RSA_Implementation

RSA

```
# Choose two primes

p = 67
q = 97
```

```
# confirm they are both primes

is_prime(p); is_prime(q)
```

```
True
True
```

```
# Compute product of primes: n
n = p* q ; n
```

```
6499
```

```
# Choose public key
e=17
```

```
#determine the private key
#It is the inverse of e
d = inverse_mod(e,(p-1)*(q-1));d
```

```
2609
```

```
# Choose plain text
m = 900; # plain text. Choose an integer for plaintext with value smaller than n
# Encryotion becomes
c = power_mod(m,e,n);c
```

```
4392
```

```
# Decryption is :
rsa_decrypt = power_mod(c,d,n); rsa_decrypt
```

```
900
```

```
# Now let us use pur typical sentence
# which is please withdraw 1000

pl = "WITHDRAW ONE THOUSAND DOLLARS" # plain text
plc = map(ord,pl) # the ord command replaces every charatcter with its ASCII index
```

```
plc[1:10]
```

```
[73, 84, 72, 68, 82, 65, 87, 32, 79]
```

```
# The list of digits plc can be understood to be the digits of a
large number of base 256.
# The next command simply computes this large number

m = ZZ(plc,256)
```

```
m # view m
```

```
224633804334567595697957867396832825244059349753398487068937408140
23
```

```
ml = m.digits(256) # To get back the other
                   # way around use this command
```

```
ml[1:10]
```

```
[73, 84, 72, 68, 82, 65, 87, 32, 79]
```

```
# Convert to charater using the command below
ms = map(chr,ml);ms
```

```
['W',
 'I',
 'T',
 'H',
 'D',
 'R',
 'A',
 'W',
 ' ',
 'O',
 'N',
 'E',
 ' ',
 'T',
 'H',
 'O',
 'U',
 'S',
 'A',
 'N',
 'D',
 ' ',
 'D',
 'O',
 'L',
 'L',
 'A',
 'R',
```

```
 'S']
```

```
''.join(ms) # use join to bring them all together
```

```
'WITHDRAW ONE THOUSAND DOLLARS'
```

```
def str2num(s):
    '''
    Define a new function that does the mapping
    referenced abive
    input : charater of message
    output: replace each character with the ascii equivalent

    '''
    return ZZ(map(ord,s),256)


def num2str(n):
    '''
    input :integer representing message characters
    output : plain text. Convert integer to its corresponging ASCII
index then to its charater equivalent
    '''

    nl=n.digits(256)

    return ''.join(map(chr,nl))
```

```
# Trying more complex RSA
p = next_prime(2^330) # first prime
```

```
q = next_prime(3^210) # 2nd prime
```

```
n = p*q # product of primes
```

```
e = 41  # public key
d = inverse_mod(e,(p-1)*(q-1)) #private key
```

```
c = power_mod(m,e,n) # cipher text
```

```
c
```

```
337205604304101432492663943411406615942668463539678835794722239604
583385958698747675514449014392715228768869190275166029192230289778
2345318491067823963242997112355077151536569929584936020303604744
```

```
# To recover plain text
```

```
pl = power_mod(c,d,n)
```

```
pl # ascii equivalent
```

```
22463380433456759569795786739683282524405934975339848706893740814O
23
```

```
num2str(pl) # see the message
```

```
'WITHDRAW ONE THOUSAND DOLLARS'
```

```
# Using chinese remainder theorem we can make computation very fast

g,s,t = xgcd(p,q)
s*p+t*q
mp = power_mod(c,d%(p-1),p)
mq = power_mod(c,d%(q-1),q)
m = (s*p*mq+t*q*mp)%n
num2str(m)
```

```
'WITHDRAW ONE THOUSAND DOLLARS'
```

RABIN CRYPTOSYSTEM

```
p = next_prime(2^100); p; f = mod(p,4); f
```

```
1267650600228229401496703205653
1
```

```
while mod(p,4)==1:
    p = next_prime(p+1)
p
```

```
1267650600228229401496703205707
```

```
mod(p,4) # test the prime is 3 mod 4
```

```
3
```

```
q = next_prime(p+1)
while mod(q,4)==1:
    q = next_prime(q+1)
```

```
mod(q,4) # test the prime is 3 mod 4
```

```
3
```

```
pl = "GET DETERMINED, GO AHEAD" # plain text
```

```
pn = str2num(pl); pn
```

```
167360683316169938229842396197504461269767718348056037101 5
```

```
N = p*q # multiply the primes
ct = power_mod(pn,2,N)
```

```
ct  # cipher text
```

```
743819548575580576227844293644122447042984361042691751099053
```

```
x,s,t = xgcd(p,q) # extended euclidean
```

```
s;t
```

```
120208246573366581176411510897
-120208246573366581176411510886
```

```
cp = power_mod(ct,(p+1)//4,p) # compute cp
```

```
cq = power_mod(ct,(q+1)//4,q) # compute cq
```

```
(s*p*cq+t*q*cp)%N # compute the solution
```

```
1673606833161699382298423961975044612697677183480560371015
```

```
(s*p*cq-t*q*cp)%N # compute quadratic residue
```

```
1304211030921332222873575688284973317349861418547942828738702
```

```
(-s*p*cq+t*q*cp)%N # quadratic residue
```

```
302727013337658052668386405042421452149904049599285100493159
```

```
(-s*p*cq-t*q*cp)%N # residue
```

```
1605264437425828576159663669365419724887067790963747368860846
```

```
num2str((+s*p*cq+t*q*cp)%N)
```

```
'GET DETERMINED, GO AHEAD'
```

POLLARD RHO

```
def pollard_rho(n):

    x = Mod(6,n)
    y=x
    while true:
        x = x^2+1
        #print(x)
        y = (y^2+1)^2 + 1
        #print(y)
        g = gcd(x-y,n)
        #print(g)
        #print(parent(g))
        #print(ZZ(g)<n);
        if (g>1) and (ZZ(g)<n):
            print (g) ; break;
        if x==y:
            #print(x)
            #print(y)
```

```
            print "no factor";break
```

```
pollard_rho(2^67-1) # test this
```

    193707721

    2