Aim: Implement Rail Fence Cipher encryption and decryption algorithm

Program Code :

#Updated Railfence Transposition

#Method for performing K-Rail Fence Encryption on given plaintext

def encryptRailFence(plaintext, key):

# Filter out spaces from the plaintext (spaces are ignored for encipherment)

plaintext = "".join(plaintext.split())

# Creation of the matrix 'rail' filled with placeholders (ph)

rail = [['ph' for i in range(len(plaintext))] # length of plaintext = number of columns

for j in range(key)] # key = number of rows

# Initialized information for sense of direction and values of row/col

direction\_down = False

col = 0

row = 0

for i in range(len(plaintext)):

# Check the direction of flow (is it the first row or last row?)

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

direction\_down = not direction\_down # Invert direction if yes

# Begin filling the cipher matrix

rail[row][col] = plaintext[i]

col += 1

# Change rows based on the flag variable 'direction\_down' logic

if direction\_down:

row += 1

else:

row -= 1

# After the matrix has been filled, we can now extract that information to create the ciphertext

ciphertext = []

for i in range(key): # Rows

for j in range(len(plaintext)): # Columns

if rail[i][j] != 'ph': # If the value is not a placeholder, append it to the ciphertext list

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

return "".join(ciphertext) # Convert the ciphertext list into a singular string

# Decryption

# Method for performing n-Rail Fence Decryption on given ciphertext

def decryptRailFence(ciphertext, key):

# Creation of the matrix 'rail' filled with placeholders (similar to encryption algorithm)

rail = [['\*' for i in range(len(ciphertext))]

for j in range(key)]

## Initialized information for sense of direction and values of row/col

direction\_down = None

col = 0

row = 0

# Create markers on the matrix with 'mkr'

for i in range(len(ciphertext)):

if row == 0: # Highest level row

direction\_down = True

if row == key - 1: # Lowest level row

direction\_down = False

# Begin filling the matrix with markers based on the key and length of text

rail[row][col] = 'mkr'

col += 1

# Change row index based on the flag variable 'direction\_down' logic

if direction\_down:

row += 1

else:

row -= 1

# For loop to begin filling marked spots with characters from the ciphertext

# idx = indexing for ciphertext, i = row index, j = col index

idx = 0

for i in range(key):

for j in range(len(ciphertext)):

if ((rail[i][j] == 'mkr') and

(idx < len(ciphertext))):

rail[i][j] = ciphertext[idx]

idx += 1

# Begin reading the filled rail matrix in a zigzag manner

plaintext = []

col = 0

row = 0

for i in range(len(ciphertext)):

if row == 0: # Highest level row

direction\_down = True

if row == key - 1: # Lowest level row

direction\_down = False

# Begin constructing the plaintext

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

col += 1

# Change rows based on the flag variable 'direction\_down' logic

if direction\_down:

row += 1

else:

row -= 1

return "".join(plaintext) # Convert the plaintext list into a singular string

#----------------------------------------------------------------------------------------------------------------------------------------------

# Main code (Performs computations and Input/Output operations)

# User input to perform encryption or decryption

user\_input = input("\nWould you like to perform encryption or decryption?\nPlease Enter 'e' or 'd': ").lower() # In case user uses capital letters

isValid = True

# Check if the user input is 'e' or 'd'

if user\_input not in ['e', 'd']:

print("Invalid input, please enter 'e' for encryption or 'd' for decryption.")

isValid = False

else:

# Taking user input for the key value

user\_input\_key = input("Enter a key to use >= 2 of type (int): ")

try: # Ensure the user can only use valid whole numbers as the key value

key = int(user\_input\_key)

if key < 2: # Need at least 2 rows or more in order to encrypt

print("Please enter a key a value >= 2.")

isValid = False

except ValueError:

print("Invalid key input, please enter a valid number.")

isValid = False

# This pair of Ciphertext and Plaintext can be used as input for testing purposes if desired:

# ciphertext = "TCECAYSRHSIHRSETILNTEUEIPIRNOC"

# key = 3

# plaintext = "THIS CIPHER IS CERTAINLY NOT SECURE"

if user\_input == 'e' and isValid == True: # If the user chooses to encrypt data and they also have valid input

user\_input\_plaintext = input("\nPlease enter the plaintext to encrypt: \n")

ciphertext = encryptRailFence(user\_input\_plaintext, key) # Call the method for encryption

print("\nPlaintext (Original): " + user\_input\_plaintext)

print("Ciphertext (Generated): " + ciphertext)

print("Key Value Chosen:", key)

elif user\_input == 'd' and isValid == True: # If the user chooses to decrypt data and they also have valid input

user\_input\_ciphertext = input("\nPlease enter the ciphertext to decrypt: \n")

plaintext = decryptRailFence(user\_input\_ciphertext, key) # Call the method for decryption

print("\nCiphertext (Original): " + user\_input\_ciphertext)

print("Plaintext (Generated): " + plaintext)

print("Key Value Chosen:", key)

else: # Will only appear if the user has invalid input data somewhere

print("\nAn error has occured, please try again.\n")

Output:



