

Network Security and Forensics

Lab Session 6

Submitted To:- Submitted By:-

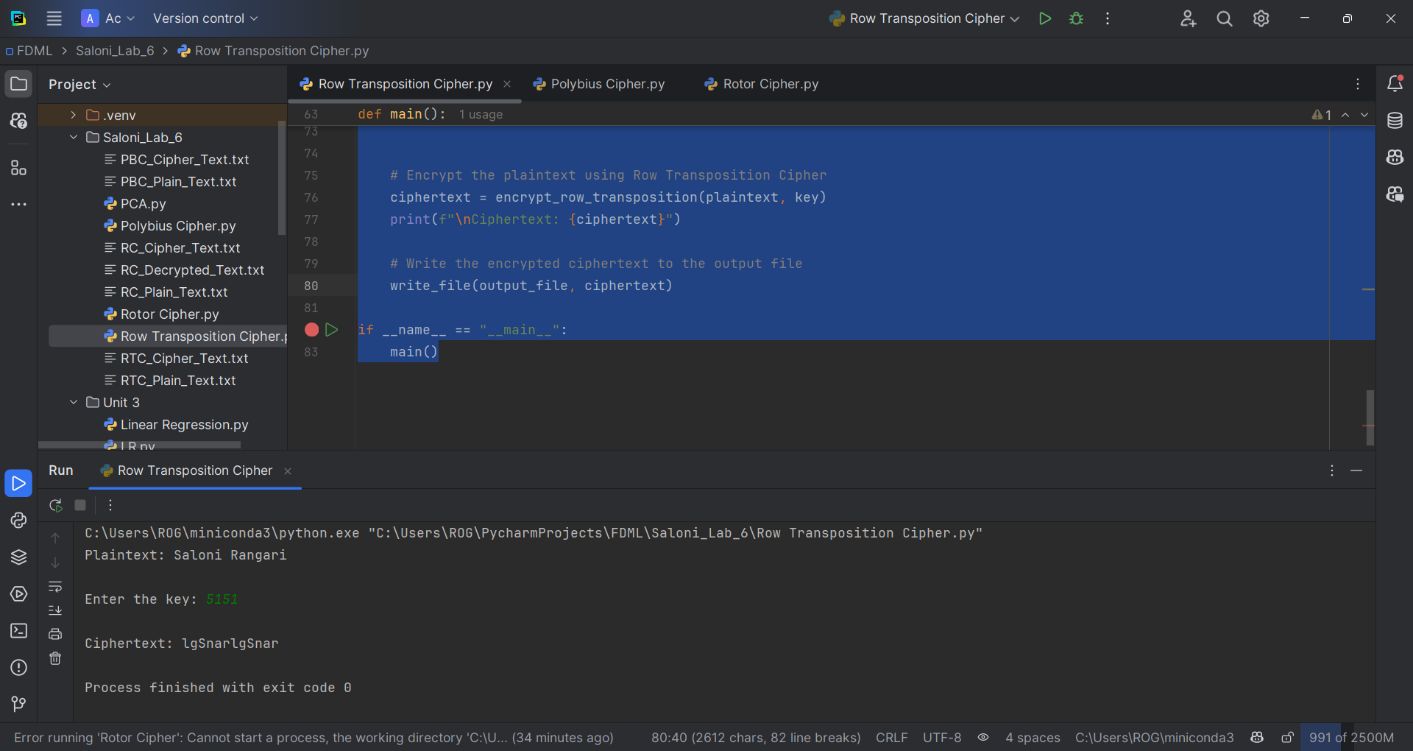
Dr. Lokesh Chauhan Sir Saloni Rangari

M.Tech. AIDS

**Assignment 1: Write a program to demonstrate Row Transposition Cipher. Take Plaintext input form File and Also Ciphertext store in File.**

