Project 6

Zac Foteff

March 2021

Problem 1:

$$\begin{aligned} d &= e^{-1} mod (p-1)(q-1) \\ d &= 7^{-1} mod (11-1)(13-1) \\ d &= 7^{-1} mod (10)(12) \\ d &= 7^{-1} mod 120 \\ 7d &= 1 mod 120 \\ 7(103) mod 120 &= 1 \\ d &= 103 \end{aligned} \tag{1}$$

Problem 2:

p = 2, q = 13, d = 103, e = , m = 99, n = p x q = 143

2A)

$$\begin{aligned} Enc(m) = c &= m^e mod n \\ c &= 99^7 mod 143 \\ c &= 44 \end{aligned} \tag{2}$$

2B)

$$\begin{aligned} Dec(c) = m &= c^d mod n \\ m &= 44^{103} mod 143 \\ m &= 99 \end{aligned} \tag{3}$$

Problem 3, 4, 5:

RSA is a public key encryption system that secures communication between two people Bob and Alice. Since we can assume that the normal encryption method of RSA is correct and secure, it is possible that Bob and Alice can share encrypted messages without also having to share a key. Assuming no one else is aware of the system, Alice can use a digital signature when encrypting her messages by encrypting her message with her own private key. Normally a person's private key is kept secret, since it is used for decryption. However, if Alice signs her message with their public key and their private key, then anyone

with the message could decrypt the message as long as they were aware of what they were looking for. This digital signature makes the message completely vulnerable to man in the middle attacks, so Bob would have to know how Alice conceals the signature in the message.

Problem 6, 7, 8:

6

```
"""
Name: Zac Foteff
Class: CPSC 353
Date Submitted: [3/21/2021]
Assignment: Project 6
Description: Program performs necessary calculations in order to
generate a public and private key for a RSA encryption system
"""

from sage.arith.misc import GCD
import random

p1 = pow(2, 100)
p2 = pow(2, 150)
p = next_prime(random.randint(p1, p2))
q = next_prime(random.randint(p1, p2))
n = p * q
phi = (p-1)*(q-1)
e = 2
d = 1

while e < n:
    if ( GCD(e, phi) == 1 ):
        break

    e = e + 1

d = inverse_mod(e, phi)

print ("P: "+str(p))
print ("Q: "+str(q))
print ("N: "+str(n))
print ("E: "+str(e))
print ("D: "+str(d)+"\n")
print ("Public Key (n, e): (%i, %i)" %(n, e))
print ("Private Key (d): (%i)" %(d))
```

7

```
========== Output ==========
P:  13843649185962876317193314857754696417205471 73
Q:  84195203108145457477271937388138538097414619 1
N:  11655688549700568963608020465551497286581335569993591784704296075183552856007364550137680 43
E:  11
D:  84768643997822319735331057931283616629682440347130343912195447347367594614100467805023613 1
    43912195447347367594614100467805023613 1

Public Key (n, e):
(11655688549700568963608020465551497286581335569993591784704296075183552856007364550137680 43, 11)
Private Key (d):
(84768643997822319735331057931283616629682440347130343912195447347367594614100467805023613 1)
```

8

```
"""
    Name: Zac Foteff
    Class: CPSC 353
    Date Submitted: [3/21/2021]
    Author/Source: Paul DePalma
    Assignment: Project 6
    Description: Converts a string to a decimal digit sequence
    and vice versa key for an RSA encryption system
"""

#Converts a string to a decimal digit sequence
#msg_in is a string

c = 365671434072182729571919499070229583742168236398802310266 75976705729832866517236887917755829297428539263691181 3
d = 84768643997822319735331057931283616629682440347130343912195447347367594614100467805023613 1
n = 11655688549700568963608020465551497286581335569993591784704296075183552856007364550137680 43

c = Integer(c)
d = Integer(d)
n = Integer(n)
m = pow(c,d,n)
m = Integer(m)

def txt_to_num(msg_in):
  #transforms string to the indices of each letter in the 8-bit ASCII table
  #ex: "AB" becomes [65,66]
   msg_idx = list(map(ord,msg_in))
```

```
  #The integers in the list, since they are ASCII,
  #are in the range 0..255
  #These are treated, collectively, as a base 256 integer,
  #but from left to right rather than right to left
  #This sequence of integers is converted to base 10
  #ex: [65,66] = 65*256^0 + 66*256^1 = 16961
  num = ZZ(msg_idx,256)
  return num

#Converts a digit sequence to a string
#num_in is a decimal integer, constructed from a string using txt_to_num
def num_to_txt(num_in):
  #transforms the base 10 num_in to a list of base 256
  #digits. 256^0 is on the left

  msg_idx = num_in.digits(256)

  #maps each index to its associated character in the ascii table
  m = map(chr,msg_idx)

  #transforms the list to a string
  m = ''.join(m)
  return m

print("Message: "+str(num_to_txt(m)))
```

Output

```
Message: ¨l=P]i$Vñj3×¯¼ødgºÕåp:
                              Ó²´
```

Problem 9:

Use Fermat's algorithm to factor n = 119143

$$
\begin{aligned}
ceil(\sqrt{119143}) &= 345.17097 = 345 \\
345^2 - 119143 &= 118 \\
346^2 - 119143 &= 573 \\
347^2 - 119143 &= 1266 \\
348^2 - 119143 &= 1961 \\
349^2 - 119143 &= 2658 \\
350^2 - 119143 &= 3357 \\
351^2 - 119143 &= 4058 \\
352^2 - 119143 &= 4761 \\
\sqrt{4761} &= 69 \\
119143 &= (352 - 69)(352 + 69)
\end{aligned} \tag{4}
$$

Problem 10:

The formula for a common modulus attack revolves around vulnerabilities created in RSA encryption that come from choosing a small value for e. The value of e is coprime to phi, or (p-1)(q-1)

Let p, q be two coprime numbers. Then we know that e has a unique solution mod pq

$$e = a \ (mod \ p) \ e = b \ (mod \ q)$$

Assuming the programmer follows good practice, those numbers a, b will be large prime numbers that are combined with a modulus n to form a public key. An attacker can try different keys and modulus' using the Chinese Remainder Theorem.

Suppose $p' = p^{-1} \ (mod \ q)$ and $q' = q^{-1} \ (mod \ p)$. Using the Chinese Remainder Theorem:

$$e = aqq' + bpp' \ (mod \ pq)$$

Since we know that $qq' = 1 \ (mod \ p)$, if we mod e by p that will yield $e = a \ (mod \ p)$. Similarly, $pp' = 1 \ (mod \ q)$, meaning that if we mod e by q that yields $e = b \ (mod \ q)$.

So as long as the attackers has two different numbers and modulus, they will be able to find the e that is shared in common by the encryption clients.