Q1 Commands
5 Points

List the commands was used in this level?

go ,enter ,pluck , c ,c ,back ,give,
back, back ,thrnxxtzy ,read
,the_magic_of_wand

Q2 Cryptosystem
10 Points

What cryptosystem was used in the game to reach the password?

We used Monoalphabetic Substitution Cipher and Block Cipher named Permutation Cipher Cryptosystem in the game to reach the password. Substitution -Permutation Networks (SPN)
Networks were used in the cryptosystem.

Q3 Analysis
30 Points

What tools and observations were used to figure out the cryptosystem and the password? (Explain in less than 1000 lines)

Tools:
(i) Used python script to check whether the given ciphertext is
encrypted with "SHIFT CIPHER (CAESAR CIPHER)" or not.
(ii) Used python script (attached in answer 6) to find frequency
of each letter and bigrams in the ciphertext.
(iii) Used table showing letter frequencies (unigram, bigram) in
English language from the lecture slides and internet.
(iv) Used python script to apply decryption using permutation cipher using the keys.

(v) Used python script to apply decryption using substitution cipher using the keys .

Observations:
1. We used python script to check if the ciphertext is encrypted with shift cipher or not. We found out that none of the 26 possibilities resulted in a meaningful text. Hence the possibility of encryption using shift cipher was rejected on the basis of our observation in the context.

2. Then we proceed to check whether it is encrypted with Affine cipher and substitution cipher or not using Frequency analysis methodology which was discussed profoundly in the earlier lectures by the professor . The key in the mono-alphabetic substitution cipher which will be the essential tool in the decryption of the ciphertext defines a map from each letter of the plaintext alphabet to some (only one)letter of the ciphertext alphabet where the map can be arbitrary subject only to the constraint that it be one-one so that decryption is possible. As a result, the key space contains all of the alphabet's bijections or permutations.

3. When using English alphabets, the key Space is of size 26!=26*25.."1 or approximately 2^88, making brute-force attack impossible. The reason behind this claim is that if we assume that letter 'a' is mapped to any of the 26 letters and letter 'b' is mapped to any 25 remaining letters because letter 'a' has been already mapped to a letter ,we have to exclude that letter , similarly ,letter 'c' can be mapped to 24 letters, letter 'd' can be mapped to 23 letter and so on till letter 'z' will have only one choice, totaling the total number of possibilities to be factorial of 26. So we are left with the choice of analyzing the counts of the letters to decrypt the ciphertext. Hence we go for frequency analysis which is going to utilize the statistical

patterns of alphabets in English language text .The statistical patterns of english alphabets
are more or less fixed hence this attacks work for the monoalphabetic substitution ciphers
where by analyzing the frequencies of the letters in the ciphertext we will relate it to the
normal distribution of the alphabets of english language text and will find the correlation
between the letters of the plaintext and the ciphertext .

4. The letters frequencies we got after studying the statistical patterns of the letters were same
as the standard letters frequencies in the English language text, but to our surprise we saw
that when we substitute the corresponding letters to obtain the plaintext we are not getting
any meaningful text which made us to think on the ciphertext that there may be some more
cryptographic technique applied along with monoalphabetic substitution cipher in the
encryption phase.

5. After analyzing unigrams , bigrams and trigrams frequency of the letters in the ciphertext
,We observed some interesting patterns in the ciphertext which are as follows :
None of the unigrams ,bigrams , or trigrams are repeated which lead us to an idea that there
is a mixing operation or more specifically permutation operation applied on the blocks of
plaintext to obtain the corresponding ciphertext, To further strengthen our claim we were
further sure that substitution cipher is not the only encryption algorithm used in current
scenario, because none of the two letter and three letter words are repeated , We were able
to draw a conclusion over this observation that there may be some other type of encryption
done along with substitution cipher and this puts our obvious attention over permutation
cipher where the blocks of alphabets of plaintext may be jumbled or rearranged to produce
the ciphertext.

6. The BLOCK Cipher named permutation cipher may have been applied with a very high
possibility. First a permutation over all the block of letters must have been applied and then
the substitutions must have been applied over the generated intermediate ciphertext or
there we can apply reverse that first substitution has been applied over whole plaintext and
then the intermediate ciphertext may have been divided into blocks of block size which we
have found later and the the permutation has been applied similarly on all the blocks.

7. Our obvious challenges were to find the block length ,the permutation and the substitution
over the plaintext which we cracked by using the following procedures:

8. Firstly, we have done frequency analysis in which we have passed a string of text as an
input and returns a dictionary containing the frequency of each letter in the input text.
It is shown in the uploaded code .

