```
#2.21 Below are given four examples of ciphertext, one obtained from a Substitu tion Cipher, one from a Vigenere Cipher ` , one from an Affin
#unspecified. In each case, the task is to determine the plaintext.Give a clearly written description of the steps you followed to decrypt ea
# susbsitution Ciper
# Exercise 2.1

ciphertext = "EMGLOSUDCGDNCUSWYSFHNSFCYKDPUMLWGYICOXYSIPJCKQPKUGKMGOLICGINCGACKSNISACYKZSCKXECJCKSHYSXCGOIDPKZCNKSHICGIWYGKKGKGOLDSILKGOIUSIG


# Lets create  a dictionary to store the frequency of each letter in the ciphertext
freq = {}
for letter in ciphertext:
    if letter in freq:
        freq[letter] += 1
    else:
        freq[letter] = 1

# Sort the dictionary by descending frequency
sorted_freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

# Lets print the frequency to see the output and try to substitute some values
print("Ctxt: A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z")
print("Freq:", end=" ")
for letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
    if letter in freq:
        print(f"{freq[letter]:<4}", end="")
    else:
        print(" "*4, end="")
print()
print("Rank:", end=" ")
for item in sorted_freq:
    print(f"{item[1]:<4}", end="")
print()
print("Ptxt:", end=" ")
for item in sorted_freq:
    print(item[0], end=" ")


# First we ran the code above and after analyzing the output generate the frequency table and add the respecitve letters to substitutio using
substitution_dict = {
    'C': 'e',  # C is the most highest occurance
    'Q': 'j',  # in analysis both only occured once from what we see
    'Z': 'h',  #  there is 7 ZC and HE is the most frequent in our diagram also there is 4 ZCN and HER has high probability too
    'N': 'l',  # It was just a guess
    'U': 't',  # THE is most frequent and there UZC
    'S': 'o',  # there will be l-ved there and when checked both o and i were tried and the in the output o made more sense
    'O': 'n',  # GO occured 5 times, and it is frequent
    'K': 's',
    'I': 'd', # IGCI which can be dead -ea-
    'A': 'v',  # I->d substitution, NCG-C is lea-e => leave
    'W': 'g',  # There is WYGKK which is "grass"
    'L': 'y', # alwa-s => always
    'X': 'p',  #res-e-ted = > "respected"
    'J': 'c',  # like above respe-ted should be "respected"
    'E': 'i',  #veh-cle is "vehicle"
    'P': 'u',  # prod-ces is "produces"
    'D': 'b',  # "-ought" => "bought"
    'M': 'm', # I-aynot" => "I may not
    'G': 'a', # all of the others are trail and hit guess
    'Y': 'r',
    'F': 'w', # given in the question as a hint
    'H': 'f',
    'V': 'y',
    'B': 'k'  # I had manually added all the dictionary after manually adding and cheking. If you remove the letter from the dictionaries you
}

plaintext = ""
for char in ciphertext:
    if char in substitution_dict:
        plaintext += substitution_dict[char]
    else:
        plaintext += char
print("\n")
print(plaintext)
```

```
Ctxt: A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
Freq: 5       37  8   12  9   24  5   15  7   18  7   5   13  10  6   1       20      14      5   7   15  13
Rank: 37  24  20  18  15  15  14  13  13  12  10  9   8   7   7   7   6   5   5   5   5   1
Ptxt: C G S K Y I U N Z E O F D L X J P M W H A Q

imaynotbeabletogrowflowersbutmygardenproducesjustasmanydeadleavesoldovershoespiecesofropeandbushelsofdeadgrassasanybodysandtodayiboughta
```

In this substituion cipher we have first calculated the frequency and then assigned the characters for the letters by calculating the probaility of the letters in English alphabets. Then we use logic and frequency table with the probability of 2 combination and three combination letters and get the substitution key and then we just substitute.

