s
q = t + s
```

```
print 'p =',p
print 'q =',q
print 'check : p * q =', p * q
print 'PrimeSize =',PrimeSize,'bits'
print 'DifferenceSize =',DifferenceSize,'bits'
print 'found at the',c,'-th step'
print 'used time = ',time()-t1
```

実行例（作戦 A）

```
strategyA.dat : 作戦 A ( Fermat 法 ) による RSA 暗号解析実験

n =  428262960746243064223941559099460643122986848272191363104694 9  ( 202 bits )

Fermat Method start
p = 20694515233419768604072036784 09
q = 20694515233419778480706266960 61
check : p * q = 428262960746243064223941559099460643122986848272191363104694 9
PrimeSize = 101 bits
DifferenceSize = 50 bits
found at the 0 -th step
used time =  0.00100016593933

---------

n =  411980414820649885388942474096144440519863777542427400003610 9  ( 202 bits )

Fermat Method start
p = 20297300678184883651804572433 93
q = 20297300678185147113569511040 13
check : p * q = 411980414820649885388942474096144440519863777542427400003610 9
PrimeSize = 101 bits
DifferenceSize = 55 bits
found at the 42 -th step
used time =  0.010999917984

---------

n =  187928759457503924455223334234971000331705753561777573156748 7  ( 201 bits )

Fermat Method start
p = 13708711079364757447309681103 87
q = 13708711079365185736730351433 01
check : p * q = 187928759457503924455223334234971000331705753561777573156748 7
PrimeSize = 101 bits
DifferenceSize = 56 bits
found at the 167 -th step
used time =  0.0439999103546

---------

n =  346533878704431361754492475809098067235449106925228993877096 3  ( 202 bits )

Fermat Method start
p = 18615420454677178158684333593 03
```

```
q = 18615420454678138832429449552 21
check : p * q = 346533878704431361754492475809098067235449106925228993877096 3
PrimeSize = 101 bits
DifferenceSize = 57 bits
found at the 619 -th step
used time =  0.171000003815

---------

n =  265432067745633560251315308621224064271399891112205679470781 7  ( 201 bits )

Fermat Method start
p = 16292086046470500501857023904 89
q = 16292086046472635085059758671 53
check : p * q = 265432067745633560251315308621224064271399891112205679470781 7
PrimeSize = 101 bits
DifferenceSize = 58 bits
found at the 3495 -th step
used time =  0.986000061035

---------

n =  350279190747654876237569605616942724979939483825623736304649 1  ( 202 bits )

Fermat Method start
p = 18715747133031450226112113430 71
q = 18715747133035720832549597020 21
check : p * q = 350279190747654876237569605616942724979939483825623736304649 1
PrimeSize = 101 bits
DifferenceSize = 59 bits
found at the 12180 -th step
used time =  3.61999988556

---------

n =  235099630530736243011126102566410796516776938434712944092588 7  ( 201 bits )

Fermat Method start
p = 15332958962006632683769025500 27
q = 15332958962016854339708862731 81
check : p * q = 235099630530736243011126102566410796516776938434712944092588 7
PrimeSize = 101 bits
DifferenceSize = 60 bits
found at the 85177 -th step
used time =  24.9359998703
```

サンプルプログラム（作戦 B）

```
#!/bin/env python
#
# strategyB.py
# RSA 暗号の解析実験
# 作戦 B : p-1 と q-1 が大きな公約数を持つときに有効な方法

from sys import *
from math import *
```

```
from random import *
from time import *
from crypto060601 import *        # crypto060601.py を引用

gcdSize = 50                      # p - 1 と q - 1 の公約数のビット数を設定
g0 = 2 * randbit(gcdSize)
p = (rand(16)+1) * g0 + 1
while solovaystrassen(p) == 0:    # p - 1 が g0 を約数にもつような素数 p を生成
    p += g0
q = p + (rand(16)+1) * g0
while solovaystrassen(q) == 0:    # q - 1 が g0 を約数にもつような素数 q を生成
    q += g0

n = p * q                         # RSA の鍵を生成
m = (p-1) * (q-1)                 # 秘密鍵の m を計算
e = rand(m)                       # 公開鍵の e を生成
while gcd(e,m) != 1:
    e = rand(m)
d = modinv(e,m)                   # 秘密鍵の d を計算