9. To find the block size n, We tried to analyze all the repeated group of letters namely some
bigrams and trigrams like as follows:
(a): 'xja' was repeated twice and gap of letters between the first occurrence of 'xja' and the
second occurrence of 'xja' is 80 letters.
(b): 'eas' was repeated twice and gap of letters between the first occurrence of 'eas' and the
second occurrence of 'eas' is 60 letters.
(c): 'fv' was repeated thrice and gap of letters between the first occurrence of 'fv' and the
second occurrence of 'fv' is 35 letters.
(d): Gap of letters between the second occurrence of 'fv' and the third occurrence of 'fv' is
45 letters.

10. Since 5 is the common factor in all the gap lengths there was very high possibility that 5 is

the block length of this particular permutation block cipher.

11. After that to find the permutation key , We tried to analyze 3-letter words (trigrams) which
can be all possible combination of unique 'the' word.

12. 'xja' ,'lnf', 'eas', 'ugf', 'pqq' were the words we found. the five -letter blocks which contain
this words are 'fvxja', 'lhfav', 'veasf' , 'ugfav' , 'yppqq' . Some were seperating in different
blocks so we have negleted them.

13. Now we calculated the common 3 letters in the five length block and we found that 'v' ,'a',
'f' were the common letters which had very high possibility of being the word 'the'.
The 'yppqq' block does not have this so we assumed that it may be some different word.

14. Now we looked at the words that are not common, so we now analyze this not common
words step by step.
(a): Now 'x' and 'j' are not common in 'fvxja'. They are present in 3rd and 4th position .To
form the full fledged letter 'the' by 'v' , 'a', 'f' . This not common words should come in
1st and 2nd positions . So We come to conclusion that {3,4) -> {1,2}.

(b):Now 'l' and 'h' are not common in 'lhfav'. They are present in 1st and 2nd position .To
form the full fledged letter 'the' by 'v' , 'a', 'f' . This not common words should come in
4th and 5th positions . So We come to conclusion that {1,2) -> {4,5}.

(c): Now 'e' and 's' are not common in 'veasf'. They are present in 2nd and 4th position .To
form the full fledged letter 'the' by 'v' , 'a', 'f' . This not common words should come in
1st and 5th positions . So We come to conclusion that {2,4) -> {1,5}.

(d): Now 'u' and 'g' are not common in 'ugfav'. They are present in 1st and 2nd position .To
form the full fledged letter 'the' by 'v' , 'a', 'f' . This not common words should come in
4th and 5th positions . So We come to conclusion that {1,2) -> {4,5}.

So, We have mapped the non common words to their possible places.

15. {3,4) -> {1,2}, {1,2) -> {4,5}, {2,4) -> {1,5}, {1,2) -> {4,5} are the possible mappings, Now we
come to a conclusion from this mappings that the permutation cipher must have used this
mappings which we have concluded below:
3->2, 4->1, 2->5, 1->4, 5->3 .

16. So the permutation key we got after doing all this analysis is " 4 5 2 1 3".
This is the encryption key we got . Now to decrypt the ciphertext we have to apply the
decryption key which is "4 3 5 1 2 ".

17. The ciphertext which we had to decrypt is
" qmnjvsa nv wewc flct vprj tj tvvplvl fv xja vqildhc xmlnvc nacyclpa fc gyt vfvw. fv wgqyp, pqq
pqcs y wsq rx qmnjvafy cgv tlvhf cw tyl aeuq fv xja tkbv cqnsqs. lhf avawnc cv eas fuqb qvq tc
yllrqr xxwa cfy. psdc uqf avrqc gefq pyat trac xwv taa wwd dv eas flcbq. vd trawm vupq quw x
decgqcwt, yq yafl vlqs yqklhq! snafq vml lhvqpawr nqg_vfusr_ec_wawy qp fn wgawdgf."

18. The intermediate ciphertext we got after applying decryption permutation key " 4 3 5 1 2"
and after removing the punctuations and spaces is
"jnvqmvnwsafclewpvrcttjvjtvllvpjxafvlidvqmxlhcncanvlcpcy gcyafvfwtvgwqfvqpqypscypqrqxwsjnvqmcygafvlhvttwyfcu eqlajxafvvbctkqssqnafvlhcncawsafveqbvuqyclqtrqxlrcafxw dscypafvuqgcerqypafqarctttvaxwdwdawsafveqbvlcarwdtp uqmvxwdquqgcecqyywtvllafqykqssqnlhvqmafvhqllrwnpaf

vuqgcewsrqypawgwafnwdgf".

