Assignment 5

Graded

Group

KUNAL SAINI
KUSHAL MAHESHWARI
SHARATH KUMAR V
View or edit group

Total Points

85 / 100 pts

Question 1

Team Name

0 / 0 pts

✔ **+ 0 pts** Correct

Question 2

Commands

5 / 5 pts

✔ **+ 5 pts** Correct

Question 3

Cryptosystem

5 / 5 pts

✔ **+ 5 pts** Correct

Question 4

Analysis

Resolved **65** / 80 pts

- ✔ **+ 5 pts** Encoding used in the cryptosystem, i.e., odd positions contains [f−m] whereas even positions contains [f-u]
- ✔ **+ 10 pts** Correctly explain why A seems to be a lower triangular matrix. Reason: For the ith plaintext byte, changing any byte at j>i does not change the corresponding ith ciphertext byte.
- ✔ **+ 10 pts** However, changing any byte at j<i changes the corresponding ith ciphertext byte
- ✔ **+ 10 pts** Compute diagonal elements of A: Brute force each aii independtly
- ✔ **+ 10 pts** Correct A: A is a lower triangular matrix with correct values.
- ✔ **+ 10 pts** Correct E

  Solution 2: Brute forcing the plaintext vector
- ✔ **+ 10 pts** Final plaintext password without padding

Regrade Request **Submitted on: May 06**

1. I have reasoned that the inputs should be 'ff' to 'mu' using the given field GF(128) still, marks have been deducted for it.
2. I have explained that changing the ith bit changes all bits greater than equal to i, so we choose a lower triangular Matrix. Still have been deducted for it.
3. I have represented the password in ASCII representation with padding, marks have not been awarded for that also
4. I have written my final password without padding in plain text also, marks have not been awarded for that also

No MSB in ff-mu

Reviewed on: May 07

Question 5

Password

10 / 10 pts

- ✔ **+ 10 pts** Correct

Question 6

Code

0 / 0 pts

- ✔ **+ 0 pts** Correct

Q1 Team Name
0 Points

Group Name

hardwired

Q2 Commands
5 Points

List all the commands in sequence used from the start screen of this level to the end of the level

go->wave->dive->go->read->password->c->wtpuxvpygl->c

Q3 Cryptosystem
5 Points

What cryptosystem was used at this level?

EAEAE (A similar to AES cryptosystem where the E is a invertible non-linear function and A is the matrix transformation or shuffling)

Q4 Analysis
80 Points

Knowing which cryptosystem has been used at this level, give a detailed description of the cryptanalysis used to figure out the password.

After getting to the cryptosytem. It was said that each block was 8 bytes. After several input-outputs we found that all the letters in the ciphertexts where from f-u like the previous assignment. So we assumed that the plaintext is also from f-u. f-u is 16 letters to represent this only 1/2 byte is required. So the block must 16 letters each. And it is also given that the field is GF(128). Therefore our plaintext space should be of 2 letters (like 'ff'). To get 128 field we can have 2 letter space like 'ff' till 'mu' where the mapping is {f:0,g:1,h:2,..u:15}. If we give a block of all zeros, then the ciphertext must also be same as plaintext. This was confirmed when we gave plaintext as 'ffffffffffffffff', we recieved same as the ciphertext. Later we change last byte and tried. The ciphertext also changed only in last byte. Later we observed that if we change the ith byte of the plaintext as non zero and rest as zero we found that in ciphertext till (i-1)th byte all were zero and from the ith byte we found the non-zero elements. This is only possible when our transformation matrix is lower triangular. Thus we came to the conculsion that our matrix is lower triangular.
Later we gave plaintext which contains only the ith byte as zero and rest as zero. Then we analysed ith byte of the ciphertext. If x is the ith byte of plaintext then the corresponding byte of ciphertext came out to be f(x) = (a[i][i]*(a[i][i]* (x^e[i]))^e[i])^e[i]. Using this we found out the possible values of a[i][i] and e[i]. We implemented all the operations over finited field of 128 with the generator x^7 + x + 1.
We found out that there were 3 possible pairs for a[i][i] and e[i] for each i.
To break the ties between them we started to analyse the (i+1)th bit of ciphertext(ith bit of plaintext is non-zero rest zero). We found the that it was using only one extra element in the a matrix. It was basically using 2 diagonal elements i,i and i+1,i+1. And the element i+1,i. We tried all the possible values for this element this also helped us break the ties between the diagonal elements. After this we have found the diagonal elements and its neighbouring elements and all the elements of the e(exponent matrix).
Now we for the rest of the elements since we know the exponent matrix we brute force(try all 128 values for each element) and find the matrix a.
The matrix a and e came out to be(transpose of a is given below)
a = [[84, 119, 13, 124, 98, 30, 14, 69], [0, 70, 31, 26, 56, 38, 124, 14], [0, 0, 43, 5, 6, 24, 23, 76], [0, 0, 0, 12, 111, 45, 102, 28], [0, 0, 0, 0, 112, 97, 3, 13], [0, 0, 0, 0, 0, 11, 82, 69], [0, 0, 0, 0, 0, 0, 27, 10], [0, 0, 0, 0, 0, 0, 0, 38]]

e = [23, 115, 39, 79, 86, 49, 23, 28]

Now the password was 18 bytes. So we divided into halfs.Now to decrypt the password we used the same way i.e. we iterated over all possible values for a block and check if our EAEAE function's output is same as the current password block we have. By doing this we got the password in bytes. Later we tried to map using the f-u map but the answer which came was not accepted. So we simply mapped using the ASCII values which gave us the answer wtpuxvpygl000000, which was also not accepted then we removed the trailing zeros and tried. The password wtpuxvpygl was accepted.