plaintext = range(0,16)           # 平文は 0,1,...,15
cyphertext = []
for i in range(0,16):
    cyphertext.append(RSAencrypt(plaintext[i],e,n))      # 暗号文作成

print
print '           p = ',p
print '           q = ',q
print 'gcd(p-1,q-1) = ',gcd(p-1,q-1)
print '           n = ',n,' (',bitlen(n),'bits )'
print '           m = ',m
print '           e = ',e
print '           d = ',d
print
print 'plain  text : ',plaintext
print 'cypher text : ',cyphertext
print

print 'strategy B start'
t1 = time()                       # 開始時刻を計測
ab = 2
while 1 :
    g = longsqrt(n/ab)            # gcd(p-1,q-1) の候補は sqrt(n/小さな数)
    mm = ab * g                   # p-1 と q-1 の公倍数の候補
    # print 'ab =',ab,', g =',g,', mm =',mm
    if gcd(e,mm) == 1:
        dd = modinv(e,mm)         # mm が本当に公倍数なら dd は復号化指数
        y = []
        for i in range(0,16):     # dd を用いた復号を試みる
            y.append(RSAdecrypt(cyphertext[i],dd,n))
        if y == plaintext:
            print '   ab =',ab
            print '    g =',g
            print '   mm =',mm
            print '   dd =',dd
            print 'decrypted text : ',y
```

```
            print 'used time = ',time()-t1
            exit()
    ab += 1
```

実行例（作戦 B）

```
strategyB.dat : 作戦 B による RSA 暗号解析実験

          p =  20075661919403533
          q =  37920694736651117
gcd(p-1,q-1) =  2230629102155948
          n =  761283047281912814854878558196361  ( 110 bits )
          m =  76128304728191275685858521902141712
          e =  42410957988014178895295210226911
          d =  31189868423863312469810094176203l

plain  text :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
cypher text :  [0L, 1L, 175108370484592411964419593418319L, 492028905916209791895477 29...

strategy B start
   ab = 68
    g = 3345943653233922
   mm = 227524168419906696
   dd = 116678435335513031
decrypted text :  [0L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L]
used time =  0.763000011444

----------

          p =  312073802064111638052338943240 91
          q =  233015105541203356412413077619873
gcd(p-1,q-1) =  208049201376074425368225962 1606
          n =  727179099246135791979680171812481795045864725372895431436426044 3
               ( 213 bits )
          m =  727179099246135791979680171812455372797289963920873666739231648 0
          e =  298561597877382877927353692361205734113422127980208354882570903
          d =  399845235434927228136867317354907175522310706559788848655663760 7

plain  text :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
cypher text :  [0L, 1L, 142007888642940493953745785272540840851545543557127358419070 99...

strategy B start
   ab = 420
    g = 41609840275214885073645192432 12
   mm = 17476132915590251730930980821490 40
   dd = 12230467626356056825220733655436 87
decrypted text :  [0L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L]
used time =  15.0199999809

----------

          p =  37404632397853945723189469206796072191352225161
          q =  15377459985784399908422337340571718567555914788 1
gcd(p-1,q-1) =  41560702664282161914654965785328969101502472 40
          n =  57518823798097384046015577669557548328976261421402492258448320705029376 92
```

```
                53476139189840803384l  ( 312 bits )
          m =  57518823798097384046015577669557548328976261419490699935891341256955248499222481340314966608000
          e =  27751630507902604563997252626054833172142509842339483591322111268009804234414337309639192368870
          d =  43150486450821911286379596460570382885484054308088232751682358155198990282886785024633442388230
plain  text :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
cypher text :  [0L, 1L, 39303957662543965841927244954526418088590702594999379897055874...

strategy B start
   ab = 333
    g = 41560702664282161914654965785328969101502472400
   mm = 13839713987205959917580103606514546710800323309200
   dd = 10380369020862391225623671274844634935991218185430
decrypted text :  [0L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L]
used time =  19.1540000439
```

サンプルプログラム（作戦 C）

