Homework Number: 06
Name: Yi Qiao
ECN Login: qiao22
Due Date: 02/26/2019

Result

Part 1

p: 264326933705609766920147123337637190039
q: 329072960787296093812735848547851562909
n: 86982846690332337001247453950354282745617187287836178510428741138069196663451

Encrypted text

783465d9df2995d4167b47eda3139e551ecaeb5fbbd93c9bb65ad201a59fb43a18edfd2e8430c77b68f8c54
eeb913c29a5f2802fffda6ceaf8435b5d94731725269b0089fd24624ce6ed1bfd683c24fae9ef77d89b0a1a1f
3570f4027aa6f46bc014894cd13e4aa1bec63ac1fe852dcb89d4cff4916c7bd4630b53c39c9008bd303051a
5c55089ecc84e64bff461efc0f422956a4ce18394fb41114277832c3e153ef4aba32c78a7df1461cd4eb3553f
a39a29d9fbf057f9fbc7f632fd3c1f8f1675130221c10ae68dd6f063e6e9ec599563f99f4b932e67c4871c8471
d72e9978de2d8955a3d800327c1a5ef87e7766789d33c747c3c91ac630ed3df06b890152140d6227a98e7df
0ffdeb23cbdff27f7b40927f10685022258a77b073975437036badc83a37eea6be297a3a43dc18ddac550dc
710c3cd27c59e23f72dd601d5f7d261099fc2c736b7fb275e5c5c38a161c1086e55bf366ae7f16be325dfb60
4a4fb0bae7ad8c0e3184c28103ce48021ca57c5ed811b7e44536aa456b8a0613

Decrypted text

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.

Part 2

Receiver 1

n:1121798129024364688046341727854958966854114610827482245149865669544634015273339
e656f2643c105260780b7c7ee76be3126bfc8d2406ff012a30b434336f42019fe7606a38a8d8d0c7227a3c81
c309402b58b8a7aa491a927c7c716aa7bd1543fa9991446ba30958949b744e4179ff0097827f7c2e4cca998
645e7b9f5ecaf8c19905cf2a225031b1e9e78aa8ea11e7d8c3de05a4a0bcde19bbf854d24f593b7b181b376
af3abd3ababd7a8114543cab4b2f3a21e4b9d4052cfa222974407d6fbc70860bab472cadc8bf7aa9252453
e1d180de3d837d414ee55d622f42af0959031dd0ef86b46e3d059255206181308d73a58b3f958b60ad9494
7e270b432e1b023221985a6637d54be024b6b42aeba280b4f0635d6b1d362654411f757ff95c9cad502c39
eab772734b565807f4fac971d8a4194be0100fe31f4d22dbfd4988c6b03c3fb99f46cade0ebe2a557e917bf1f
77ee837c9d03a9abcf1438ae6207b073f8832bcdbe0694b7a21a2066250789809c0f3ebcfc491802f5f62778
0271a3f6ed02b937cb68c33b1fb421c0b6322db3c9d3243c009c7b55d083808182b7358

Receiver 2

n:8848240529951571100354292496126570498446505425549286654562955859161715271 7557
220aba0d9e613809005d97b9f13e4b2e80eff1c18e63db0c46c7ebaad4e10ff719aee1e908bb7ac80fbd7ab9
b14b1c5b2903830d27bfb69b0b464a7d535b42b4aabf94247cbac3952dc4653c18f0d86788f149fa96c7c1
bb472dffdc6a4cde871ca6b5479925b87aefc899984fa1d5941ecefe2e1466b40168478289e96604eb48a0b
b369f885abc60de8ede5f0f103ccded7cc92ae92074caf59d6eade0274dad6bef692082d467463f001027b12
8ed11687c65be075d9e11ec497eff27f3ee6a45b9c3fe6082ee8e57be47d3407446b980b16285b71c5b931e
b464dc912315bab52546fe44d3f937eea43d149e6009418a68fa08335a5cfe4a002f467a4a84297e905c7e37
82de324214f25af57f712db300645e34bd25b2fcf02d27dc53ac6b9bc75a30362c20e1bc2265dd5f09fa6920
179f52550c467a714dcdd95848bebd4d3d23830dc6dca5fd2de61ec8345fee728c0f473fc8ad8338151ebb2
ab105b0bff0c2151df27cb7e0a8b7803ab5b8affc9c0ced39261ce23abf9565e594f8

