Advanced Security Lab 4

# Part C

* DDoS Attack using NTP Amplification
  + NTP amplification is a type of Distributed Denial of Service (DDoS) attack in which the attacker exploits publicly accessible Network Time Protocol (NTP) servers to overwhelm the targeted with User Datagram Protocol (UDP) traffic.
* NTP Spoofing
  + NTP reflection attack is a distributed denial-of-service attack (DDoS) that uses the NTP protocol (network time protocol). The attacker spoofs the source IP address to that of the victim, sends small packets to a vulnerable NTP server, and the NTP server sends a big response to the victim.
* NTP Reflection
  + Reflection attacks involve eliciting a response from a server to a spoofed IP address. The attacker sends a packet with a forged IP address (the victim's) and the server replies to this address.
* The most troublesome problems have involved NTP server addresses hardcoded in the firmware of consumer networking devices. As major manufacturers and OEMs have produced hundreds of thousands of devices using NTP coupled with customers almost never upgrading the firmware of these devices, NTP query storms problems will persist for as long as the devices are in service.
* One particularly common NTP software error is to generate query packets at short (less than five second) intervals until a response is received
  + When placed behind aggressive firewalls that block the server responses, this implementation leads to a never-ending stream of client requests to the variously blocked NTP servers.
  + Such over-eager clients (particularly those polling once per second) commonly make up more than 50% of the traffic of public NTP servers, despite being a minuscule fraction of the total clients.

# Part D

import math

print("RSA ENCRYPTOR/DECRYPTOR")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#Input Prime Numbers

print("PLEASE ENTER THE 'p' AND 'q' VALUES BELOW:")

p = int(input("Enter a prime number for p: "))

q = int(input("Enter a prime number for q: "))

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#Check if Input's are Prime

def prime\_check(a):

    if(a==2):

        return True

    elif((a<2) or ((a%2)==0)):

        return False

    elif(a>2):

        for i in range(2,a):

            if not(a%i):

                return False

    return True

check\_p = prime\_check(p)

check\_q = prime\_check(q)

while(((check\_p==False)or(check\_q==False))):

    p = int(input("Enter a prime number for p: "))

    q = int(input("Enter a prime number for q: "))

    check\_p = prime\_check(p)

    check\_q = prime\_check(q)

#RSA Modulus

n = p \* q

print("RSA Modulus(n) is:",n)

#Eulers Toitent

r= (p-1)\*(q-1)

print("Eulers Toitent(r) is:",r)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#GCD

def egcd(e,r):

    while(r!=0):

        e,r=r,e%r

    return e

#Euclid's Algorithm

def eugcd(e,r):

    for i in range(1,r):

        while(e!=0):

            a,b=r//e,r%e

            if(b!=0):

                print("%d = %d\*(%d) + %d"%(r,a,e,b))

            r=e

            e=b

#Extended Euclidean Algorithm

def eea(a,b):

    if(a%b==0):

        return(b,0,1)

    else:

        gcd,s,t = eea(b,a%b)

        s = s-((a//b) \* t)

        print("%d = %d\*(%d) + (%d)\*(%d)"%(gcd,a,t,s,b))

        return(gcd,t,s)

#Multiplicative Inverse

def mult\_inv(e,r):

    gcd, s, \_ = eea(e,r)

    if(gcd != 1):

        return None

    else:

        if(s<0):

            print("s = %d. Since %d is less than 0, s = s(modr), i.e., s = %d."%(s,s,s%r))

        elif(s>0):

            print("s = %d."%(s))

        return s%r

#e Value Calculation

for i in range(1, 1000):

    if(egcd(i, r) == 1):

        e = i

print("The value of e is:",e)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#d, Private and Public Keys

print("EUCLID'S ALGORITHM:")

eugcd(e, r)

print("END OF THE STEPS USED TO ACHIEVE EUCLID'S ALGORITHM.")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("EUCLID'S EXTENDED ALGORITHM:")

d = mult\_inv(e, r)

print("END OF THE STEPS USED TO ACHIEVE THE VALUE OF 'd'.")

print("The value of d is:", d)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

public = (e, n)

private = (d, n)

print("Private Key is:", private)

print("Public Key is:", public)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#Encryption

def encrypt(pub\_key, n\_text):

    e, n = pub\_key

    x = []

    m = 0

    for i in n\_text:

        if(i.isupper()):

            m = ord(i)-65

            c = (m\*\*e)%n

            x.append(c)

        elif(i.islower()):

            m = ord(i)-97

            c = (m\*\*e)%n

            x.append(c)

        elif(i.isspace()):

            spc = 400

            x.append(400)

    return x

#Decryption

def decrypt(priv\_key,c\_text):

    d,n=priv\_key

    txt=c\_text.split(',')

    x=''

    m=0

    for i in txt:

        if(i=='400'):

            x+=' '

        else:

            m=(int(i)\*\*d)%n

            m+=65

            c=chr(m)

            x+=c

    return x

#Message

message = input("What would you like encrypted or decrypted?(Separate numbers with ',' for decryption):")

print("Your message is: ", message)

#Choose Encrypt or Decrypt and Print

enc\_msg=encrypt(public, message)

print("Your encrypted message is: ", enc\_msg)

enc\_msg = str(enc\_msg)[1:-1]

print("Your decrypted message is: ",decrypt(private, enc\_msg))

print("Thank you for using the RSA Encryptor. Goodbye!")