19. The substitution key we used is as follows given in the mapping:

key = {'a': 't', 'b': 'v', 'c': 'i', 'd': 'u', 'e': 'c', 'f': 'h', 'g': 'g', 'h': 'p', 'i': 'q', 'j': 'b',
'k':'j' 'l': 's', 'm': 'k', 'n': 'r', 'p': 'd', 'o' : ?? , 'q': 'a', 'r': 'w', 's': 'f', 't': 'l',
'u': 'm', 'v': 'e', 'w': 'o', 'x': 'y', 'y': 'n', 'z': ?? ,
'.' : '.' , '_' :' _' '!' : '!' , ' ' : ' ' , ',' :','}.

19. After that we applied the substitution key and we got the final ciphertext as follows:

"breakerofthiscodewillbeblessedbythesqueakyspiritresidingintheholegoaheadandfindawayofbreakingthespellonhimcastbytheeviljaffarthespiritofthecavemanisalwayswithyoufindthemagicwandthatwillletyououtofthecavesitwouldmakeyouamagiciannolessthanjaffarspeakthepasswordthemagicofwandtogothrough

Mapped Ciphertext: breaker of this code will be blessed by the squeaky spirit residing in the hole. go ahead, and find a way of breaking the spell on him cast by the evil jaffar. the spirit of the cave man is always with you. find the magic wand that will let you out of the caves. it would make you a magician, no less than jaffar! speak the password the_magic_of_wand to go through."

Q4 Password
5 Points

What was the final command used to clear this level?

the_magic_of_wand

Q5 Codes

0 Points

Upload any code that you have used to solve this level.

Modern_Cryptology3 _a.ipynb Download

Shift Cipher: Trying to check whether shift cipher is meaningful or not

In [22]:
```
def bruteforce_shift_cipher(ciphertext):
    for i in range(1, 26):
        plaintext = ""
        for char in ciphertext:
            if char.isalpha():
                char_code = ord(char)
                if char.isupper():
                    char_code -= i
                    if char_code < ord('A'):
                        char_code += 26
                elif char.islower():
                    char_code -= i
                    if char_code < ord('a'):
                        char_code += 26
                plaintext += chr(char_code)
            else:
                plaintext += char
        print(f"Key: {i}, Plaintext: {plaintext}")
```

In [23]:
```
ciphertext ="qmnjvsa nv wewc flct vprj tj tvvplvl fv xja vqildhcxmlnvc nacyclpa fc gyt vfvw. fv wgqyp, pqq pqcs y wsq rx qmnjvafy cgv tlvhf cw tyl aeuq fv
```

```
xja tkbv cqnsqs.lhf avawnc cv eas
fuqb qvq tc yllrqr xxwa cfy. psdc
uqfavrqc gefq pyat trac xwv taa wwd
dv eas flcbq. vd trawmvupq quw x
decgqcwt, yq yafl vlqs yqklhq!
snafq vmllhvqpawr nqgvfusrecwawy qp
fn wgawdgf."
bruteforce_shift_cipher(ciphertext)
```

```
Key: 1, Plaintext: plmiurz mu vdvb ekbs
Key: 2, Plaintext: oklhtqy lt ucua djar
Key: 3, Plaintext: njkgspx ks tbtz cizq
Key: 4, Plaintext: mijfrow jr sasy bhyp
Key: 5, Plaintext: lhieqnv iq rzrx agxo
Key: 6, Plaintext: kghdpmu hp qyqw zfwn
Key: 7, Plaintext: jfgcolt go pxpv yevm
Key: 8, Plaintext: iefbnks fn owou xdul
Key: 9, Plaintext: hdeamjr em nvnt wctk
Key: 10, Plaintext: gcdzliq dl mums vbs
Key: 11, Plaintext: fbcykhp ck ltlr uar
Key: 12, Plaintext: eabxjgo bj kskq tzq
Key: 13, Plaintext: dzawifn ai jrjp syp
Key: 14, Plaintext: cyzvhem zh iqio rxo
Key: 15, Plaintext: bxyugdl yg hphn qwn
Key: 16, Plaintext: awxtfck xf gogm pvm
Key: 17, Plaintext: zvwsebj we fnfl oul
Key: 18, Plaintext: yuvrdai vd emek ntk
Key: 19, Plaintext: xtuqczh uc dldj msj
Key: 20, Plaintext: wstpbyg tb ckci lri
Key: 21, Plaintext: vrsoaxf sa bjbh kqh
Key: 22, Plaintext: uqrnzwe rz aiag jpg
Key: 23, Plaintext: tpqmyvd qy zhzf iof
Key: 24, Plaintext: soplxuc px ygye hne
Key: 25, Plaintext: rnokwtb ow xfxd gmd
```

Frequency Analysis : Unigrams

In [24]:

```
def
frequency_analysis(ciphertext):

    freq_dict = {}
    for char in ciphertext:
        if char.isalpha():
            if char in freq_dict:
                freq_dict[char] +=
1
            else:
                freq_dict[char] =
```

```
1

    freq_dict =
dict(sorted(freq_dict.items(),
key=lambda item: item[1],
reverse=True))

    print("Frequency:", freq_dict)
```

In [25]:

```
ciphertext = "qmnjvsa nv wewc flct
vprj tj tvvplvl fv xja vqildhc
xmlnvc nacyclpa fc gyt vfvw. fv
wgqyp, pqq pqcs y wsq rx qmnjvafy
cgv tlvhf cw tyl aeuq fv xja tkbv
cqnsqs.lhf avawnc cv eas fuqb qvq
tc yllrqr xxwa cfy. psdc uqfavrqc
gefq pyat trac xwv taa wwd dv eas
flcbq. vd trawm vupq quw x
decgqcwt, yq yafl vlqs yqklhq!
snafq vml lhvqpawr
nqg_vfusr_ec_wawy qp fn wgawdgf."
frequency_analysis(ciphertext)
```

```
Frequency: {'q': 30, 'v': 29, 'a': 23,
```

Bigram Analysis

In [26]:

```
def
letter_pair_analysis(ciphertext):

    freq_dict = {}
    for i in
range(len(ciphertext)-1):
        if ciphertext[i].isalpha()
and ciphertext[i+1].isalpha():
            letter_pair =
ciphertext[i] + ciphertext[i+1]
            if letter_pair in
freq_dict:

freq_dict[letter_pair] += 1
            else:

freq_dict[letter_pair] = 1

    freq_dict =
dict(sorted(freq_dict.items(),
key=lambda item: item[1],
```

```
reverse=True))

    print("Frequency:", freq_dict)
```

In [27]:

```
ciphertext = "qmnjvsa nv wewc flct
vprj tj tvvplvl fv xja vqildhc
xmlnvc nacyclpa fc gyt vfvw. fv
wgqyp, pqq pqcs y wsq rx qmnjvafy
cgv tlvhf cw tyl aeuq fv xja tkbv
cqnsqs.lhf avawnc cv eas fuqb qvq
tc yllrqr xxwa cfy. psdc uqfavrqc
gefq pyat trac xwv taa wwd dv eas
flcbq. vd trawm vupq quw x
decgqcwt, yq yafl vlqs yqklhq!
snafq vml lhvqpawr
nqg_vfusr_ec_wawy qp fn wgawdgf."
letter_pair_analysis(ciphertext)
```

```
Frequency: {'aw': 5, 'fv': 4, 'fl': 3,
```

Trigram Analysis

In [28]:

```
import re
from collections import Counter

def trigram_analysis(ciphertext):
    # Remove non-letter characters
and convert to lowercase
    ciphertext = re.sub(r'[^a-z]',
'', ciphertext.lower())

    # Count frequency of trigrams
    trigrams = [ciphertext[i:i+3]
for i in range(len(ciphertext)-2)]
    trigram_counts =
Counter(trigrams)

    # Print top 10 most frequent
trigrams
    for trigram, count in
trigram_counts.most_common(10):
        print(trigram, count)
```

In [29]:

```
ciphertext = "qmnjvsa nv wewc flct
vprj tj tvvplvl fv xja vqildhc
xmlnvc nacyclpa fc gyt vfvw. fv
```

```
wgqyp, pqq pqcs y wsq rx qmnjvafy
cgv tlvhf cw tyl aeuq fv xja tkbv
cqnsqs.lhf avawnc cv eas fuqb qvq
tc yllrqr xxwa cfy. psdc uqfavrqc
gefq pyat trac xwv taa wwd dv eas
flcbq. vd trawm vupq quw x
decgqcwt, yq yafl vlqs yqklhq!
snafq vml lhvqpawr
nqg_vfusr_ec_wawy qp fn wgawdgf."
trigram_analysis(ciphertext)
```

```
qmn 2
mnj 2
njv 2
flc 2
lvl 2
fvx 2
vxj 2
xja 2
fvw 2
pqq 2
```

In [ ]:

Modern_cryptology3_b.ipynb

Download

In [2]:

