ECE404

HW6

Yudi Wu

0026402011

[wu553@purdue.edu](mailto:wu553@purdue.edu)

**RSA**

**Code**

#!/usr/bin/env python

import sys

import pickle

from PrimeGenerator import \*

from BitVector import \*

import numpy as np

from solve\_pRoot import \*

def keygeneration():

e\_val = 65537

e\_bitvec = BitVector(intVal = e\_val)

while True:

generator = PrimeGenerator(bits = 128, debug = 0)

p = generator.findPrime()

q = generator.findPrime()

p\_bitvec = BitVector(intVal = p,size = 128)

q\_bitvec = BitVector(intVal = q,size = 128)

##check if p and q satisfy three conditions, if not generate new p and q

if p\_bitvec[0] & p\_bitvec[1] & q\_bitvec[0] & q\_bitvec[1]:

if p != q:

if (int(BitVector(intVal = (p-1)).gcd(e\_bitvec)) == 1) & (int(BitVector(intVal = (q-1)).gcd(e\_bitvec)) == 1):

break

##calculte totient\_n and d

n = p \* q

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

tn\_bitvec = BitVector(intVal = totient\_n)

d\_bitvec = e\_bitvec.multiplicative\_inverse(tn\_bitvec)

key = [e\_val, int(d\_bitvec), n, p, q]

return key

def encryption(inputfile, outfile, e,n,p,q):

bv = BitVector(filename = inputfile)

fptr = open(outfile,'w')

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file(128)

while bitvec.length() < 128:

bitvec += BitVector(textstring = "\n")

bitvec.pad\_from\_left(128)

##Use CRT to calculate power % n

vp = pow(int(bitvec),e,p)

vq = pow(int(bitvec),e,q)

p\_bitvec = BitVector(intVal = p,size = 128)

q\_bitvec = BitVector(intVal = q,size = 128)

xp = q \* int(q\_bitvec.multiplicative\_inverse(p\_bitvec))

xq = p \* int(p\_bitvec.multiplicative\_inverse(q\_bitvec))

data = (vp \* xp + vq \*xq) % n

BitVector(intVal = data,size = 256).write\_to\_file(fptr)

##print(BitVector(intVal = data,size = 256).get\_bitvector\_in\_hex())

fptr.close()

def decryption(inputfile, outfile,d,n,p,q):

bv = BitVector(filename = inputfile)

fptr = open(outfile,'w')

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file(256)

##Use CRT to calculate power % n

vp = pow(int(bitvec),d,p)

vq = pow(int(bitvec),d,q)

p\_bitvec = BitVector(intVal = p,size = 128)

q\_bitvec = BitVector(intVal = q,size = 128)

xp = q \* int(q\_bitvec.multiplicative\_inverse(p\_bitvec))

xq = p \* int(p\_bitvec.multiplicative\_inverse(q\_bitvec))

data = (vp \* xp + vq \*xq) % n

BitVector(intVal = data,size = 128).write\_to\_file(fptr)

##print(BitVector(intVal = data,size = 128).get\_bitvector\_in\_hex())

fptr.close()

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) != 4:

print("Error: wrong number of arguments")

exit()

if sys.argv[1] == '-e':

key = keygeneration()

encryption(sys.argv[2],sys.argv[3],key[0],key[2],key[3],key[4])

fptr = open('key.txt','w')

pickle.dump(key,fptr)

fptr.close()

elif sys.argv[1] == '-d':

with open ('key.txt') as fptr:

key = pickle.load(fptr)

##print(key[1])

##print(key[3])

##print(key[4])

decryption(sys.argv[2],sys.argv[3],key[1],key[2],key[3],key[4])

**Result**

**Encrypted text in Hex**

4d9681c40459334e304ae7f86cb12eb0669d6f7def40186df2cea068b0dab2ac

a6c63ba6e9d740ad66587c2021270df7fedb05490795dcaac796a7339aad5966

29474a520e395066d590cbcf87d28c91dee1f9c380253b0c81aae18aeb6d81d9

942f5e13759d20e513948459a0387040c4654385e87ef8c0a22d17b9dbe43807

2b2893a7b03330b3002c73ba8d2de994431c0ebf214d00e3048e1b9c73eaebb8

5823d9686ddca47430b09769d97e558a4eb4a906b66b3ff5d1c69c79d7a7c99f

8065338625911aef69160529c220f578858343a2a4d53436d981f62371ea26b4

6928545134c03078c95f40a13ca46d3183bece82983ada9a3516b5f7ee563964

725626aad5b53a755ee5e96b3d48dace22a176fae5f59b989ac792cc23c92f3d

0fe87a3c5f25727df03880c08a46ccb6673525b827bcae2a3c1ee5056dd3d9d7

1cd907f02f1f056fe775c0146256b39c66c9c8072bde9262f1962f05dc1aac29

591006e3336e960cf69f9b42c94adb27ba6633d7f3d9e98931b6dd86df2ddd58

**Decrypted text in Hex**

4c69666527732062757420612077616c

6b696e6720736861646f772c20612070

6f6f7220706c61796572207468617420

73747275747320616e64206672657473

2068697320686f75722075706f6e2074

686520737461676520616e6420746865

6e206973206865617264206e6f206d6f

72652e20497420697320612074616c65

20746f6c6420627920616e206964696f

742c2066756c6c206f6620736f756e64

20616e6420667572792c207369676e69

6679696e67206e6f7468696e672e0d0a

**Decrypted text in ASCII**

Life's but a walking shadow, a poor player that struts and frets his hour upon the stage and then is heard no more. It is a tale told by an idiot, full of sound and fury, signifying nothing.

