ECE 4802 - Assignment 6

Thomas Mackintosh

December 9, 2016

1 7.12

1.1 Show that the multiplicative property holds for RSA, i.e., show that the product of two ciphertexts is equal to the encryption of the product of the two respective plaintexts.

Prove: $c_1 \times c_2 = Enc(m_1 \times m_2)$

$$\begin{gathered}
c_1 = Enc(m_1) = m_1^e \bmod N \\
c_2 = Enc(m_2) = m_2^e \bmod N \\
m_1 = Dec(c_1) = c_1^d \bmod N \\
m_2 = Dec(c_2) = c_2^d \bmod N \\
m_1 \times m_2 = (c_1^d \bmod N) \times (c_2^d \bmod N) \\
m_1 \times m_2 = c_1^d \times c_2^d \bmod N = (c_1 \times c_2)^d \bmod N = Dec(c_1 \times c_2) \\
Enc(m_1 \times m_2) = c_1 \times c_2
\end{gathered}$$

1.2 This property can under certain circumstances lead to an attack. Assume that Bob first receives an encrypted message y_1 from Alice which Oscar obtains by eavesdropping. At a later point in time, we assume that Oscar can send an innocent looking ciphertext y_2 to Bob, and that Oscar can obtain the decryption of y_2. In practice this could, for instance, happen if Oscar manages to hack into Bob's system such that he can get access to decrypted plaintext for a limited period of time.

If Oscar can get Bob to decrypt $c_1 \times c_3 \bmod N$, where c_3 is the encryption of some message m chosen by Oscar, then Oscar can multiply the decrypted ciphertext by the multiplicative inverse of m to recover c_1.

2 Let N = pq be the product of two distinct primes. Show that if Φ(N) and N are known, then it is possible to compute p and q efficiently.)

Below is the Python3 code I used to compute p and q, computepq.py:

```
#   Show that if Phi(N) and N are known, then it is possible
#   to compute p and q efficiently.

#   PhiN) = (p-1)(q-1), N = pq
#   Phi(N) = pq - p - q + 1
#   Phi(N) = N - p - q + 1
#   q = N - p - Phi(N) + 1
```

```python
#   N/p = N - p - Phi(N) + 1
#   N = -p^2 + (N - Phi(N) + 1)p
#   0 = -p^2 + (N - Phi(N) + 1)p - N
import math

def isqrt(n): # Returns square root of input parameter
    x = n
    y = (x + 1) // 2
    while y < x:
        x = y
        y = (x + n // x) // 2
    return x

def solveQuad(a,b,c): # Solves quadratic with coefficient inputs
    #   calculate the discriminant
    d = (b**2) - (4*a*c)
    #   find two solutions
    sol1 = (-b-isqrt(d))//(2*a)
    sol2 = (-b+isqrt(d))//(2*a)

    # Not likely for our application but we'll include anyway:
    if sol2 > 0 & sol1 > 0: print( "Found more than one p."); return 1
    # Take whichever solution is not negative
    if sol1 > 0: print( "p = ", sol1 ); return sol1
    if sol2 > 0: print( "p = ", sol2 ); return sol2

if __name__ == "__main__":
    N=int("""
    207223154043965088701210756045126564627197934600164356385160399263771929
    991483408993337800744326333103137124134534068872908011827512897157390544
    596397117851242454073619092829540312195768292334791998692595110781482773
    595602219169897575776397522579344394080292332296096534859053608770823602
    964966611853830620470922076915989174277656925726593353119528887412084256
    743778409391376962049150174045041670223051272854509883078794488172348520
    369982870504279948335463394069143911301107892455488608193251819241526996
    491211158743786862171618065746669565843195845506062710797638743027444024
    27213265557318790786231798363244525880467""".replace("\n", ""))

    phiN=int("""
    207223154043965088701210756045126564627197934600164356385160399263771929
    991483408993337800744326333103137124134534068872908011827512897157390544
    596397117851242454073619092829540312195768292334791998692595110781482773
    595602219169897575776397522579344394080292332296096534859053608770823602
    964966611853830620468009690792285362076713801673941032673369520316702623
    305074259327218842599485632260406669720612371578425139758356180720911055
    082483056557587459550582045572353288650857631123389336096043963659327817
    400064870576724820131537945680331366523553997280372523429091908140867101
    58216677046856242470152484190679864786400""".replace("\n", ""))

    #   Solve quadratic 0 = -p^2 + (N - phi(N) + 1)p - N
    p = solveQuad(-1,(N - phiN + 1),-N) #  Solves for p
    #   Solve for q from N = pq
    q = N//p

    print( "q = ", q )

    if(p*q == N):
        print( "Found p and q!")
    else: print("Not quite...")
```

3 Implement padded RSA, as introduced in class. Assume that the message m is always a 256 bit key, i.e. $|m| = 256$ and that $|N| = 1024$ bit.

Below is the code I used to implement a padded RSA scheme, pad_rsa.py:

```
import random
import math
from Crypto.Util import number

def pad(M,N):
    """
    The function that return padded message with size |N| -1
    first generate random r and return r||M
    The size of r is determined by size of M and N
    """
    magN = 1024
    l = 256
    r = random.getrandbits(magN - l - 1)
    r << (magN + 1)
    res = r | M
    return res

def paddedRSAEnc(m,e,n):
    """
    The function that return m^e (mod n)
    We know that m = r||M
    """
    enc = pow(m,e,n)
    return enc

def paddedRSADec(c,d,n,l):
    """
    The function that return last l bit of c^d (mod n)
    """
    res = bin(pow(c,d,n))[:l+2]
    return res


def RSAGen():
    """
    The function to create prime numbers p and q
    calculate N = p*q and phi = (p-1)*(q-1) and
    also public key <N,e> and private key <p,q,d>
    """

    q = number.getPrime(512)
    p = number.getPrime(512)

    n = p*q
    phi = (p-1)*(q-1)
    print(int(math.log(phi, 2)) + 1)
    print(int(math.log(n, 2)) + 1)
    e = 2**16+1
    if number.GCD(e,phi) != 1: print("Error: GCD(e,phi) != 1.")
    d = number.inverse(e,phi)

    return e,d,n

def RSATest():
    """
    create random message and encrypt the message
    to get ciphertext and decrypt the ciphertext to
    getback to plaintext.
    """
    M = random.randint(0,2**256-1)
    (e,d,N) = RSAGen()
    padded_message = pad(M,N)

    c = paddedRSAEnc(M,e,N)
    l = 256
    M1 = paddedRSADec(c,d,N,l)
    print("M1 = ", M1)
    print("M =  ", bin(M))
    if M == int(M1,2):
        print("Correct")
    else:
        print("Wrong ...")


RSATest()
```

The length of pad $r = |N| - l - 1 = 1024 - 256 - 1 = 767$ bit.

4 In this question you will become familiar with real world usage of public key encryption. The goal is to send a correctly encrypted email to ece4802@WPI.EDU, containing your name and explain why you might need to use this method to send an email to someone. Your email should be encrypted using the public key available in mywpi.

Using mailvelope I successfully encrypted an email that could be decrypted using the public key provided. Sending an email this way is useful in hiding the contents of emails between users. This could be useful in a company or a government corporation where secrecy could be very important, if not critical to a successful operation.