Q5 Password
10 Points

What was the password used to clear this level?

wtpuxvpygl

Q6 Code
0 Points

Please add your code here. It is MANDATORY.

gen_out.py

Download

```python
import subprocess
import sys

def generate_ciphertext_via_ssh(input_list):
    intxt = ['hardwired', 'hardwired', '5','go', 'wave', 'dive', 'go', 'read', 'password', 'c']

    for s in input_list:
        intxt.extend([s,'c'])
    intxt.extend(['back','exit'])
    file = open("input_ssh.txt","w")
    for i in intxt:
        file.write(i)
        file.write("\n")
    file.close()

    process = subprocess.Popen('./extract.sh')
    process.wait()
    search_line = 'Slowly, a new text starts appearing on the screen. It reads ...\n'
    file = open("extracted.txt", "r")
    out = file.readlines()

    x=out.index(search_line)
    out = out[x:]
    output_list = []
    for i,s in enumerate(out):
        if s==search_line:
            output_list.append(out[i+1][2:-1])
    return output_list




file_in = open("plaintext.txt","r")
file_out = open("ciphertext.txt","w")

inpu = []

out = file_in.readlines()
for x in out:
    inpu.append(x[:-1])

out_list = generate_ciphertext_via_ssh(inpu)
password = out_list.pop(0)
for i in out_list:
    file_out.write(i)
    file_out.write("\n")
```

```
47 file_out.close()
```

extract.sh Download

```
1 #!/bin/sh
2
3 sshpass -p cs641 ssh -tt student@172.27.26.188 < input_ssh.txt > extracted.txt
```

crackEAEAE.py

Download

```python
from pyfinite import ffield

Field = ffield.FField(7)

powr = {}

dict = {0:'f',
        1:'g',
        2:'h',
        3:'i',
        4:'j',
        5:'k',
        6:'l',
        7:'m',
        8:'n',
        9:'o',
        10:'p',
        11:'q',
        12:'r',
        13:'s',
        14:'t',
        15:'u',}

for x in range(0,128):
    powr[x] = {}

def byte_str(k):
    res = ""
    res += dict[int(k/16)]
    n = k%16
    res += dict[n]
    return res

def str_block(ch):
    res = 16*(ord(ch[0])-ord('f')) + 1*(ord(ch[1])-ord('f'))
    return res

def cal_powr(base, n):
    if base in powr:
        if n in powr[base]:
            return powr[base][n]

    if base == 1:
        return 1

    result = 0
```

```
    if n == 0:
        result = 1
    elif n == 1:
        result = base
    elif n%2 == 0:
        base_root = cal_powr(base, n>>1)
        result = Field.Multiply(base_root, base_root)
    else:
        base_root = cal_powr(base, n>>1)
        result = Field.Multiply(base_root, base_root)
        result = Field.Multiply(base, result)

    powr[base][n] = result
    return result

def possible_diagonal(plain, cipher, possiAd, possiE, index):

    redu_possiAd = []
    redu_possiE = []

    for A in possiAd:
        for E in possiE:
            ispossible = True
            for i,p in enumerate(plain):
                k = cal_powr( Field.Multiply( cal_powr( Field.Multiply( cal_powr(plain[i][index], E), A), E), A), E)
                if cipher[i][index] != k:
                    ispossible = False
            if ispossible:
                redu_possiAd.append(A)
                redu_possiE.append(E)

    return redu_possiAd, redu_possiE

def break_ties(plain, cipher, possiAd, possiE, index):

    for a in range(0, 128):
        x = 0
        while (len(possiE[index+1])>x):
            y = 0
            while (len(possiE[index])>y):
                flag = True
                for i in range(0, len(plain)):
                    if cipher[i][index+1] != cal_powr(Field.Multiply(cal_powr(Field.Multiply(cal_powr(plain[i][index], possiE[index][y]), possiAd[index][y]), possiE[index][y]), a) ^ Field.Multiply(cal_powr(Field.Multiply(cal_powr(plain[i][index], possiE[index][y]), a), possiE[index+1][x]), possiAd[index+1][x]), possiE[index+1][x]):
                        flag = False
```

```
                    break
            if flag:
                possiE[index+1] = [possiE[index+1][x]]
                possiAd[index+1] = [possiAd[index+1][x]]
                possiE[index] = [possiE[index][y]]
                possiAd[index] = [possiAd[index][y], a]
            y = y+1
        x = x+1

    return possiAd, possiE

def vector_addition(v1, v2):
    res = []
    for i in range(0, len(v1)):
        res.append(v1[i] ^ v2[i])

    return res

def scalar_vector_multi(v1, k):
    res = []
    for i in range(0, len(v1)):
        res.append(Field.Multiply(v1[i], k))

    return res

def applyA(A, v):
    res = [0]*8
    for r, e in zip(A, v):
        res = vector_addition(scalar_vector_multi(r, e), res)

    return res

def applyEAEAE(plain, A, E):
    res = [0]*8
    for i in range(0, len(plain)):
        res[i] = cal_powr(plain[i], E[i])

    res = applyA(A, res)

    for i in range(0, len(res)):
        res[i] = cal_powr(res[i], E[i])

    res = applyA(A, res)

    for i in range(0, len(res)):
        res[i] = cal_powr(res[i], E[i])

    return res