**D**

10492775928840364572810847938072758951945711672130007334941390549922630732033

**P**

291403807707174521434716511117408954141

**Q**

272372520758896986895845329717850829033

**RSA CRACK**

**Code**

#!/usr/bin/env python

import sys

import pickle

from PrimeGenerator import \*

from BitVector import \*

import numpy as np

from solve\_pRoot import \*

def keygeneration():

e\_val = 3

e\_bitvec = BitVector(intVal = e\_val)

while True:

generator = PrimeGenerator(bits = 128, debug = 0)

p = generator.findPrime()

q = generator.findPrime()

p\_bitvec = BitVector(intVal = p,size = 128)

q\_bitvec = BitVector(intVal = q,size = 128)

if p\_bitvec[0] & p\_bitvec[1] & q\_bitvec[0] & q\_bitvec[1]:

if p != q:

if (int(BitVector(intVal = (p-1)).gcd(e\_bitvec)) == 1) & (int(BitVector(intVal = (q-1)).gcd(e\_bitvec)) == 1):

break

n = p \* q

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

tn\_bitvec = BitVector(intVal = totient\_n)

d\_bitvec = e\_bitvec.multiplicative\_inverse(tn\_bitvec)

key = [e\_val, int(d\_bitvec), n, p, q]

return key

def encryption(inputfile,e,n,p,q):

bv = BitVector(filename = inputfile)

output = BitVector(size = 0)

while(bv.more\_to\_read):

bitvec = bv.read\_bits\_from\_file(128)

while bitvec.length() < 128:

bitvec += BitVector(textstring = "\n")

bitvec.pad\_from\_left(128)

vp = pow(int(bitvec),e,p)

vq = pow(int(bitvec),e,q)

p\_bitvec = BitVector(intVal = p,size = 128)

q\_bitvec = BitVector(intVal = q,size = 128)

xp = q \* int(q\_bitvec.multiplicative\_inverse(p\_bitvec))

xq = p \* int(p\_bitvec.multiplicative\_inverse(q\_bitvec))

data = (vp \* xp + vq \*xq) % n

output += BitVector(intVal = data,size = 256)

return output

if \_\_name\_\_ == '\_\_main\_\_':

key1 = keygeneration()

key2 = keygeneration()

key3 = keygeneration()

file1 = encryption(sys.argv[1],key1[0],key1[2],key1[3],key1[4])

##print(file1.get\_bitvector\_in\_hex())

##print()

##print(key1[2])

##print()

file2 = encryption(sys.argv[1],key2[0],key2[2],key2[3],key2[4])

##print(file2.get\_bitvector\_in\_hex())

##print()

##print(key2[2])

##print()

file3 = encryption(sys.argv[1],key3[0],key3[2],key3[3],key3[4])

##print(file3.get\_bitvector\_in\_hex())

##print()

##print(key3[2])

##print()

N\_val = key1[2] \* key2[2] \* key3[2]

N1 = N\_val/key1[2]

N2 = N\_val/key2[2]

N3 = N\_val/key3[2]

N1\_MI = BitVector(intVal = N1).multiplicative\_inverse(BitVector(intVal = key1[2]))

N2\_MI = BitVector(intVal = N2).multiplicative\_inverse(BitVector(intVal = key2[2]))

N3\_MI = BitVector(intVal = N3).multiplicative\_inverse(BitVector(intVal = key3[2]))

fptr = open(sys.argv[2],'w')

for i in range(0,len(file1),256):

bitvec1 = file1[i+0:i+256]

bitvec2 = file2[i+0:i+256]

bitvec3 = file3[i+0:i+256]

M\_3 = (int(bitvec1) \* N1 \* int(N1\_MI) + int(bitvec2) \* N2 \* int(N2\_MI) + int(bitvec3) \* N3 \* int(N3\_MI)) % N\_val

M = solve\_pRoot(3,M\_3)

BitVector(intVal = M, size = 128).write\_to\_file(fptr)

##print(BitVector(intVal = M, size = 128).get\_bitvector\_in\_hex())

fptr.close()

**Result**

**Encrypted text 1**



**n1**

101091715750483326187981027852750635293769679395912127556722558276491731908183

**Encrypted text 2**



**n2**

93994491382432519021472872928048141858057552326074843048549607440006449119951

**Encrypted text 3**



**n3**

80876193365623001939077725994344763548795035931633503938265481608630262768019

**Cracked text in hex**

4c69666527732062757420612077616c

6b696e6720736861646f772c20612070

6f6f7220706c61796572207468617420

73747275747320616e64206672657473

2068697320686f75722075706f6e2074

686520737461676520616e6420746865

6e206973206865617264206e6f206d6f

72652e20497420697320612074616c65

20746f6c6420627920616e206964696f

742c2066756c6c206f6620736f756e64

20616e6420667572792c207369676e69

6679696e67206e6f7468696e672e0d0a

**Cracked text in ASCII**

Life's but a walking shadow, a poor player that struts and frets his hour upon the stage and then is heard no more. It is a tale told by an idiot, full of sound and fury, signifying nothing.