Q1 Team Name

0 Points

ANV

Q2 Commands

5 Points

List the commands used in the game to reach the ciphertext.

Commands Used :-

- **go**
- **wave**
- **dive**
- **go**
- **read**

After executing the above commands in sequence, we reach where we

Q3 Analysis

50 Points

Give a detailed description of the cryptanalysis used to figure out the password. (Use Latex wherever required. If your solution is not readable, you will lose marks. If necessary the file upload option in this question must be used TO SHARE IMAGES ONLY.)

■Given that :-

Each block is of size 8 bytes as 8x1 vector over F_{128} -- constructed using the degree 7 irreducible polynomial x^7 + x + 1 over F_2.
There were two transformations: first a linear transformation given by invertible 8x8 key matrix A with elements from F_{128} and second an exponentiation given by 8 x 1 vector E whose elements are numbers between 1 and 126. E is applied on a block by taking ith element of the block and raising it to the power given by the ith element in E.
The transformations are applied in the sequence EAEAE on the input block to obtain the output block. Both E and A are part of the key.

We used an **expect** script to automate cipher text generation from the server, for the given plain texts.

Using **expect** script , we captured all output actions of each ciphertext generation using **Logs** , and then using a Python script we retrieved Ciphertexts respectively.

■Observation :-

First, we randomly entered some plain texts each of length 16 and got some cipher texts. Our observation from these cipher texts was that the only the characters from **f to u** were present in the cipher text. Each character is 4 bits. So 2 characters are 8 bits which is a byte.
The size of the field F is 128. So, we guessed the bytes present would be from **ff to mu**. We verified by observing many cipher texts and we concluded that our observation is very correct. We assume the byte

ff to be mapped to 0, fg to be mapped to 1, ...

We can say that the characters in the even position contain characters from 'f' to 'u'. The characters in the odd position contain characters from 'f' to 'm'.

The **Matrix Multiplication Transforms** were similar to that of **AES**.

We wanted to observe the changes that were made to the cipher text by changing the bytes in the plain text and we wanted to figure out if we could find any pattern. We decided that if we found any pattern, then we can use that pattern to our advantage to decode the cipher text. If we did not find any pattern, then we wanted to discover some other way to decode the cipher text.

■Analysis :-

Each plain text was of length 16. That means, we have total **8 bytes**.
Ex: plain text is **fmghqspfhiqtilmu**. Then the bytes are
fm, gh, qs, pf, hi, qt, il, mu
We tried out different ways of changing plain text and observing the corresponding cipher texts. Some of these different ways of changing plain text we used are:

- Keeping one byte of characters constant and changing other byte of characters
- Keeping all the bytes constant and changing only one byte
- Changing some arbitrary bytes at the same time and keeping some arbitrary bytes constant etc.

We found out a pattern using the following procedure:

Change the i^{th} byte in plain text and keep the remaining bytes constant where i takes the values from 1 to 8.

So, a total of 1024 plain texts and their corresponding cipher texts were observed.

The pattern we found out was that the k^{th} byte of the cipher text will be affected by the m^{th} byte of the plain text if and only if m<=k
That means, 1 st byte of cipher text will depend upon (1st byte of the plain text).
Second byte of the cipher text will depend upon (1st and 2nd byte of the plain text),
.....
8th byte of cipher text will depend upon (1st, 2nd ..... and 8th byte of the plain text).

The above pattern might be possible if A is a lower triangular matrix.
So, using this above pattern, we can iteratively find out our desired plain text from the cipher text.

When we press password in the server, we got the cipher text as **lhhklqfgiqfhisljljipgjfpgmjtktho** . This is the encrypted password. We divided this into two blocks. First block is **lhhklqfgiqfhislj** and second block is **ljipgjfpgmjtktho**. We will call each of these as "password_cipher_block".

We will do the following for each "password_cipher_block":
First, we will take a variable named "partial_password". This means the password plain text that we have found till this particular iteration. It is empty string at the start of 1st iteration.
For 'k' in range 1 to 8, we will do the following:
We will first find 128 plain texts like this:
We will fix the first k-1 bytes of each plain text to the partial password. Then, for k^{th} byte, we will consider the characters from 'ff' to 'mu' (total 128) and set the bytes from k+1 to 8 as 'ff'. Total length of each of the plain text is 16. In this way, we have obtained a total of 128 plain texts. Then, for each of these, we will find the cipher texts using our expect script.
We will denote cipher text with the symbol **zc** and our plain text with the

symbol **zp**.
We will find out the cipher text **zc** which has the k_th byte same as the k_th byte of our password cipher block.
That means, we find zc such that zc[k_th_byte] == password_cipher_block[k_th_byte]
After we find this zc, we will consider it's corresponding plain text zp.
The corresponding plain text of the cipher text **zc** is **zp**. Then the k_th byte of the **zp** will be same as the k_th byte of the password plain block
We then update our partial_password like this:

partial_password= partial_password + zp[k_th byte]

partial_password for first block of password_cipher_block is "mklqmilpmnlgmimj"
partial_password for second block of password_cipher_block is "lglpifififififif"
We will combine the partial_password values from both blocks to get:
mklqmilpmnlgmimjlglpifififififif

We will take each byte from above text. We will map
ff to 0, fg to 1, ... mu to 128.
We will get a total of 16 values from this process. We considered these values as ascii values and reversed mapped them to characters.
Then, we will get the following text: **uksjxastaj000000**.
We removed the zeroes on the end to get the final decrypted password as:
Decrypted password: **uksjxastaj**
We entered the above password **uksjxastaj** to clear the level.