Here in this code I wanted to add it at once and get the outputs but there would be a lots of results and it is a lengthy time so I checked in once and again and added in the dictionary one at a time. If you remove letters from the dictionary it will show the same thing

```
#21.b
#Vignere Cipher

import string

# First of all we will define a a function to shift a letter by a given amount
def shift_letter(letter, amount):
    letters = string.ascii_uppercase
    index = letters.index(letter)
    shifted_index = (index - amount) % len(letters)
    return letters[shifted_index]

# Define the keyword and the shifts for each letter
# this key word made more sense from all the keywords . All 15 relative shifts were agreeing, so the keyword is the result of some shift appl
keyword = 'CRYPTO'
shifts = [ord(k) - ord('A') for k in keyword]

#Add our Ciphertext to encrypt
ciphertext = 'KCCPKBGUFDPHQTYAVINRRTMVGRKDNBVFDETDGILTXRGUDDKOTFMBPVGEGLTGCKQRACQCWDNAWCRXIZAKFTLEWRPTYCQKYVXCHKFTPONCQQRHJVAJUWETMCMSPKQDYHJ'

# Apply the Vigenere decryption with the keyword and shifts
plaintext = ''
for i, c in enumerate(ciphertext):
    shift = shifts[i % len(shifts)]
    if c in string.ascii_uppercase:
        plaintext += shift_letter(c, shift)
    else:
        plaintext += c

# Lets print  the decrypted plaintext
print(plaintext)
```

```
:HASWINDOWSANDDOORSTHENYOUALLOWTHEOTHERHALFFORMATCHINGTHEPATTERNTHENYOUDOUBLETHEWHOLETHINGAGAINTOGIVEAMARGINOFERRORANDTHENYOUORDERTHEPAPE
```

The first function in this code, shift letter(), takes a letter and a shift amount, and then returns the letter that results from shifting. To do this, it first determines the letter's index in the uppercase alphabet, deducts the shift amount, then multiplies the outcome by 26. (the number of letters in the alphabet). The resulting letter is then obtained using this shifted index. What we did was:

1. We applied the Kakashi test to find HJV occuring 5 times. The intervals are divisible by 2, 3, and 6 and are 18, 138, 54, and 12. Hence, the keyword's length can be 2, 3, or 6.
2. We we compute the Index of Coincidence(Ic):{Assuming the longest key} The fact that the higher numbers were in row 6 and column 6 further supported the conclusion that the keyword length is 6. Column 2 3 4 5 6 7 8

1	0.044	0.064	0.049	0.057	0.079	0.050	0.056
2	0.0524	0.056	0.054	0.057	0.097	0.062	0.062
3		0.057	0.049	0.048	0.066	0.063	0.057
4			0.060	0.049	0.082	0.061	0.063
5				0.057	0.060	0.064	0.062
6					0.090	0.064	0.068

```
7                                          0.061   0.063
8                                                  0.077
```