Receiver 3

n:8656644840686106395490844849255716649528638625673217762558323836823666064 1963
743c275152b52f032ea6367302acc38271520d0b73d966a1ac0608e809b07cf7b5d2b7ac7e0e4948dba5d3
6ebcb190bc116115af33b84b5ae38ed4285f031ae03c130188f6ccd3434a4c5ba7dbc9b0e2a518affa1fb217
58182dabb95d1e3501aafb89c76611bb3a16d5e52f687fa41ca681b7fe1bff402939cb7dc325b6c99e8d701
324ba14fc04f9f21c370c69b4f8ddc25a8fcf53792653aef0f22a2f0de40cb518fc3e525969cb908ddae12367
d993ae9c50ba9d4edcabc71ad4c17264367883cac3b4836bdb7d300833b88d0636b50b177b9ca21c3f52a
6ed46a8930e2b8cf295d1f7e497dc8c785e46762af3b4070692bd104c4c764847d242c083f1ec2503621b86
e1f02bcce6db21cc4ed0cf406f62817b9adaea6a430881838d9c429b7258be5a6f2137d26796f003b24a1d3
b28035c4e7bc0e71c5eda3297d31beb5565a62322b5807d0779328e2b27d4244b758251e25ee122dbd7de
13c314c8525b80dcaf69eef31c59049f740150bc6d77d4b22a1803e5b1d502cb24dc805dc3

Cracked Output

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.

Code

RSA

```
#! /usr/bin/env python3

from PrimeGenerator import *
from BitVector import *
import os
import sys

e = 65537 # public key

def gcd(a, b):
    if b == 0: return a
    return gcd(b, a%b)


def RSAEncryption(bv, key, n):
    '''
    Encrypt msg with key using modulus n
    '''
    bv.pad_from_right(128-len(bv))  # pad to the right if less then 128 bits
    bv.pad_from_left(128)           # pad to the 128 bits to the left make a
                                    # total of 256 bits

    return BitVector(intVal=pow(int(bv), int(key), n),size=256)


def RSADecryption(bv, key, n):
    '''
    Decrypt msg with key using modulus n
    '''
    bv = BitVector(intVal = pow(int(bv), int(key), n),size=256)
    return bv[-128:] # take the right most 128 bits


def generateN():
    '''
    This function is used to generate two prime numbers p, q while guaranteeing:
    1. p != q
    2. p, q are coprime to e
    3. 2 left most bits of p and q are set
    ret: p, q, n
    '''
    generator = PrimeGenerator(bits = 128)

    p = -1
    q = -1
    while(True):
        p = generator.findPrime()
        q = generator.findPrime()
        while(q == p):
            q = generator.findPrime()
        if gcd((p-1),e) == 1 and gcd((q-1),e) == 1:
            break
    with open("p.txt", "w") as fp:
        print(p, end='', file=fp)

    with open("q.txt", "w") as fp:
        print(q, end='', file=fp)

    print("p and q generated")


def get_pq_from_file():
    '''
    This function is used to read p and q form file
    ret: p, q
    '''
    with open("p.txt", "r") as fp:
        p = int(fp.read())

    with open("q.txt", "r") as fp:
        q = int(fp.read())

    return p, q


def main():
    if len(sys.argv) != 4:
        print("""Usage: ./rsa.py <flag> <input file> <output file>
        flags:
            -e: encryption
            -d: decryption""")
        sys.exit(1)

    if sys.argv[1] == "-e":
        # encryption mode
        if not os.path.exists("p.txt") or not os.path.exists("q.txt"):
            generateN()

        e_Bv = BitVector(intVal = e)
        p, q = get_pq_from_file()
        n = p * q
        print("p: %d"%(p))
        print("q: %d"%(q))
        print("n: %d"%(n))

        with open(sys.argv[2], "r") as fp:
            msg = fp.read()

        msg_bv = BitVector(textstring=msg)

        with open(sys.argv[3], "w") as fp:
            for i in range(0, len(msg_bv), 128):
                encryptedMsg = RSAEncryption(msg_bv[i:min(len(msg_bv),i+128)],
                 ↪  e_Bv, n)
                fp.write(encryptedMsg.get_bitvector_in_hex())


    elif sys.argv[1] == "-d":
        # decryption mode
        if not os.path.exists("p.txt") or not os.path.exists("q.txt"):
            print("no p and q found, encrypt first")
            sys.exit(1)

        e_Bv = BitVector(intVal = e)
        p, q = get_pq_from_file()
        n = p * q

        d_Bv = e_Bv.multiplicative_inverse(BitVector(intVal = (p-1)*(q-1)))
        with open(sys.argv[2], "r") as fp:
            msg = fp.read()

        msg_bv = BitVector(hexstring=msg)

        with open(sys.argv[3], "wb") as fp:
            for i in range(0, len(msg_bv), 256):
                decryptedMsg = RSADecryption(msg_bv[i:i+256], d_Bv, n)
                if i + 256 < len(msg_bv):
                    decryptedMsg.write_to_file(fp)
                else:
                    for k in range(0, 128, 8):
                        if int(decryptedMsg[k:k+8]) != 0:
                            decryptedMsg[k:k+8].write_to_file(fp)

    else:
        print("""Usage: ./rsa.py <flag> <input file> <output file>
        flags:
            -e: encryption
            -d: decryption""")
        sys.exit(1)
```

```python
if __name__ == "__main__":
    main()
```

