Cryptography hw10

YanLiang 112889478 2018/4/10

1.

(a) I can't find the plaintext from the cyphertext, since only know n and e, we can not know p and q , so we can not know the d, and we also don't know how many digits the plain text is , so we can not do exhaustive search.

(b) know the decryption key, we can solve the solution, p=c^d mod (n)

So using sage:

sage: n=**1259531756783983515701499777642110356794201569384295868500005799617750 54**

....: **8880147110509521944049285041602433244172023804646590835427723055191592 1446**

....: **3831847643286738542961736012 1**

sage: d=**8798291625428500747488389737164626414702923210768430788704131331385418 94**

....: **3151675346554285160058983961221033242939250579818020233301861067940906 4495**

....: **2807381680714475934931163153**

sage: c=**112889478**

sage: R=Integers(n)

sage: R(c)^d

73285852951768619689849048209951829860939990805474855984914576794390613865372733937
35476147768257269757981084998761901646194810287447667447727630923699607231580361214
48893940

The above is the plaintext for the cyphertext.

(c) yeah we can factor n by knowing e, d using the following algorithms

sage: p

13958346820346854795879058730587957395875986247058246704760586029856023786027620782
3

sage: q

90235023745647628560576284052173474203670237452758427654762875017624769204804850672
09574327

sage: p.is_prime()

True

sage: q.is_prime()

```
True

sage: p*q

1259531756783983515701499777642110356794201569384295868500005799617750548880147110509521944049285041602433244172023804646590835427723055191592144638318476432867385429617360121

from sage.all import *

import random

e=65537

d =
879829162542850074748838973716462641470292321076843078870413133138541894315167534655428516005898396122103324293925057981802023330186106794090644952807381680714475934931163153

def findK():
    data=e*d-1
    count=0
    while(data%2==0):
        data=(data//2);
        count=count+1
    return [data,count];

k=(findK()[0])

s=findK()[1]

print(s)


import random

s=4

k=3603835239098172834300916238778488258377409240400816553745641594156288757983320919919551165842410199165892847765685407809584950186900430060269912392008585825561538084223977472385

#n=1259531756783983515701499777642110356794201569384295868500005799617750548880147110509521944049285041602433244172023804646590835427723055191592144638318476432867385429617360121
```

```
n=12595317567839835157014997776421103567942015693842958685000057996177505488801471105095219440492850416024332441720238046465908354277230551915921446383184764328673854
29617360121

d=87982916254285007474883897371646264147029232107684307887041313313854189431516753465542851600589839612210332429392505798180202333018610679409064495280738168071447593
4931163153

t=0

count=0

R=Integers(n);

result=0;

for i in range(0,1):

   a=random.randint(1,n);

   while(gcd(a,n)!=1):

      a=random.randint(1,n)

   #print(a)

   while(t<=s-1):

      gcddata=gcd(R(a)**((2**t)*k)-1,n)

      if(gcddata==1):

         #print("I am always 1")

         count=count+1

      else:

         print(gcddata)

         #result=gcddata

         break;

      t=t+1

   t=0

print(count)
```

(d) let make the loop run 100 times by changing for I in range(0,100), then we count how many a we get that works out,

```
sage:
sage:
sage: t=0
....: count=0
....: R=Integers(n);
....: gcddata=1
....: result=0;
....: for i in range(0,100):
....:     a=random.randint(1,n);
....:     if(gcd(a,n)!=1):
....:         nocount=nocount+1
....:         continue;
....:         #a=random.randint(1,n)
....:     #print(a)
....:     while(t<=s-1):
....:         gcddata=gcd((R(a)**((2**t)*k)-1),n)
....:         if(gcddata>1):
....:             count=count+1
....:             break;
....:         else:
....:             t=t+1
....:     t=0
....: print(count)
....:


53
sage:
```

So I get the result is around 0.5.

2.

we can not use p==1 mod (e) since in order for the RSA to work, we need to make sure that e.d=1 mod (phi(n)), phi(n)=(p-1)*(q-1) if p==1 mod e ,then e|p-1 which means e is not relative prime to phi(n), so we can not find d, so we can not decrypt this algorithm.

```
sage: datastart=11288947800000000000000000000000000000000000000000000000000000000000000000000000000
....: 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
....: 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
....: 000000000000000000000000000031683
sage: e=65537
sage: for i in range(0,1000):
....:     if((datastart+e*i).is_prime()):
....:         print(i)
....:

202
488
sage:
sage: finaldata=11288947800000000000000000000000000000000000000000000000000000000000000000000000000
....: 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
....: 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
....: 000000000000000000000000013270157
sage: finaldata.is_prime()
True
sage: finaldata>2**1000
True
sage: finaldata<2**1004
True
sage: finaldata%e
1
sage:
```

Finaldata is the data I find and the above sage code is to verify that it satify all the conditions we required.

p=11288947800013270157

The idea how I get this:

So my id is 112889478 which is 27 bits, if we require 2^1000 at lease so I need to multiply data=id*10^(log(2^973,10)), doing in this way will keep the leading digit become with my id,

Then I test this number is within 2^1000 and 2^1004

Then I did

Datastart=data+(e-data%e)+1, this will make sure that it mod e will get 1,

Then I only explore 1000 times to see if I can find a prime like this:

```
for i in range(0,1000):
   if((datastart+e*i).is_prime()):
      print(i)
```

I get two solution I=202 and I=488, the p is based on 202.