ECE 4802, Project 6

Calvin Figuereo-Supraner

December 9 2016

Versioning

```
$ python3 --version
Python 3.5.2
```

Usage

```
$ ./q2.py
$ ./q3.py
```

Packages

The program `q3.py` requires the package below.

```
$ pip3 install pycrypto
```

Problem 1

1a

Let, in mod n,

$$\begin{aligned} c_1 &= m_1^e \\ c_2 &= m_2^e \end{aligned}$$

Encrypt the product of the plaintexts:

$$c_3 = (m_1 m_2)^e$$

The product of the ciphertexts is:

$$c_1 c_2 = (m_1^e)(m_2^e)$$

Show that the two equations above are equal:

$$\begin{aligned} c_3 &= (m_1 m_2)^e \\ &= (m_1^e)(m_2^e) \\ &= c_1 c_2 \end{aligned}$$

Therefore the property holds.

1b

Let

$$y_3 = \mathsf{Enc}(m)$$

If Oscar sends Bob the ciphertext:

$$y_2 = y_1 y_3 \mod n$$

Then Oscar can capture $\mathsf{Dec}(y_2)$ from Bob's machine, and determine y_1 by:

$$y_1 = \boxed{\mathsf{Dec}(y_2) m^{-1} \mod n}$$

Problem 2

Script output:

```
$ ./q2.py
p
12364962413333143004054211290476631725616049550716574932219803710807452780088434
01830212236838298820346482667081675466967754865176447993798670995852709540114327
68851031546765883114628629742983764951656761606600919103832684391588782989363935
449585879242176232109445135220311653193946613126726811471408064051651
q
16758898823704979005377029236046572735977621403099759454821696930773741716556983
82804788265600602455738260653356631984469710422323365952893817789028638956676294
96173994586444044095092155815236152957457867210105285100175327615045036942600249
372983139494678451379212557048684935316515935189352502701156597042417
```

2a

$$
\begin{aligned}
\phi(N) &= (p-1)(q-1) \\
&= pq - p - q + 1 \\
&= N - p - q + 1 \\
q &= N - p - \phi(N) + 1 \\
N/p &= N - p - \phi(N) + 1 \\
N &= -p^2 + (N - \phi(N) + 1)p \\
0 &= \boxed{-p^2 + (N - \phi(N) + 1)p - N}
\end{aligned}
$$

The script `q2.py` finds $p, q \in \mathbb{Z}$ using the quadratic equation for the formula above.

Problem 3

Script output:

```
$ ./q3.py
.
----------------------------------------------------------------------
Ran 1 test in 0.321s

OK
```

3a

$$|r| = |N| - |m| - 1 = 1024 - 256 - 1 = \boxed{767 \text{ bits}}$$

3b

`PaddedRSA.gen()` returns N, e, and d, necessary values for a public and a private RSA key.

3c

`PaddedRSA.enc()` encrypts 256-bit messages by adding padding.

3d

`PaddedRSA.dec()` decrypts 1024-bit messages by removing padding. Both the `.dec()` and `.enc()` methods are tested using the `unittest` module.

Problem 4

The encrypted e-mail submitted was:

```
Hi,

One application of e-mail encryption is protecting
the confidentiality and privacy of a journalist's
news sources.

-- Calvin Figuereo-Supraner
```

Source Code

q2.py

```
#!/usr/bin/env python3

import sympy

# N = p*q
N = int(
    '207223154043965088701210756045126564627197934600164356385160399263771929'
    '991483408993337800744326333103137124134534068872908011827512897157390544'
    '596397117851242454073619092829540312195768292334791998692595110781482773'
    '595602219169897575776397522579344394080292332296096534859053608770823602'
    '964966611853830620470922076915989174277656925726593353119528887412084256'
    '743778409391376962049150174045041670223051272854509883078794488172348520'
    '369982870504279948335463394069143911301107892455488608193251819241526996'
    '491211158743786862171618065746669565843195845506062710797638743027444024'
    '27213265557318790786231798363244525880467')

# t = totient(N)
t = int(
    '207223154043965088701210756045126564627197934600164356385160399263771929'
    '991483408993337800744326333103137124134534068872908011827512897157390544'
    '596397117851242454073619092829540312195768292334791998692595110781482773'
    '595602219169897575776397522579344394080292332296096534859053608770823602'
    '964966611853830620468009690792285362076713801673941032673369520316702623'
    '305074259327218842599485632260406669720612371578425139758356180720911055'
    '082483056557587459550582045572353288650857631123389336096043963659327817'
    '400064870576724820131537945680331366523553997280372523429091908140867101'
    '58216677046856242470152484190679864786400')

a, b, c, x = -1, N-t+1, -N, sympy.Symbol('x')   # params
p, q = sympy.solve(a*x**2 + b*x + c, x)         # soln

assert N == p*q; print("p\n{}\nq\n{}".format(p, q))
```

q3.py

```
#!/usr/bin/env python3

import unittest
from Crypto.Random import random
from Crypto.Random.random import randint
from Crypto.PublicKey import RSA

MLEN = 256
NLEN = 1024
RLEN = NLEN-MLEN-1

def _mod_mult_inv(x, modulo):
    t1, t2, r1, r2 = 0, 1, modulo, x
    while r2 != 0:
        q = (-1 if (r1<0)!=(r2<0) else 1) * (r1 // abs(r2))
        t1, t2, r1, r2 = t2, t1-q*t2, r2, r1-q*r2
    return (t1 if t1>0 else t1+modulo)

class PaddedRSA():
    def __init__(self):
        self.N, self.e, self.d = self.gen()

    def gen(self):
        rsa = RSA.generate(1024)
        p = getattr(rsa.key, 'p')
        q = getattr(rsa.key, 'q')
        N = p*q
        e = 2**16 + 1 # assume phi > e, usually
        d = _mod_mult_inv(e, (p-1)*(q-1))
        return N, e, d

    def enc(self, m):
        r = randint(0, 2**RLEN)
        m2 = (r << MLEN) | m
        return pow(m2, self.e, self.N)

    def dec(self, c):
        m2 = pow(c, self.d, self.N)
        return m2 & (2**MLEN - 1)

class Tests(unittest.TestCase):
    def _rand_msg(self):
        return randint(0, 2**(MLEN//8))
```

```
    def test_enc_and_dec(self):
        """Encrypt and decrypt using own RSA"""
        m = self._rand_msg()    # generate random message
        rsa = PaddedRSA()       # initialize RSA class
        c = rsa.enc(m)          # encrypt the message
        m2 = rsa.dec(c)         # decrypt the message
        self.assertEqual(m, m2) # message should be unchanged

if __name__ == '__main__':
    unittest.main()
```