3. Mutual Index of Coincidence 390(15x26) (MIc). The numbers were K1 - K2 = 11 K1 - K3 = 4 K1 - K4 = 13 K1 - K5 = 9 K1 - K6 = 14 K2 - K3 = 19 K2 - K4 = 2 K2 - K5 = 24 K2 - K6 = 3 K3 - K4 = 9 K3 - K5 = 5 K3 - K6 = 10 K4 - K5 = 22 K4 - K6
4. One result, CRYPTO, out of the 26 potential shifts of APWNRM, made sense.

```
#Affine Cipher
# Ecercise 21. c
import string


def freq_analysis(ciphertext):
    n = len(ciphertext) # we are finding out the length of our ciphertext

    # Create a dictionary to store letter counts
    freq = {} # this will add the frequencies of the letter of our ciphertext
    for char in ciphertext:# this will give the count for number of occurance of that letter
        if char in freq:
            freq[char] += 1
        else:
            freq[char] = 1

    # Now,lets sort the letters by frequency
    sorted_frequency = sorted(freq.items(), key=lambda x: x[1], reverse=True)

    # Then, we create a list of the letters in order of frequency
    sorted_letters = [x[0] for x in sorted_frequency] #we are using the array to check in our frequency that is already sorted

    # Lets print the frequency analysis
    print(f"Total number of characters: {n}")
    print("Ordr: " + '  '.join([str(i) for i in range(26)]))
    print("Ctxt: " + '  '.join(sorted_letters))
    print("Freq: " + '  '.join([str(freq[x]) for x in sorted_letters]))
    ranks = {}
    rank = 1
    for char, count in sorted_freq:
        if count not in ranks:
            ranks[count] = rank
        rank += 1

    # Assign the most common letters
    e = sorted_letters[0] # as in sorted letter 0 and 1 will have the highest occuring letters and we denote e and t to them
    t = sorted_letters[1]

    # lets print them for us to see in output for the most common letters
    print(f"\nMost common letters:")
    print(f"{e} --> e")
    print(f"{t} --> t")

    # Prompt the user to confirm the most common letters or enter new ones. Lets say if they want to have other letters then we can prompt th
    choice = input("\nDo you want to use these common letters? (y/n): ")
    if choice.lower() == 'n':
        # We will allow the user to manually assign the most common letters if they select no
        for i in range(2):
            letter = input(f"Enter the letter to assign to '{string.ascii_lowercase[i]}': ")
            if letter.upper() in sorted_letters:
                if letter.upper() == e:
                    e = string.ascii_uppercase[i]
                elif letter.upper() == t:
                    t = string.ascii_uppercase[i]
            else:
                print(f"Invalid letter '{letter}'.")
                break

    # Here we have assigned the values of a and b ,which is hard coded but is found out by using the equation
    # also remember that the equation modulus should be 1 else that a and b is discarded and we assign t to second highest.
    # this has been calculated manually off the code and we found this value by using the equations
    # 4a+b = 2 and 19a +b =10 which satisfies the condition of remainder.  for finding out a and b  so we have assigned a and b
    # 15a = 8    -> a = 19, b = 4
    a = 19
    b = 4

    # Lets define the decryption function.
    # for that we know dk(y) = 11(y-4) = 11y - 44 and ek(x) = 19x + 4
```

```
    def decrypt_char(char):
        if char.isalpha():
            return chr(((ord(char.upper()) - 65 - b) * pow(a, -1, 26)) % 26 + 65)
        else:
            return char

    # Finally lets apply the decryption functions to the ciphertext

    decrypted = ''.join([decrypt_char(char) for char in ciphertext])

    # Print the results

    print(f"\nDecrypted text: {decrypted}")

# this is the ciphertext that has been provided to us
ciphertext= 'KQEREJEBCPPCJCRKIEACUZBKRVPKRBCIBQCARBJCVFCUPKRIOFKPACUZQEPBKRXPEIIEABDKPBCPFCDCCAFIEABDKPBCPFEQPKAZBKRHAIBKAPCCIBURCCDKDCCJCIDF
freq_analysis(ciphertext)
```

```
Total number of characters: 198
Ordr: 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25
Ctxt: C  B  K  P  I  E  A  R  F  D  J  U  Q  Z  V  O  X  H  N  Y  S
Freq: 32  21  20  20  16  13  13  12  10  9  6  6  4  4  4  2  2  1  1  1  1

Most common letters:
C --> e
B --> t

Do you want to use these common letters? (y/n): y

Decrypted text: OCANADATERREDENOSAIEUXTONFRONTESTCEINTDEFLEURONSGLORIEUXCARTONBRASSAITPORTERLEPEEILSAITPORTERLACROIXTONHISTOIREESTUNEEPC
```

Analysis

Here , first we have calculated the frequency and ranking and order of the cipher text. Then we have the option to check . Also we don't have to change the letter to assign to a and b as it has been manually added by calculating the equation. I wanted to change the value by iterating it but it would have been lengthy and I couldnot do it too. So i just added after manually calculating them.