```
#!/bin/env python
#
# strategyC.py
# RSA 暗号の解析実験
# 作戦 C : p-1 と q-1 が小さい素因数しか持たないときに有効な方法

from sys import *
from math import *
from random import *
from time import *
from crypto060601 import *      # crypto060601.py を引用

bound = 64                      # 何番目までの素数を「小さい」と考えるか
smallprime = []
r = 1
while len(smallprime) < bound:  # bound 番目までの素数を登録
    while primetest(r) == 0:
        r += 1
    smallprime.append(r)
    r += 1

pSize = 40                      # 素数 p のビット数を設定
qSize = 60                      # 素数 q のビット数を設定

# p - 1 が小さな素因数しか持たないような素数 p を作為的に生成
p = 1
while solovaystrassen(p) == 0:
    a = 1
    while bitlen(a) < pSize:
        i = rand(bound)
        a *= smallprime[i]
    p = a + 1

# q も同様
q = 1
```

```
while solovaystrassen(q) == 0:
    a = 1
    while bitlen(a) < qSize:
        i = rand(bound)
        a *= smallprime[i]
    q = a + 1

n = p * q                          # RSA の鍵を生成
m = (p-1) * (q-1)                  # 秘密鍵の m を計算
e = rand(m)                        # 公開鍵の e を生成
while gcd(e,m) != 1:
    e = rand(m)
d = modinv(e,m)                    # 秘密鍵の d を計算

plaintext = range(0,16)            # 平文は 0,1,...,15
cyphertext = []
for i in range(0,16):
    cyphertext.append(RSAencrypt(plaintext[i],e,n))        # 暗号文作成

print
print '           p = ',p
print '           q = ',q
print '           n = ',n,' (',bitlen(n),'bits )'
print '           m = ',m
print '           e = ',e
print '           d = ',d
print
print 'plain  text : ',plaintext
print 'cypher text : ',cyphertext
print

print 'strategy C start'
t1 = time()                        # 開始時刻を計測
dbln = float(n)                    # log を取るために n を 実数型に変換
mm = 1                             # p - 1 と q - 1 の公倍数の候補
for i in range(0,bound):
    r = smallprime[i]
    if gcd(e,r) == 1:              # 暗号化指数 e と互いに素でない素数は不要
        f = int(log(dbln)/log(r))
        mm *= ( r**f )

dd = modinv(e,mm)                  # mm が本当に公倍数なら dd は復号化指数
y = []
for i in range(0,16):          # dd を用いた復号を試みる
    y.append(RSAdecrypt(cyphertext[i],dd,n))
if y == plaintext:
    print 'bound =',bound
    print '   mm =',mm
    print '   dd =',dd
    print 'decrypted text : ',y
    print 'used time = ',time()-t1
```

実行例（作戦 C）

```
strategyC.dat : 作戦 C による RSA 暗号解析実験
```

```
            p =  18358364839
            q =  103824673219247
            n =  1906051230248889062856233  ( 81 bits )
            m =  190605123014504603127214 8
            e =  1443342126349629598880113
            d =  106030321061191229483083 3

plain  text :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
cypher text :  [0L, 1L, 7253740315980742233550 55L, 1256603932456007622331681L, 1681548...