In this way, we have found out the password without finding out the matrices E and A.

No files uploaded

Q4 Password

5 Points

What was the final commands used to clear this level?

The password used to clear this level is :- **uksjxastaj**

Q5 Codes

0 Points

It is mandatory that you upload the codes used in the cryptanalysis. If you fails to do so, you will be given 0 for the entire assignment.

cipher.sh Download

```
1  gnome-terminal --disable-factory -- expect inputs1.exp
2
3
```

decode.py Download

```
1  import os
2
3  password=""
4
5  def func():
6      x=open("output.txt")#extracting ciphertext from logs.
7      file=open("ciphers.txt",'w')
```

```
    c=0
    for i in x:
        if "\t" in i:
                i=i.replace("\n","").replace("\t","").strip()
                if len(i)==16:
                        file.write(i+"\n")
                        c+=1

    file.close()
    x.close()

block=1
for cipher_password in ("lhhklqfgiqfhislj","ljipgjfpgmjtktho"):

    partial_password=""



    for l in range(8):

        file=open('input.txt','w')

        for i in range(8):

            for j in range(16):

                z=partial_password+chr(ord('f')+i)+chr(ord('f')+j)+'f'*(16-2*l-2)
                #z='f'*(16-2*l-2)+chr(ord('f')+i)+chr(ord('f')+j)+partial_password

                file.write(z+"\n")

        file.close()
```

```
        open("output.txt",'w').close()

        #os.system("gnome-terminal -- expect inputs.exp")
        os.system("sh cipher.sh")
        func()
        #break
        file=open("ciphers.txt",encoding='utf-8')

        data=list(file)

        file.close()

        for i in range(8):

            for j in range(16):

                z=data.pop(0).replace("\n","")

                #if z[14-l:16-l]==cipher_password[14-l:16-l]:
                if z[2*l:2*l+2]==cipher_password[2*l:2*l+2]:
                        partial_password+=chr(ord('f')+i)+chr(ord('f')+j)
                    #partial_password=chr(ord('f')+i)+chr(ord('f')+j)+partial_password
        print("Block :",block,", Byte:",l+1)

    password+=partial_password
    block+=1
passw=""
fi={}
c=0
for i in range(8):
    for j in range(16):
        fi[chr(ord('f')+i)+chr(ord('f')+j)]=c
        c+=1
for i in range(0,len(password),2):
    passw+=chr(fi[password[i:i+2]])
```

```
passw="".join(i for i in passw if i!='0')
print("Decrypted Password is:  ",passw)

```

inputs1.exp

Download

```
#!/usr/bin/expect -f
log_file output.txt
set timeout 5
spawn ssh -o StrictHostKeyChecking=no students@172.27.26.188

expect "*?assword:*"
send -- "cs641a\r"

expect "*group name*"

send -- "ANV\r"

expect "*assword*"

send -- "anv@1998\r"

expect "*start at:*"

send "5\r"

expect "*some deep underground well!*"

send -- "go\r"

expect "*nothing to grab ...*"

send -- "wave\r"

expect "*yields the same result!*"
```

```
send -- "dive\r"

expect "*somewhere deep inside...*"

send -- "go\r"

expect -- "*Who designed all this?*"

send -- "read\r"

expect "*the screen...*"

set f [open "input.txt"]
set inputs [split [read $f] "\n"]
close $f

foreach inp $inputs {

send -- "$inp\r"

expect "*ress c to continue>*"


send -- "c\r"

}

log_file

```

readme.txt Download

```
TO run the attack, do the following:

```

```
        press "make" in the terminal in the directory of all the files without quotes.

be sure to be connected to the iitk network.


Also, make sure the internet is very fast. We have set the timeout in expect script to
5 seconds.
If your internet is very slow, make sure to change the timeout accordingly in the
expect script named "inputs1.exp"

The attack will take around 10 to 20 minutes to complete on normal internet speed.
ON fast internet speed, it will take atmost 3 minutes.

```

▼ Makefile Download

```
run: install
        python3 decode.py

install:
        sudo apt-get install -y expect


```

GROUP

Dibbu Amar Raja
Vikas
Idamakanti Venkata Nagarjun Reddy
View or edit group

TOTAL POINTS

60 / 60 pts

QUESTION 1

Team Name — **0** / 0 pts

QUESTION 2

Commands — **5** / 5 pts

QUESTION 3

Analysis — **50** / 50 pts

QUESTION 4

Password — **5** / 5 pts

QUESTION 5

Codes — **0** / 0 pts