```

```
140 def complete_matrix(plain, cipher, A, E, index):
141
142     for idx in range(0,6):
143         offset = idx + 2
144
145         exp = [x[0] for x in E]
146         Ad = [[0 for i in range(0,8) ] for j in range(0,8)]
147
148         for i in range(0,8):
149             for j in range(0,8):
150                 Ad[i][j] = 0 if len(A[i][j]) == 0 else A[i][j][0]
151
152         for index in range(0,8):
153             if index + offset > 7:
154                 continue
155
156             plain1 = plain[index*15: (index+1)*15]
157             cipher1 = cipher[index*15: (index+1)*15]
158
159             for i in range(0, 128):
160                 Ad[index][index+offset] = i
161                 flag = True
162                 for j,p in enumerate(plain1):
163                     if cipher1[j][index+offset] != applyEAEAE(p, Ad, exp)[index+offset]:
164                         flag = False
165                         break
166                 if flag:
167                     A[index][index+offset] = [i]
168
169     comp_A = [[0 for i in range(0,8)] for j in range(0,8)]
170
171     for i in range(8):
172         for j in range(8):
173             comp_A[i][j] = 0 if len(A[i][j]) == 0 else A[i][j][0]
174
175     exp = [x[0] for x in E]
176
177     return comp_A, exp
178
179 plain = open("plaintext.txt",'r')
180 cipher = open("ciphertext.txt", 'r')
181
182 plain_list = plain.readlines()
183 cipher_list = cipher.readlines()
184
185 plain_list = [x[:-1] for x in plain_list]
186 cipher_list = [x[:-1] for x in cipher_list]
187
188 byte_plain = []
```

```
byte_cipher = []

for p in plain_list:
    x = []
    for i in range(0, len(p), 2):
        x.append(str_block(p[i:i+2]))
    byte_plain.append(x)

for c in cipher_list:
    x = []
    for i in range(0, len(c), 2):
        x.append(str_block(c[i:i+2]))
    byte_cipher.append(x)

possiAii = []
possiEi = []

for i in range(0,8):

    possiAd = [ x for x in range(0,128) ]
    possiE = [ x for x in range(0,128) ]

    possiAd, possiE = possible_diagonal(byte_plain[i*15: (i+1)*15], byte_cipher[i*15: (i+1)*15], possiAd, possiE, i)

    possiAii.append(possiAd)
    possiEi.append(possiE)

# print(possiAii)
# print(possiEi)

for i in range(0,7):

    possiAii, possiEi = break_ties(byte_plain[i*15: (i+1)*15], byte_cipher[i*15: (i+1)*15], possiAii, possiEi, i)

# print(possiAii)
# print(possiEi)

A = [ [ [] for x in range(0,8) ] for y in range(0,8) ]
E = [ x[0] for x in possiEi ]

for i in range(0,8):
    for j in range(0,8):
        if i == j:
            A[i][j] = [possiAii[i][0]]
        if i == j-1:
            A[i][j] = [possiAii[i][1]]

```

```
comp_A, E = complete_matrix(byte_plain, byte_cipher, A, possiEi, i)

print(comp_A)
print(E)

def Decrypt(password):
    byte_password = []

    for i in range(0, len(password), 2):
        byte_password.append(str_block(password[i:i+2]))

    decrypted_password = ""
    byte_dp = []

    for i in range(0, len(byte_password)):
        for j in range(0, 128):
            check = decrypted_password + byte_str(j) + (16-len(decrypted_password)-2)*'f'
            byte_check = []
            for a in range(0, len(check), 2):
                byte_check.append(str_block(check[a:a+2]))
            if byte_password[i] == applyEAEAE(byte_check, comp_A, E)[i]:
                decrypted_password += byte_str(j)
                byte_dp.append(j)
                break

    return byte_dp

password1 = "msltfplimqhimsim"
password2 = "ltmhlplminksjiir"

byte_dp1 = Decrypt(password1)

byte_dp2 = Decrypt(password2)

print(byte_dp1)
print(byte_dp2)

pass1 = ""
for x in byte_dp1:
    pass1 += chr(x)

pass2 = ""
for x in byte_dp2:
    pass2 += chr(x)

print(pass1+pass2)
```