strategy C start
bound = 64
   mm = 587070046999156648419092301661278404916206553037251265988183473617772818704259327
        736551999931318046564167498430268509134494347045585281914770321768007797882204543
        692237546738337193355948414418639193266770824813975210731953070711528009682622560
        309096240218426479542004689539017308300076749935436499895537678094007645472418686
        753074456650065898114127079168475869403910946731211291824812034821146681354336387
        241472797940923402275269388923848097474493262092246891905035246618451867250279619
        610412319069643637990746067561999825462827949518665157333690649253203531591726089
        570279171376221308683571806384759773467625054077502410376260631408891722023598480
        551985307139623426382258007465045967852362609617216105715186328989229155257683937
        078641676016062166238113753980936593132286858321809938375457452499259717663304493
        005077397447439740357305207471486010937284694174229158765407633259924071710657429
        944540325124842382035470745815510981234915029361676950867080241087178951823817952
        019353705041461255754094381662746477629569885964118516827827747623190760975183122
        163339578569059301648611416958734825725624849486756259717466581769753733532710857
        446193529338840937227326543212359004285761389622726092648596105721041428721339921
        571350131880787971025205778602120302145224750390823979520196715208329242164137039
        554777694967271697061965367796288055980579906185131809430908965606070498169921707
        000270972661541023325434225113644055516331358778525971395740286577353550725120000
        000000000000000000000000000000
   dd = 461962364101066749127923573490325045694698331759177809952062490016200977015249911
        961453098808321217760975119629598855906087421075797438766984327937744440380229590
        301596325054143996497220401476058995941133661029310599574849537297761400777665918
        390264378879293072144367658962396270811708110696450809220162139517298967498351102
        697173260894193648721576753914452779594713815291797702299698623087311230703872487
        814937437853998886356624111360743909905243204880468384239273941160125776755676546
        012444469833318411029674398267343667134826020390398570202629295203115766981067277
        220718473796328957385070256461200876216418068928060676234957610327678917489769004
        527280241636380299347724872958854189148236039988326101088006527592239669050277827
        583736299872992592432297695568890266879274297227952053394038816450267441440214737
        153281621967790482888517389560097072614020909435887880513263622007737465199175723
        648215320702188676905591839853742022313219714482741096937742979543249085920786473
        976616437659772891328331697908726994982427067411746650841153253888652971952601595
        300542201400026819021332238558488919118973973017207346600566129116933365767625985
        477876827645623843271290213982156227067163074134425103494470569371459350014634669
        739942271191212267358137274823556200676950262057088917422368345704650623295610618
        461453573130435585586106122094705052895278386682316169258466485569087512515896409
        919713060120283447968558236791161419514050974976985451407700683276193679974668173
        957437934514579591103484940177
decrypted text :  [0L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L]
used time =  15.8993270397

----------
```

```
           p =  108925706789397741901 1
           q =  870341752518062010138281 99
           n =  94802590541352996326095689082939517862590491189  ( 157 bits )
           m =  94802590541352996326008653818430643767599243980
           e =  89254088317544791462343525633162135796786634 99
           d =  39357224929724256841730745120980192573116674139

plain  text :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
cypher text :  [0L, 1L, 240435001707039459252033199138218620573817999 98L, 561348108170...

strategy C start
bound = 64
   mm = 191485174729639985257510535762029959517190349726445256427764994759282320840052...
   dd = 129842375031093279584252092162638839141172204469533680946162852981244365419900...
decrypted text :  [0L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L]
used time =  52.8363931179

----------

           p =  37957525687687858914128854477072619620629 53
           q =  175297803347716882083413587103298173889900 3898013
           n =  665387087356621830114465531478996425324503592559110755663788683372186043 2
                974231273897612389  ( 302 bits )
           m =  665387087356621830114465531478996425324503417260927832690029721369457651 4
                544785112931651424
           e =  382153322804329537086449972011063550725934843365875671915991447678925277 0
                93989810659061155
           d =  415171428710533322327934855476613690996850858267115684406976862625707612 1
                691508480660664363

plain  text :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
cypher text :  [0L, 1L, 937047413215640379590080884747899102286115827608177433732773 73...

strategy C start
bound = 64
   mm = 176795183069223870859335765171621683907723840210137709489847415747297333873462...
   dd = 162612161944897135719429389345548881123959956325411345664981898600939001179563...
decrypted text :  [0L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L]
used time =  321.186975002
```

RSA 暗号の復習

鍵：

- p, q：大きな素数 (合わせて 1000 ビット程度)
- $n = pq$
- $m = (p-1)(q-1) = \varphi(n)$
- e：m と互いに素な正整数
- $d = (e \bmod m)^{-1} = \texttt{modinv}(e, m)$

このうち

- 公開鍵：e, n
- 秘密鍵：p, q, m, d

RSA 暗号システム： $P \xrightarrow{E} C \xrightarrow{D} P$

- 平文の集合 P、暗号文の集合 C はともに

$$P = C = \{\, x \,|\, 0 \leq x \leq n-1 \,\}$$

- 暗号化関数：$E(x) = (x^e,\ \bmod n) = \texttt{powermod}(x, e, n)$
- 復号化関数：$D(y) = (y^d,\ \bmod n) = \texttt{powermod}(y, d, n)$
- 送信者はメッセージ $x \in P$ を $y = E(x)$ に変換して送信する。
- 受信者は暗号文 $y \in C$ を $D(y)$ によって復号する。

- $D(E(x)) = x$ となることは前回確かめた。
- 今日は
  - RSA 暗号が安全だと信じられている理由
  - 使ってはいけない危険な鍵