Lab 2: RSA Encryption

March 27th, 2023

William Pearce

DFSC 3316

MWF

“A” Option

**“D” Option Requirements:**

In order to properly encrypt data in 2 byte blocks, the n value had to be increased to at least be larger than 65535, which is the largest decimal number 2 bytes can represent. This is because n has to be a larger number than m, or else data will be lost in the encryption process.

**d\_option.py:**

# the following code was used to check values and generate d value

# p = 137 # prime

# q = 577 # prime

# n = p \* q

# e = 133 # relatively prime to r

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

# k = r + 1 # first number equal to 1 mod r

# d = k / e # divide by e and check validity vv

# while((e\*d % r) != 1 or (d % 1) != 0): # check if coprime and if d is integer

# k += r

# d = k / e

# d = int(d)

# print(f' n = {n}\n r = {r}\n k = {k} \n d = {d}')

bs = int(input('Enter block size (bytes): '))

infile = input('Enter file to encrypt: ')

outfile = input('Enter file to save ciphertext: ')

with open(infile, 'rb') as infile:

with open(outfile, 'wb') as outfile:

with open('key.pub', 'r') as pubkey:

n = int(pubkey.readline())

e = int(pubkey.readline())

data = infile.read(bs)

while data:

msg\_block = int.from\_bytes(data, 'big', signed=False)

cipher\_txt = pow(msg\_block, e, n)

out\_block = cipher\_txt.to\_bytes(bs\*2, 'big')

outfile.write(out\_block)

data = infile.read(bs)

infile = input('Enter file to decrypt: ')

outfile = input('Enter file to save plaintext: ')

with open(infile, 'rb') as infile:

with open(outfile, 'wb') as outfile:

with open('key.priv', 'r') as privkey:

n = int(privkey.readline())

d = int(privkey.readline())

data = infile.read(bs\*2)

while data:

msg\_block = int.from\_bytes(data, 'big', signed=False)

decrypt\_txt = pow(msg\_block, d, n)

out\_block = decrypt\_txt.to\_bytes(bs, 'big')

outfile.write(out\_block)

data = infile.read(bs\*2)

**message.txt:**

Hello World.

Nice Day!

An "A" for the class would be nice. :)-

**key.pub:**

79049

133

**key.priv:**

79049

589

**Console output:**

Enter block size: 2

Enter file to encrypt: message.txt

Enter file to save ciphertext: cipher.txt

Enter file decrypt: cipher.txt

Enter file to save plaintext: decrypted.txt

**cipher.txt:**

(can’t be displayed due to encoding error)

**decrypted.txt:**

Hello World.

Nice Day!

An "A" for the class would be nice. :)-

**“C” Option Requirements:**

**coprime.py:**

def is\_prime(num):

if num == 1:

return False

elif num > 1:

# check for factors

for i in range(2, num):

if (num % i) == 0:

return False

return True

def euclid(a, b):

while b != 0:

temp = a % b

a = b

b = temp

return a

p = int(input('Input value for p: '))

if not is\_prime(p):

print(f'Error: {p} is not a prime number')

q = int(input('Input value for q: '))

if not is\_prime(q):

print(f'Error: {q} is not a prime number')

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

e = int(input('Input value for e: '))

gcd = euclid(e, r)

print(f'e = {e} \n(p-1) \* (q-1) = {r} \ngcd = {gcd}')

if gcd != 1:

print(f'e: {e}, is not relatively prime to (p-1)\*(q-1): {r}')

else:

print(f'e: {e} and (p-1) \* (q-1): {r} are relatively prime')

**Console output:**

Data set 1:

Input value for p: 11

Input value for q: 3

Input value for e: 3

e = 3

(p-1) \* (q-1) = 20

gcd = 1

e: 3 and (p-1) \* (q-1): 20 are relatively prime

Data set 2:

Input value for p: 11

Input value for q: 3

Input value for e: 7

e = 7

(p-1) \* (q-1) = 20

gcd = 1

e: 7 and (p-1) \* (q-1): 20 are relatively prime

Data set 3:

Input value for p: 11

Input value for q: 3

Input value for e: 8

e = 8

(p-1) \* (q-1) = 20

gcd = 4

e: 8, is not relatively prime to (p-1)\*(q-1): 20

**“B” Option Requirements:**