breakRSA

```
#! /usr/bin/env python3

from PrimeGenerator import *
from BitVector import *
from solve_pRoot import solve_pRoot as sp
import sys

e = 3

def gcd(a, b):
    if b == 0: return a
    return gcd(b, a%b)


def generateN():
    '''
    This function is used to generate two prime numbers p, q while guaranteeing:
    1. p != q
    2. p, q are coprime to e
    3. 2 left most bits of p and q are set
    ret: p, q, n
    '''
    generator = PrimeGenerator(bits = 128)

    p = -1
    q = -1
    while(True):
        p = generator.findPrime()
        q = generator.findPrime()
        while(q == p):
            q = generator.findPrime()
        if gcd((p-1),e) == 1 and gcd((q-1),e) == 1:
            break

    return p,q,p*q


def RSAEncryption(bv, key, n):
    '''
    Encrypt msg with key using modulus n
    '''

    bv.pad_from_right(128-len(bv))  # pad to the right if less then 128 bits
    bv.pad_from_left(128)           # pad to the 128 bits to the left make a
                                    # total of 256 bits
    return BitVector(intVal=pow(int(bv), int(key), n),size=256)


def CRT(rs):

    '''
    This function is used to find the desired number with Chinese Reminder
↪   Theorem
    rs is a list of tuples containing (n, reminder using modulus n)
    '''
    prod = 1
    for r in rs:
        prod *= r[0]

    coeff = [prod // r[0] for r in rs]

    sum = 0
    for index, r in enumerate(rs):
        mi = int(BitVector(intVal =
         ↪  coeff[index]).multiplicative_inverse(BitVector(intVal=r[0])))
        sum += (r[1] * mi % r[0]) * coeff[index]
        sum %= prod

    return sum


if __name__ == "__main__":

    if len(sys.argv) != 3:
        print("Usage: ./breakRSA.py <input_file> <output_file>")
        sys.exit(1)

    # generate p, q, n for 3 receivers
    receivers = [generateN() for i in range(3)]

    e_bv = BitVector(intVal = e)
    with open(sys.argv[1], "r") as fp:
        msg = fp.read()

    msg_bv = BitVector(textstring=msg)
    encryptedMsg = [BitVector(size=0) for _ in range(3)]

    # for each receiver, encrypt the message with their respectively public key
    for index, receiver in enumerate(receivers):

        with open("receiver_%d_encryted.txt"%(index), "w") as fp:
            fp.write("n:%d\n"%(receiver[2]))
            for i in range(0, len(msg_bv), 128):
```

```
            encryptedMsg[index] +=
            ↪  RSAEncryption(msg_bv[i:min(len(msg_bv),i+128)], e_bv,
            ↪  receiver[2])
        fp.write(encryptedMsg[index].get_bitvector_in_hex())

# crack RSA with CRT
with open(sys.argv[2], "wb") as fp:
    for i in range(0, len(encryptedMsg[0]), 256):
        rs = [(receivers[j][2], int(encryptedMsg[j][i:i+256])) for j in
        ↪  range(len(receivers))]
        msg_cube = CRT(rs)
        print("block %d cracked"%(i//256))
        sys.stdout.flush()

        # take cube root
        msg = BitVector(intVal = sp(3, msg_cube), size=128)
        if i + 256 < len(encryptedMsg[0]):
            msg.write_to_file(fp)
        else:
            for k in range(0, 128, 8):
                if int(msg[k:k+8]) != 0:
                    msg[k:k+8].write_to_file(fp)
```