21BDS0340

Abhinav Dinesh Srivatsa

Cryptography and Network Security Lab

Digital Assignment – IV

**Question 1**

**Code**

def binary\_list(n):

binary = []

while n > 0:

binary.append(int(n % 2))

n = int(n / 2)

return binary

def exponentiate(m, n, mod):

binary\_power = binary\_list(n)

current\_val = m % mod

prod = 1

for i in binary\_power:

if i == 1:

prod = (prod \* current\_val) % mod

current\_val = (current\_val \* current\_val) % mod

return prod % mod

def factors(n):

res = []

i = 2

while n != 1:

if n % i == 0:

n //= i

res.append(i)

continue

i += 1

return res

def totient(n):

f = factors(n)

t = 1

count = 1

for i in range(len(f) - 1):

if f[i] == f[i + 1]:

count += 1

else:

t \*= f[i] \*\* count - f[i] \*\* (count - 1)

count = 1

t \*= f[len(f) - 1] \*\* count - f[len(f) - 1] \*\* (count - 1)

return t

# key setup

p = int(input("enter p: "))

q = int(input("enter q: "))

e = int(input("enter e: "))

n = p \* q

totient\_n = (p - 1) \* (q - 1)

d = exponentiate(e, totient(totient\_n) - 1, totient\_n)

# message encryption

m = int(input("enter m: "))

c = exponentiate(m, e, n)

print(f"encrypted message is {c}")

# message decryption

decrypted = exponentiate(c, d, n)

print(f"decrypted message is {decrypted}")

**Output**

A screenshot of a computer program

Description automatically generated

**Question 2**

**Code**

def binary\_list(n):

binary = []

while n > 0:

binary.append(int(n % 2))

n = int(n / 2)

return binary

def exponentiate(m, n, mod):

binary\_power = binary\_list(n)

current\_val = m % mod

prod = 1

for i in binary\_power:

if i == 1:

prod = (prod \* current\_val) % mod

current\_val = (current\_val \* current\_val) % mod

return prod % mod

def factors(n):

res = []

i = 2

while n != 1:

if n % i == 0:

n //= i

res.append(i)

continue

i += 1

return res

def totient(n):

f = factors(n)

t = 1

count = 1

for i in range(len(f) - 1):

if f[i] == f[i + 1]:

count += 1

else:

t \*= f[i] \*\* count - f[i] \*\* (count - 1)

count = 1

t \*= f[len(f) - 1] \*\* count - f[len(f) - 1] \*\* (count - 1)

return t

def inverse(a, n):

return exponentiate(a, totient(n) - 1, n)

def add(x1, y1, x2, y2, a, mod):

if x1 == x2 and y2 == y1:

return double(x1, y1, a, mod)

l = ((y2 - y1) \* inverse(x2 - x1, mod)) % mod

x3 = (l \* l - x1 - x2) % mod

y3 = (l \* (x1 - x3) - y1) % mod

return x3, y3

def double(x1, y1, a, mod):

l = ((3 \* x1 \* x1 + a) \* inverse(2 \* y1, mod)) % mod

x3 = (l \* l - x1 - x1) % mod

y3 = (l \* (x1 - x3) - y1) % mod

return x3, y3

def negate(x1, y1, mod):

return x1, mod - y1

def multiply(x1, y1, c, a, mod):

xcurr, ycurr = x1, y1

for i in range(c):

xcurr, ycurr = add(x1, y1, xcurr, ycurr, a, mod)

return xcurr, ycurr

# key generation

q = int(input("enter q: "))

a = int(input("enter a: "))

b = int(input("enter b: "))

na = int(input("enter na: "))

gx = int(input("enter gx: "))

gy = int(input("enter gy: "))

pax, pay = multiply(gx, gy, na, a, q)

# encryption

nb = int(input("enter nb: "))

mx = int(input("enter mx: "))

my = int(input("enter my: "))

c1x, c1y = multiply(gx, gy, nb, a, q)

c2x, c2y = multiply(pax, pay, nb, a, q)

c2x, c2y = add(c2x, c2y, mx, my, a, q)

print(f"encrypted message is (({c1x}, {c1y}), ({c2x}, {c2y}))")

# decryption

decx, decy = multiply(c1x, c1y, na, a, q)

decx, decy = negate(decx, decy, q)

decx, decy = add(c2x, c2y, decx, decy, a, q)

print(f"decrypted message is ({decx}, {decy})")

**Output**