Here we have assigned the values of a and b ,which is hard coded but is found out by using the equation also remember that the equation modulus should be 1 else that a and b is discarded and we assign t to second highest.

```
 # this has been calculated manually off the code and we found this value by using the equations
 # 4a+b = 2 and 19a +b =10 which satisfies the condition of remainder.  for finding out a and b  so we have assigned a and b
 # 15a = 8    -> a = 19, b = 4



# Ecercise 2.30

# let's get the method first to do the permutation inverse

def per_inv(pi):
    inverse= [0] * 26
    for i, p in enumerate(pi):
        inverse[p]=i
    return inverse

# now lets define another method

def decryptTxt(ciphertext, key,inverse):
    stream = [(key +i-1)%26 for i in range(1, len(ciphertext) +1)]

    # lets define a string
    plaintext =""

    #now lets use loop for ever characters of given ciphertext
    for i,char in enumerate(ciphertext):
        y= ord(char) - 65
        x= (inverse [y-stream[i]]+ 26) %26
        plaintext += chr(x+65)
    return plaintext
```

```
# The given permutation of Z26
pi = [23, 13, 24, 0, 7, 15, 14, 6, 25, 16, 22, 1, 19,18, 5, 11, 17, 2, 21, 12, 20, 4, 10, 9, 3, 8]

# Calling method
inverse= per_inv(pi)
ciphertext= 'WRTCNRLDSAFARWKXFTXCZRNHNYPDTZUUKMPLUSOXNEUDOKLXRMCBKGRCCURR'

# lets call the decrpt text
for K in range(26):
    plaintext= decryptTxt(ciphertext, K, inverse)
    print(f"The key is: {K}, Plaintext is: {plaintext}")
```

```
 The key is: 0, Plaintext is: KJQIXTOKWQSFOXKZFROXOKQWFIFRQKJFVOERWERWIFVTKQQRSFVRWOFICFPW
 The key is: 1, Plaintext is: SFJCZPVSXJUGVZSEGLVZVSJXGCGLJSFGYVHLXHLXCGYPSJJLUGYLXVGCAGWX
 The key is: 2, Plaintext is: UGFAEWYUZFMBYEUHBDYEYUFZBABDFUGBRYODZODZABRWUFFDMBRDZYBAKBXZ
 The key is: 3, Plaintext is: MBGKHXRMEGNTRHMOTIRHRMGETKTIGMBTLRVIEVIEKTLXMGGINTLIERTKSTZE
 The key is: 4, Plaintext is: NTBSOZLNHBQPLONVPCLOLNBHPSPCBNTPDLYCHYCHSPDZNBBCQPDCHLPSUPEH
 The key is: 5, Plaintext is: QPTUVEDQOTJWDVQYWADVDQTOWUWATQPWIDRAORAOUWIEQTTAJWIAODWUMWHO
 The key is: 6, Plaintext is: JWPMYHIJVPFXIYJRXKIYIJPVXMXKPJWXCILKVLKVMXCHJPPKFXCKVIXMNXOV
 The key is: 7, Plaintext is: FXWNROCFYWGZCRFLZSCRCFWYZNZSWFXZACDSYDSYNZAOFWWSGZASYCZNQZVY
 The key is: 8, Plaintext is: GZXQLVAGRXBEALGDEUALAGXREQEUXGZEKAIURIURQEKVGXXUBEKURAEQJEYR
 The key is: 9, Plaintext is: BEZJDYKBLZTHKDBIHMKDKBZLHJHMZBEHSKCMLCMLJHSYBZZMTHSMLKHJFHRL
 The key is: 10, Plaintext is: THEFIRSTDEPOSITCONSISTEDOFONETHOUSANDANDFOURTEENPOUNDSOFGOLD
 The key is: 11, Plaintext is: POHGCLUPIHWVUCPAVQUCUPHIVGVQHPOVMUKQIKQIGVMLPHHQWVMQIUVGBVDI
 The key is: 12, Plaintext is: WVOBADMWCOXYMAWKYJMAMWOCYBYJOWVYNMSJCSJCBYNDWOOJXYNJCMYBTYIC
 The key is: 13, Plaintext is: XYVTKINXAVZRNKXSRFNKNXVARTRFVXYRQNUFAUFATRQIXVVFZRQFANRTPRCA
 The key is: 14, Plaintext is: ZRYPSCQZKYELQSZULGQSQZYKLPLGYZRLJQMGKMGKPLJCZYYGELJGKQLPWLAK
 The key is: 15, Plaintext is: ELRWUAJESRHDJUEMDBJUJERSDWDBRELDFJNBSNBSWDFAERRBHDFBSJDWXDKS
 The key is: 16, Plaintext is: HDLXMKFHULOIFMHNITFMFHLUIXITLHDIGFQTUQTUXIGKHLLTOIGTUFIXZISU
 The key is: 17, Plaintext is: OIDZNSGOMDVCGNOQCPGNGODMCZCPDOICBGJPMJPMZCBSODDPVCBPMGCZECUM
 The key is: 18, Plaintext is: VCIEQUBVNIYABQVJAWBQBVINAEAWIVCATBFWNFWNEATUVIIWYATWNBAEHAMN
 The key is: 19, Plaintext is: YACHJMTYQCRKTJYFKXTJTYCQKHKXCYAKPTGXQGXQHKPMYCCXRKPXQTKHOKNQ
 The key is: 20, Plaintext is: RKAOFNPRJALSPFRGSZPFPRAJSOSZARKSWPBZJBZJOSWNRAAZLSWZJPSOVSQJ
 The key is: 21, Plaintext is: LSKVGQWLFKDUWGLBUEWGWLKFUVUEKLSUXWTEFTEFVUXQLKKEDUXEFWUVYUJF
 The key is: 22, Plaintext is: DUSYBJXDGSIMXBDTMHXBXDSGMYMHSDUMZXPHGPHGYMZJDSSHIMZHGXMYRMFG
 The key is: 23, Plaintext is: IMURTFZIBUCNZTIPNOZTZIUBNRNOUIMNEZWOBWOBRNEFIUUOCNEOBZNRLNGB
 The key is: 24, Plaintext is: CNMLPGECTMAQEPCWQVEPECMTQLQVMCNQHEXVTXVTLQHGCMMVAQHVTEQLDQBT
 The key is: 25, Plaintext is: AQNDWBHAPNKJHWAXJYHWHANPJDJYNAQJOHZYPZYPDJOBANNYKJOYPHJDIJTP
```

Analysis: Initially, using the information provided in the question, we must get the inverse of permutation, "pi," using the modulo formula. The modulo formula approach will be used here, as opposed to the matrix approach used for question 16(b). The given text will then be decrypted using the information produced by the inverse of permutation.

Also, the total is 75 so 75 mod 26 = 23 which is 9 if you see the table so the key is in 10

```
### This is just a demo and I have used the online resource  and I have to learn this . For now I dont know how to use this code and I took r
## This is not my code and its just to check the possible key length
from collections import defaultdict

def kasiski_examination(ciphertext, min_len=3, max_len=5, verbose=True):
    # Perform the Kasiski examination on the given ciphertext.

    # Find all repeated sequences of length min_len to max_len
    repeats = defaultdict(list)
    for length in range(min_len, max_len + 1):
        for i in range(len(ciphertext) - length):
            seq = ciphertext[i:i+length]
            if seq in repeats:
                continue
            # Find all occurrences of the sequence
            indices = [j for j in range(i+1, len(ciphertext)-length+1) if ciphertext[j:j+length] == seq]
            if indices:
                repeats[seq] = indices

    if verbose:
        print(f"Found {len(repeats)} repeated sequences.")

    # Find the distances between all pairs of occurrences of each repeated sequence
    distances = defaultdict(list)
    for seq, indices in repeats.items():
        for i in range(len(indices) - 1):
            for j in range(i+1, len(indices)):
                distance = indices[j] - indices[i]
                distances[seq].append(distance)

    # Find the factors of the distances and count their frequencies
```

```
    factors = defaultdict(int)
    for seq, dists in distances.items():
        for dist in dists:
            for factor in range(2, dist+1):
                if dist % factor == 0:
                    factors[factor] += 1

    # Find the most common factors and their frequencies
    common_factors = sorted(factors.items(), key=lambda x: x[1], reverse=True)[:10]

    if verbose:
        print(f"Found {len(distances)} sequences with at least 2 occurrences.")
        print(f"Found {len(factors)} distinct factors of sequence distances.")
        print("Most common factors and their frequencies:")
        for factor, freq in common_factors:
            print(f"{factor}: {freq}")

    # Estimate the key length based on the most common factors
    key_lengths = []
    for factor, freq in common_factors:
        # Only consider factors that occur more than once
        if freq <= 1:
            break
        for i in range(2, freq):
            if freq % i == 0:
                key_lengths.append(factor*i)
    key_lengths = sorted(set(key_lengths))

    if verbose:
        print(f"Estimated key lengths: {key_lengths}")

    return key_lengths
keylength= kasiski_examination(ciphertext,2,6,True)
ciphertext = 'BNVSNSIHQCEELSSKKYERIFJKXUMBGYKAMQLJTYAVFBKVTDVBPVVRJYYLAOKYMPQSCGDLFSRLLPROYGESEBUUALRWXMMASAZLGLEDFJBZAVVPXWICGJXASCBYEHOSNMU
```

```
Found 92 repeated sequences.
Found 20 sequences with at least 2 occurrences.
Found 63 distinct factors of sequence distances.
Most common factors and their frequencies:
2: 21
3: 20
6: 13
4: 12
9: 8
12: 7
8: 7
5: 5
7: 5
18: 5
Estimated key lengths: [6, 8, 12, 14, 15, 16, 18, 24, 30, 36]
```

```
# I was just surfing and I found out that the code or KEY which was 6 was THEORY. THIS is just addition effor  please ignore this Dr.Geiger

def decrypt_vigenere(ciphertext, key):
    plaintext = ""
    key_len = len(key)
    for i in range(len(ciphertext)):
        c = ciphertext[i]
        k = key[i % key_len]
        p = chr(((ord(c) - ord(k)) % 26) + ord('A'))
        plaintext += p
    return plaintext

ciphertext = 'BNVSNSIHQCEELSSKKYERIFJKXUMBGYKAMQLJTYAVFBKVTDVBPVVRJYYLAOKYMPQSCGDLFSRLLPROYGESEBUUALRWXM'
key = 'THEORY'
plaintext = decrypt_vigenere(ciphertext, key)
print(plaintext)
```

```
IGREWUPAMONGSLOWTALKERSMENINPARTICULARWHODROPPEDWORDSAFEWATATIMELIKEBEANSINAHILLANDWHENIGO
```

FOR 16 .b

To decrypt the given ciphertext "TGEEMNELNNTDROEOAAHDOETCSHAEIRLM" using a permutation cipher with block size m=8 and the given key π, we can follow these steps:

1. Divide the ciphertext into blocks of size 8:

TGEEMNEL NNTDROEO AAHDOETC SHAEIRLM

2. For each block, use the key π to map the positions of the ciphertext characters to the corresponding plaintext characters: TGEE MLNN NDRO TOEA AHDOT EC SHRI ALM
3. The plaintext characters obtained from each block are concatenated to form the final plaintext: GENTLEMEN DONOT READ EACH OTHERS MAIL

✓ 0s completed at 11:22 PM

16 (b)

Sajal.

For any +ve integer m,

Let's assume $P = C = (Z_{26})^m$ and K consist permutations

For a key, i.e. permutation π,

$$e_\pi(x_1 \ldots x_m) = (x_{\pi(1)} \ldots x_{\pi(m)})$$

$$d_\pi(y_1 \ldots y_m) = (y_{\pi^{-1}(1)} \ldots y_{\pi^{-1}(m)})$$

where π^{-1} is inverse

Ex.

x	1	2	3	4	5	6	7	8
$\pi(x)$	4	1	6	2	7	3	8	5

Permutation cipher $m = 8$, encrypted using π

ciphertext = TGIEEMNELNNIDROEDAAHDOETCSHAEIRLM

For ~~cipher~~ given permutation π, we need 2 matrix K_π and its inverse

So,

K_π will be.

$$= \begin{bmatrix} 0&0&0&1&0&0&0&0 \\ 1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0 \end{bmatrix}$$

Safal

16 b Lets get the inverse matrix

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

From the above matrix, inverse will be.

x	1	2	3	4	5	6	7	8
$\pi^{-1}(x)$	2	4	6	1	8	3	5	7

Now, to decrpt. Lest mak block of m = 8

TGEEMNEL NNTDRDED AAHDOETC SHAEIRLM
GEINTLEME NDONOTRE ADEACHOT HERSMAIL

→ Decrpted text - ~~Gpen~~

GIENTLEMEN DONOT READ EACH OTHERS MAIL

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ T & G & E & E & M & N & L & E \\ N & N & T & D & R & O & E & O \\ A & A & H & D & O & E & T & C \\ S & H & A & E & I & R & L & M \end{bmatrix}$$

$$\begin{bmatrix} 2 & 4 & 6 & 1 & 8 & 3 & 5 & 7 \\ G & E & N & T & E & E & M & L \\ N & D & O & N & O & T & R & E \\ A & D & E & A & L & H & O & T \\ H & E & T & S & M & A & I & L \end{bmatrix}$$

decrypted Message.