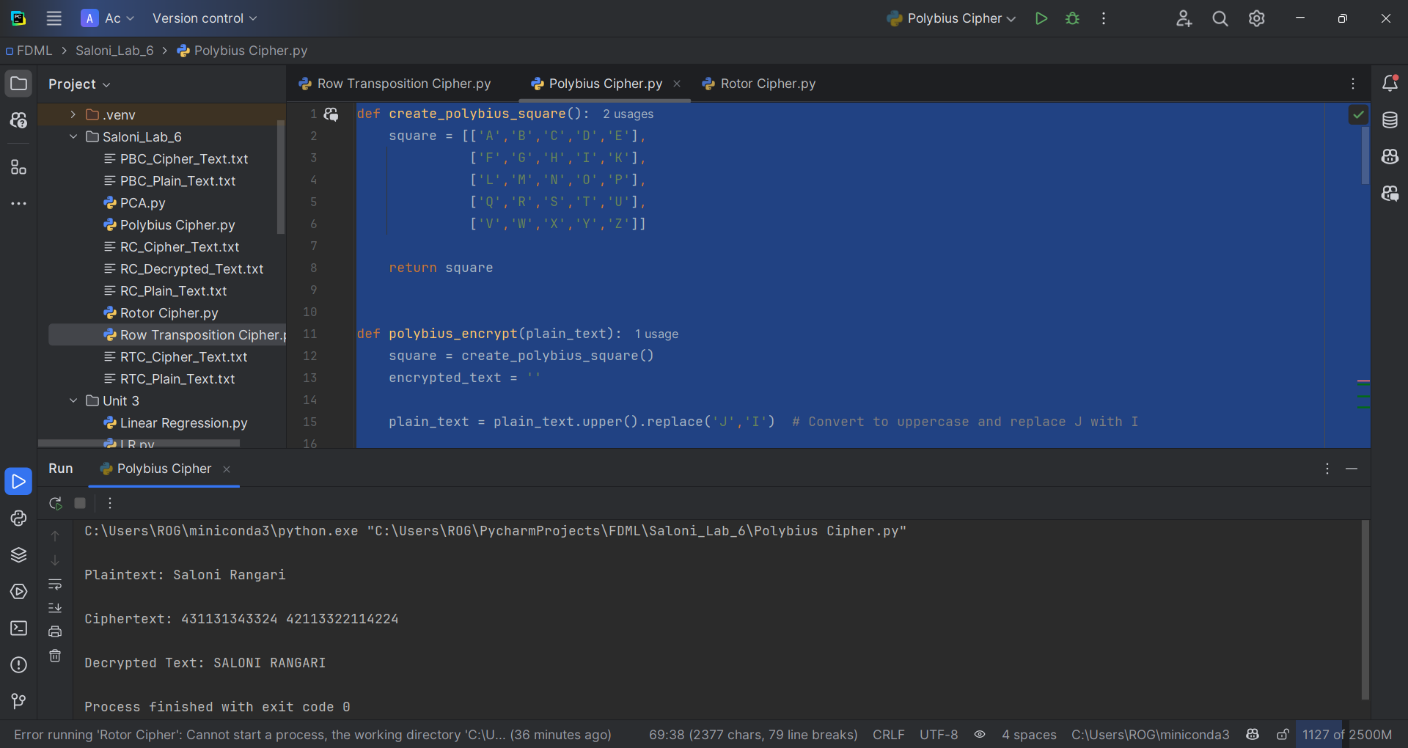
import math  
  
def encrypt\_row\_transposition(plain\_text, key):  
 # Convert the key to a list of numbers for reordering the columns  
 key\_length = len(key)  
 sorted\_key = sorted(list(key))  
 col\_order = [sorted\_key.index(k) for k in key]  
  
 # Calculate the number of rows needed for the grid  
 rows = math.ceil(len(plain\_text) / key\_length)  
  
 # Fill the grid with the plaintext (padding with spaces if necessary)  
 grid = ['' for \_ in range(rows)]  
 index = 0  
 for r in range(rows):  
 for c in range(key\_length):  
 if index < len(plain\_text):  
 grid[r] += plain\_text[index]  
 index += 1  
 else:  
 grid[r] += ' ' # Padding  
  
 # Reorder the columns according to the column order derived from the key  
 encrypted\_text = ''  
 for c in col\_order:  
 for r in range(rows):  
 encrypted\_text += grid[r][c]  
  
 return encrypted\_text.replace(' ', '') # Remove padding spaces in final ciphertext  
  
def decrypt\_row\_transposition(cipher\_text, key):  
 # Calculate the number of rows  
 key\_length = len(key)  
 rows = math.ceil(len(cipher\_text) / key\_length)  
  
 # Sort the key to determine column positions  
 sorted\_key = sorted(list(key))  
 col\_order = [sorted\_key.index(k) for k in key]  
  
