Easy RSA

Problem: We encrypted the flag with RSA, can you crack it?
Hint: Short and weak public key
Given: easy-rsa.tar.gz (custom1.enc, custom2.enc, custom3.enc, and public_key.pem)

Steps:
(1) Understanding the public key that is in the given folder. The public key file is the public_key.pem file.

If you try to open this file regularly it will not open correctly. Lucky there is a tool called openssl that can make this file very readable. The following command will give you your public_key file in a readable text

$openssl rsa –pubin –in public_key.pem –text –noout

The following will be given:

```
[qijun@glap crypto]$ openssl rsa -pubin -in easyrsa.given/public_key.pem -text -noout
Public-Key: (256 bit)
Modulus:
    00:b7:bc:eb:2e:74:9f:96:0b:49:0d:89:90:e1:f9:
    34:0d:df:50:bb:d1:99:c8:c8:f3:88:53:7f:d3:2a:
    a7:c8:01
Exponent: 65537 (0x10001)
[qijun@glap crypto]$
```

We now have our public key: modulus and exponent. They were used for encryption. We will need them for cracking and decryption later.

The current modulus is not usable as is. It needs to be converted to a number. This can be done easily in python. We concatenate the three lines of the modulus to one line.
modulus='00:b7:bc:eb:2e:74:9f:96:0b:49:0d:89:90:e1:f9:34:0d:df:50:bb:d1:99:c8:c8:f3:88:53:7f:d3:2a:a7:c8:01'

Then, then we do some string manipulation and number conversion. We got the modulus in decimal:
n=831070417017470035489516199160732676570521364540798304365782671279776999526 41

```
[qijun@glap crypto]$ python
Python 2.7.11 (default, Sep 29 2016, 13:33:00)
[GCC 5.3.1 20160406 (Red Hat 5.3.1-6)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> modulus='00:b7:bc:eb:2e:74:9f:96:0b:49:0d:89:90:e1:f9:34:0d:df:50:bb:d1:99:
c8:c8:f3:88:53:7f:d3:2a:a7:c8:01'
>>> modulus=''.join(modulus.split(':'))
>>> modulus
'00b7bceb2e749f960b490d8990e1f9340ddf50bbd199c8c8f388537fd32aa7c801'
>>> n=int(modulus,16)
>>> n
831070417017470035489516199160732676570521364540798304365782671279776999526 41L
>>>
```

(2) Now that we have the decimal version the modulus we need to get the two largest prime factors of this modulus. There is a great tool called ‘yafu’ that can be used to do this for us.
Yafu can be read about at this link. Yafu can be installed here. After installation, make sure that you have the executable permissions to run yafu. If yafu is not executable (permissions don’t contain the character ‘x’), try running ‘chmod 755 yafu’. Run yafu by entering ‘./yafu’.

Now we will factorize this integer to get our P1 and P2 which are needed for the decryption of rsa. When running this tool it looks like the following and will take a little time.

```
>> factor(831070417017470035489516199160732676570521364540798304365782671279776999526 41)

fac: factoring 831070417017470035489516199160732676570521364540798304365782671279776999526 41
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
rho: x^2 + 3, starting 1000 iterations on C77
rho: x^2 + 2, starting 1000 iterations on C77
rho: x^2 + 1, starting 1000 iterations on C77
pm1: starting B1 = 150K, B2 = gmp-ecm default on C77
ecm: 30/30 curves on C77, B1=2K, B2=gmp-ecm default
ecm: 74/74 curves on C77, B1=11K, B2=gmp-ecm default
ecm: 149/149 curves on C77, B1=50K, B2=gmp-ecm default, ETA: 0 sec

starting SIQS on c77: 831070417017470035489516199160732676570521364540798304365782671279776999526 41

==== sieving in progress (1 thread):   36224 relations needed ====
====           Press ctrl-c to abort and save state           ====
36320 rels found: 19113 full + 17207 from 184309 partial, (2363.50 rels/sec)

SIQS elapsed time = 88.1404 seconds.
Total factoring time = 109.7184 seconds


***factors found***

P39 = 294456935544012154828625255162223768143
P39 = 282238357022244718977290902746061069487

ans = 1
```

We now have P1 and P2. We can perform the decryption.
P1 = 294456935544012154828625255162223768143
P2 = 282238357022244718977290902746061069487

(3) Now we need to understanding our script to decrypt the rsa files we were given.
I will go over some variables that you will need to know to go over decryption.

z = raw data in our file
c = cipher text as number
n = our long decimal value of the modulus from the public key
e = the exponent given in the public key
P1=prime factor 1 of the modulus key
P2=prime factor 2 of the modulus key

r= totient number, i.e. (P1-1)x(P2-1)
d=the decryption key
p=the plaintext as number
pt=plaintext in a string representation

There are also a couple libraries in python that we will need to have for our decryption script.
-gmpy
-Crypto.Util.number
-base64

(4) Now we can begin to build our script that will decrypt the given rsa files.

First lets open the first file custom1.enc and read it as encoded hex values

```
z = open('custom1.enc','r').read().encode('hex')
```

Now we turn the cipher text to a number

```
c = int(z,16)
```

Set e to the exponent we were given in our public key

```
e = 65537
```

Set r to the equation to generate totient number

```
r = (P1-1)*(P2-1)
```

For the d variable we user gmpy to get our invert of the e over r. We need to use the gmpy library that we imported earlier

```
d = int(gmpy.invert(e, r).digits())
```

Now set p to generate our plaintext using the rsa equation c^d%n

```
p = pow(c,d,n)
```

Let pt be your new plaintext that is decoded from hex into a readable string then print it to receive the plaintext. To use the decode, pt must be an even number.

```
pt = hex(p).strip('0x').strip('L').decode('hex')
print pt
```

We get the plaintext “FLAG{R5” from the first cipher text file.

We have two more cipher text files to decrypt ...

Sample Script:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import gmpy
from Crypto.Util.number import *
import base64
```

```
z = open('custom1.enc','r').read().encode('hex')
c = int(z,16)
n =
831070417017470035489516199160732676570521364540798304365782671279776999526
41
e = 65537
P1 = 282238357022244718977290902746061069487
P2 = 294456935544012154828625255162223768143
r = (P1-1)*(P2-1)
d = int(gmpy.invert(e, r).digits())
p = pow(c,d,n)
pt = hex(p).strip('0x').strip('L').decode('hex')
print pt
```