Takehome 1

October 18, 2020

[1]:
```
########## Preamble
########## Loading in fastpowering and euclidean algorithm and find inverse


def fastPowerSmall(g,A,N):
    a = g
    b = 1
    while A>0:
        if A % 2 == 1:
            b = b * a % N
        A = A//2
        a = a*a % N
    return b

def extendedEuclideanAlgorithm(a,b):
    u = 1
    g = a
    x = 0
    y = b
    while true:
        if y == 0:
            v = (g-a*u)/b
            return [g,u,v]
        t = g%y
        q = (g-t)/y
        s = u-q*x
        u = x
        g = y
        x = s
        y = t

def findInverse(a,p):
    inverse = extendedEuclideanAlgorithm(a,p)[1] % p
    return inverse

def textToInt(words):
    number = 0
```

```
    i = 0
    for letter in words:
        number += ord(letter)*(256**i)
        i+=1
    return number

def intToText(number):
    words = ""
    while number>0:
        nextLetter = number % 256
        words += chr(nextLetter)
        number = (number-nextLetter)/256
    return words
```

[2]:
```
########## Problem 2
def findRoot(c,e,p,q):
    d = findInverse(e,(p-1)*(q-1))
    m = fastPower(c,d,p*q)
    return m
```

[9]:
```
########## Problem 3
#####Part (a)
def millerRabin(a,n):

    #first throw out the obvious cases
    if n%2 == 0 or extendedEuclideanAlgorithm(a,n)[0]!=1:
        return True

    #Next factor n-1 as 2^k m
    m = n-1
    k = 0
    while m%2 == 0 and m != 0:
        m = m//2
        k = k+1
    #Now do the test:
    a = fastPowerSmall(a,m,n)
    if a == 1:
        return False

    for i in range(0,k):
        if (a + 1) % n == 0:
            return False
        a = (a*a) % n

    #If we got this far a is not a witness
    return True
```

```
#####Part (b)
# This function runs the Miller-Rubin test on 20 random numbers between 2 and p-1.  If it returns true there is a probability of (1/4)^20 that p is prime.
def probablyPrime(p):
    for i in range(0,20):
        a = ZZ.random_element(2,p-1)
        if millerRabin(a,p):
            return False
    return True

#####Part (c)
def findPrime(lowerBound,upperBound):
    while True:
        candidate = ZZ.random_element(lowerBound,upperBound)
        if probablyPrime(candidate):
            return candidate

#####Part (d)
p1 = findPrime(10,100)
p2 = findPrime(1000,10000)
p3 = findPrime(10**99,10**100)
p4 = findPrime(10**499,10**500)

print(p1,p2,p3,p4)
print("Is",p1,"prime?",p1 in Primes())
print("Is",p2,"prime?",p2 in Primes())
print("Is",p3,"prime?",p3 in Primes())
```

```
17 8753 33475476575455665989363411520306608993674006201596802988440160133899616952441274511601057885970643490908704593631303858302564524115048281388755724931054896365722748318719578251501245954384782205610764732082661144899123070629310052951130895811404087386813559170976669707541836909980766858323848583476866475926545650290758616879512054626028264310694342996074645564792755253164210814831453147900699453344063092178171087991449987181859123797917065371233079981332530293376381034853237603480801347008630352003929005207487099915947112713450601561266576299653926934708145591752838059607331407612757526236238530
Is 17 prime? True
Is 8753 prime? True
Is 33475476575455665989363411520306608993674006201596802988440160133899616952441274511601057885970643490 prime? True
```

[4]:
```
##########Problem 4
#####Part (a)
def generateRSAKey(b):

    #Generate some primes
    p = findPrime(2^(b-1),2^b)
```

```
    q = findPrime(2^(b-1),2^b)
    N = p*q
    M = (p-1)*(q-1)
    #next lets find an encryption exponenet
    while True:
        e = ZZ.random_element(2,M-1)
        gcd = extendedEuclideanAlgorithm(e,M)
        if gcd[0]==1:
            d = gcd[1] % M
            break
    publicKey = [N,e]
    privateKey = [N,d]
    return[publicKey,privateKey]

#####Part(b)
def RSAEncrypt(message,PublicKey):
    return fastPowerSmall(message,PublicKey[1],PublicKey[0])

def RSADecrypt(cipher,PrivateKey):
    return fastPowerSmall(cipher,PrivateKey[1],PrivateKey[0])
```

[10]:
```
#########Question 5
keys = generateRSAKey(16)
print("my keys are",keys)
smallPublicKey = keys[0]
smallPrivateKey = keys[1]

m = 314159

c = RSAEncrypt(m,smallPublicKey)
print("Ciphertext is:",c)

m1 = RSADecrypt(c,smallPrivateKey)
print("Decyphered message is:",m1)
```

```
my keys are [[1798864801, 1727276951], [1798864801, 575667287]]
Ciphertext is: 1226350422
Decyphered message is: 314159
```

[20]:
```
########DONT DELETE THIS!!!

#keys = generateRSAKey(512)
#print(keys)

#The two lines above generated the following key.
#Note, this is not the same key I shared above...never share your private key.
```

```
publicKey =␣
 ↪[145602064905073411523471307863632787948776721361386256415242440727639210101257745145626728
 ↪477756451172501264091292748222547292908706647794690995820128555634978341499781263369122178
privateKey =␣
 ↪[145602064905073411523471307863632787948776721361386256415242440727639210101257745145626728
 ↪144783121219996202044210970159268162852069434638347505011111517593955173804001157097197579
```

[13]:
```
########## Here's where I send messages to students

textMessage = "MESSAGE FOR: xxxxx.  Your secret number is xxxx. Respond with␣
 ↪your full name and twice your secret number."
studentPublicKey =␣
 ↪[145602064905073411523471307863632787948776721361386256415242440727639210101257745145626728
 ↪477756451172501264091292748222547292908706647794690995820128555634978341499781263369122178
m = textToInt(textMessage)
c = RSAEncrypt(m,studentPublicKey)

#I also encrypt and decrypt it with my own public/private key pair to make sure␣
 ↪it isn't too large a message for the keys to handle
c1 = RSAEncrypt(m,publicKey)
m1 = RSADecrypt(c1,privateKey)

print(intToText(m1))
print(c)
```

```
MESSAGE FOR: xxxxx.  Your secret number is xxxx. Respond with your full name and
twice your secret number.
646587062737024417484085875845909459353066902711085064638486915582583013843638973
061742020928934198983547740409470865252399758920463236091188456508435772069261863
218035657187643393563375888422226537393708275910289892338665172861269633918857423
902051263136374133784060308364207310598077848861768179744775493577
```

[22]:
```
########## Here is where I decrypted your messages.

studentCipher =␣
 ↪289588402608473404855212480605970766656850310279488446325557949695576367115493725972857434
8

message = RSADecrypt(studentCipher,privateKey)
messageText = intToText(message)
print(messageText)
```

```
Hello.  My name is xxxxxxxx.  Twice my secret number is xxxxx
```

[0]:

[0]:

[0]:

[0]: