**EX NO: 01 Vigenere Cipher**

# Python code to implement Vigenere Cipher

# This function generates the key in a cyclic manner until it's length isn't equal to the length of original text

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

# This function returns the encrypted text generated with the help of the key

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

# This function decrypts the encrypted text and returns the original text

def originalText(cipher\_text, key):

orig\_text = []

for i in range(len(cipher\_text)):

x = (ord(cipher\_text[i]) -

ord(key[i]) + 26) % 26

x += ord('A')

orig\_text.append(chr(x))

return("" . join(orig\_text))

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

string = "pmctechcselab7"

keyword = "spark"

key = generateKey(string, keyword)

cipher\_text = cipherText(string,key)

print("Ciphertext :", cipher\_text)

print("Original/Decrypted Text :",

originalText(cipher\_text, key))

output:

Ciphertext : TNOWAGIOVAPBNN

Original/Decrypted Text : VSIZKINIYKRGHQ

EX NO:2 Rail Fence Cipher

# Python3 program to illustrateRail Fence Cipher Encryption and Decryption

# function to encrypt a message

def encryptRailFence(text, key):

# create the matrix to cipher

# plain text key = rows ,

# length(text) = columns

# filling the rail matrix

# to distinguish filled

# spaces from blank ones

rail = [['\n' for i in range(len(text))]

for j in range(key)]

# to find the direction

dir\_down = False

row, col = 0, 0

for i in range(len(text)):

# check the direction of flow

# reverse the direction if we've just

# filled the top or bottom rail

if (row == 0) or (row == key - 1):

dir\_down = not dir\_down

# fill the corresponding alphabet

rail[row][col] = text[i]

col += 1

# find the next row using

# direction flag

if dir\_down:

row += 1

else:

row -= 1

# now we can construct the cipher using the rail matrix

result = []

for i in range(key):

for j in range(len(text)):

if rail[i][j] != '\n':

result.append(rail[i][j])

return("" . join(result))

# This function receives cipher-text and key and returns the original text after decryption

def decryptRailFence(cipher, key):

# create the matrix to cipher # plain text key = rows ,

# length(text) = columns # filling the rail matrix to

# distinguish filled spaces # from blank ones

rail = [['\n' for i in range(len(cipher))]

for j in range(key)]

# to find the direction

dir\_down = None

row, col = 0, 0

# mark the places with '\*'

for i in range(len(cipher)):

if row == 0:

dir\_down = True

if row == key - 1:

dir\_down = False

# place the marker

rail[row][col] = '\*'

col += 1

# find the next row

# using direction flag

if dir\_down:

row += 1

else:

row -= 1

# now we can construct the

# fill the rail matrix

index = 0

for i in range(key):

for j in range(len(cipher)):

if ((rail[i][j] == '\*') and

(index < len(cipher))):

rail[i][j] = cipher[index]

index += 1

# now read the matrix in

# zig-zag manner to construct

# the resultant text

result = []

row, col = 0, 0

for i in range(len(cipher)):

# check the direction of flow

if row == 0:

dir\_down = True

if row == key-1:

dir\_down = False

# place the marker

if (rail[row][col] != '\*'):

result.append(rail[row][col])

col += 1

# find the next row using

# direction flag

if dir\_down:

row += 1

else:

row -= 1

return("".join(result))

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

print(encryptRailFence("play at once", 2))

print(encryptRailFence("Kitkatkitt ", 3))

print(encryptRailFence("defend the east wall", 3))

# Now decryption of the

# same cipher-text

print(decryptRailFence("pa toclya ne", 2))

print(decryptRailFence("Katiktittk", 3))

print(decryptRailFence("dnhaweedtees alf tl", 3))

Output:

pa toclya ne

Katiktittk

dnhaweedtees alf tl

play at once

Kitkatkitt

defend the east wall

# Python3 implementation of Columnar Transposition

import math

key = "HACK"

# Encryption

def encryptMessage(msg):

cipher = ""

# track key indices

k\_indx = 0

msg\_len = float(len(msg))

msg\_lst = list(msg)

key\_lst = sorted(list(key))

# calculate column of the matrix

col = len(key)

# calculate maximum row of the matrix

row = int(math.ceil(msg\_len / col))

# add the padding character '\_' in empty

# the empty cell of the matix

fill\_null = int((row \* col) - msg\_len)

msg\_lst.extend('\_' \* fill\_null)

# create Matrix and insert message and

# padding characters row-wise

matrix = [msg\_lst[i: i + col]

for i in range(0, len(msg\_lst), col)]

# read matrix column-wise using key

for \_ in range(col):

curr\_idx = key.index(key\_lst[k\_indx])

cipher += ''.join([row[curr\_idx]

for row in matrix])

k\_indx += 1

return cipher

# Decryption

def decryptMessage(cipher):

msg = ""

# track key indices

k\_indx = 0

# track msg indices

msg\_indx = 0

msg\_len = float(len(cipher))

msg\_lst = list(cipher)

# calculate column of the matrix

col = len(key)

# calculate maximum row of the matrix

row = int(math.ceil(msg\_len / col))

# convert key into list and sort

# alphabetically so we can access

# each character by its alphabetical position.

key\_lst = sorted(list(key))

# create an empty matrix to

# store deciphered message

dec\_cipher = []

for \_ in range(row):

dec\_cipher += [[None] \* col]

# Arrange the matrix column wise according

# to permutation order by adding into new matrix

for \_ in range(col):

curr\_idx = key.index(key\_lst[k\_indx])

for j in range(row):

dec\_cipher[j][curr\_idx] = msg\_lst[msg\_indx]

msg\_indx += 1

k\_indx += 1

# convert decrypted msg matrix into a string

try:

msg = ''.join(sum(dec\_cipher, []))

except TypeError:

raise TypeError("This program cannot",

"handle repeating words.")

null\_count = msg.count('\_')

if null\_count > 0:

return msg[: -null\_count]

return msg

# Driver Code

msg = "I am Good in behaviour"

cipher = encryptMessage(msg)

print("Encrypted Message: {}".

format(cipher))

print("Decryped Message: {}".

format(decryptMessage(cipher)))

Output:

Encrypted Message: G bvraoiei\_I d aumonho\_

Decryped Message: I am Good in behaviour