In [31]:

```
#code to remove all the spaces and
special character
import re

cipher_text = "qmnjvsa nv wewc
flct vprj tj tvvplvl fv xja
vqildhc xmlnvc nacyclpa fc gyt
vfvw. fv wgqyp, pqq pqcs y wsq rx
qmnjvafy cgv tlvhf cw tyl aeuq fv
xja tkbv cqnsqs. lhf avawnc cv eas
fuqb qvq tc yllrqr xxwa cfy. psdc
uqf avrqc gefq pyat trac xwv taa
wwd dv eas flcbq. vd trawm vupq
quw x decgqcwt, yq yafl vlqs
yqklhq! snafq vml lhvqpawr
nqg_vfusr_ec_wawy qp fn wgawdgf."

ciphertext = cipher_text.strip()
```

```
ciphertext = re.sub(r"[^a-zA-Z0-9\s]", "", ciphertext)
ciphertext = re.sub(r"\s+", "", ciphertext)

print(ciphertext)
```

```
qmnjvsanvwewcflctvprjtjtvvplvlfvxjavqil
```

In [32]:

```
def permutation_decipher(ciphertext):
    key = [4, 3, 5, 1, 2] #that we have founded
    block_size = 5



    # number of full_blocks in the ciphertext
    num_blocks = (len(ciphertext)) // block_size

    # number of last_letter in the ciphertext
    last_block_letter = (len(ciphertext)) % block_size

    last_letters = ciphertext[-4:]



    # Initialize a list to hold the plaintext blocks
    plaintext_blocks = [''] * num_blocks

    # Loop through each block of the ciphertext
    for i in range(num_blocks):
        # Initialize a list to hold the characters in the block
        block_chars = [''] * block_size

        # Loop through each character in the block
        for j in range(block_size):
            # Calculate the index of the character in the ciphertext
```

```
            ciphertext_index = i *
block_size + key[j] - 1

            # If the calculated
index is out of range, use a space
character
            ciphertext_char =
ciphertext[ciphertext_index] if
ciphertext_index < len(ciphertext)
else ' '

            # Store the character
in the block list
            block_chars[j] =
ciphertext_char

        # Join the characters in
the block list into a string
        plaintext_block =
''.join(block_chars)

        # Add the plaintext block
to the plaintext blocks list
        plaintext_blocks[i] =
plaintext_block

    # Join the plaintext blocks
into a single string and return it
    plaintext =
''.join(plaintext_blocks)
    return plaintext+last_letters



new_cipher_test =
permutation_decipher(ciphertext)
print(new_cipher_test) # Output:
PLAINTEXTXAMPLECEXT
```

```
jnvqmvnwsafclewpvrcttjvjtvllvpjxafvlidv
```

In [47]:

```
def ciphertext_mapping(new_cipher_test,

    mapped_ciphertext = []
    for char in new_cipher_test:

        mapped_ciphertext.append(key[ch
```

```
    mapped_ciphertext =
''.join(mapped_ciphertext)
    print(mapped_ciphertext)

    output=""

    punctuations =
set(['!','@','#','$','%','^','&','(',')
',',+','=','{','}',';',':','/','?','|','
',',',',',~','`','[',']','-'])
    j = 0
    for i in range(len(cipher_text)):

        if cipher_text[i] in punctuatio
            output += cipher_text[i]
        else:
            output += mapped_ciphertext
            j+= 1



    print("Mapped Ciphertext:", output)

key = {'a': 't', 'b': 'v', 'c': 'i', 'd
'e': 'c', 'f': 'h', 'g': 'g', 'h': 'p',
'q', 'j': 'b', 'l': 's', 'm': 'k', 'n':
'p': 'd', 'q': 'a', 'r': 'w', 's': 'f',
'l', 'u': 'm', 'v': 'e', 'w': 'o', 'x':
'y': 'n','k':'j'}

ciphertext_mapping(new_cipher_test, key
```

```
breakerofthiscodewillbeblessedbythesque
Mapped Ciphertext: breaker of this code
```

In [ ]:

In [ ]:

Q6 Group name

0 Points

d2ce09fd5842b342d5b3b66d10b6daef

Assignment 3

Graded

Group

RAJ KUMAR
MADHAV MAHESHWARI
GUNJ MEHUL HUNDIWALA
View or edit group

Total Points

46 / 50 pts

Question 1

Commands — **5** / 5 pts

Question 2

Cryptosystem — R **8** / 10 pts

Question 3

Analysis — R **28** / 30 pts

Question 4

Password — **5** / 5 pts

Question 5

Codes — **0** / 0 pts

Question 6

Group name — **0** / 0 pts