 # Create a grid for the decrypted message  
 grid = ['' for \_ in range(rows)]  
  
 # Fill the columns in the order based on the key  
 index = 0  
 for c in col\_order:  
 for r in range(rows):  
 if index < len(cipher\_text):  
 grid[r] += cipher\_text[index]  
 index += 1  
  
 # Read the grid row by row to reconstruct the plaintext  
 decrypted\_text = ''.join(grid)  
 return decrypted\_text.strip() # Strip off any trailing spaces  
  
def read\_file(file\_path):  
 with open(file\_path, 'r') as file:  
 return file.read().strip()  
  
def write\_file(file\_path, content):  
 with open(file\_path, 'w') as file:  
 file.write(content)  
  
def main():  
 # File paths  
 input\_file = 'RTC\_Plain\_Text.txt' # Input file containing plaintext  
 output\_file = 'RTC\_Cipher\_Text.txt' # Output file to store ciphertext  
  
 # Read plaintext from the input file  
 plaintext = read\_file(input\_file)  
 print(f"Plaintext: {plaintext}")  
  
 key = input("\nEnter the key: ")  
  
  
 # Encrypt the plaintext using Row Transposition Cipher  
 ciphertext = encrypt\_row\_transposition(plaintext, key)  
 print(f"\nCiphertext: {ciphertext}")  
  
 # Write the encrypted ciphertext to the output file  
 write\_file(output\_file, ciphertext)  
  
 # Optional: Decrypt to verify  
 decrypted\_text = decrypt\_row\_transposition(ciphertext, key)  
 print(f"Decrypted Text: {decrypted\_text}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Output:**

****

**Assignment 2: Write a program to demonstrate Polybius Cipher  
Take Plaintext input form File and Also Ciphertext store in File.**

def create\_polybius\_square():  
 square = [['A','B','C','D','E'],  
 ['F','G','H','I','K'],  
 ['L','M','N','O','P'],  
 ['Q','R','S','T','U'],  
 ['V','W','X','Y','Z']]  
  
 return square  
  
  
def polybius\_encrypt(plain\_text):  
 square = create\_polybius\_square()  
 encrypted\_text = ''  
  
 plain\_text = plain\_text.upper().replace('J','I') # Convert to uppercase and replace J with I  
  
 for char in plain\_text:  
 if char.isalpha(): # Only encrypt alphabetic characters  
 for row in range(5):  
 if char in square[row]:  
 col = square[row].index(char)  
 encrypted\_text += str(row + 1) + str(col + 1) # Append row+col to the ciphertext  
 break  
 else:  
 encrypted\_text += char # Non-alphabetic characters remain unchanged  
  
 return encrypted\_text  
  
  
def polybius\_decrypt(cipher\_text):  
 square = create\_polybius\_square()  
 decrypted\_text = ''  
 i = 0  
  
 while i < len(cipher\_text):  
 if cipher\_text[i].isdigit() and cipher\_text[i + 1].isdigit(): # Ensure it's a pair of digits  
 row = int(cipher\_text[i]) - 1  
 col = int(cipher\_text[i + 1]) - 1  
 decrypted\_text += square[row][col]  
 i += 2 # Move to the next pair  
 else:  
 decrypted\_text += cipher\_text[i] # Non-numeric characters remain unchanged  
 i += 1  
  
 return decrypted\_text  
  
  
def read\_file(file\_path):  
 with open(file\_path,'r') as file:  
 return file.read().strip()  
  
  
def write\_file(file\_path,content):  
 with open(file\_path,'w') as file:  
 file.write(content)  
  
  
def main():  
 # File paths  
 input\_file = 'PBC\_Plain\_Text.txt' # Input file containing plaintext  
 output\_file = 'PBC\_Cipher\_Text.txt' # Output file to store ciphertext  
  
 # Read plaintext from the input file  
 plaintext = read\_file(input\_file)  
 print(f"\nPlaintext: {plaintext}")  
  
 # Encrypt the plaintext using Polybius Cipher  
 ciphertext = polybius\_encrypt(plaintext)  
 print(f"\nCiphertext: {ciphertext}")  
  
 # Write the encrypted ciphertext to the output file  
 write\_file(output\_file,ciphertext)  
  
 # Optional: Decrypt to verify  
 decrypted\_text = polybius\_decrypt(ciphertext)  
 print(f"\nDecrypted Text: {decrypted\_text}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Output: **

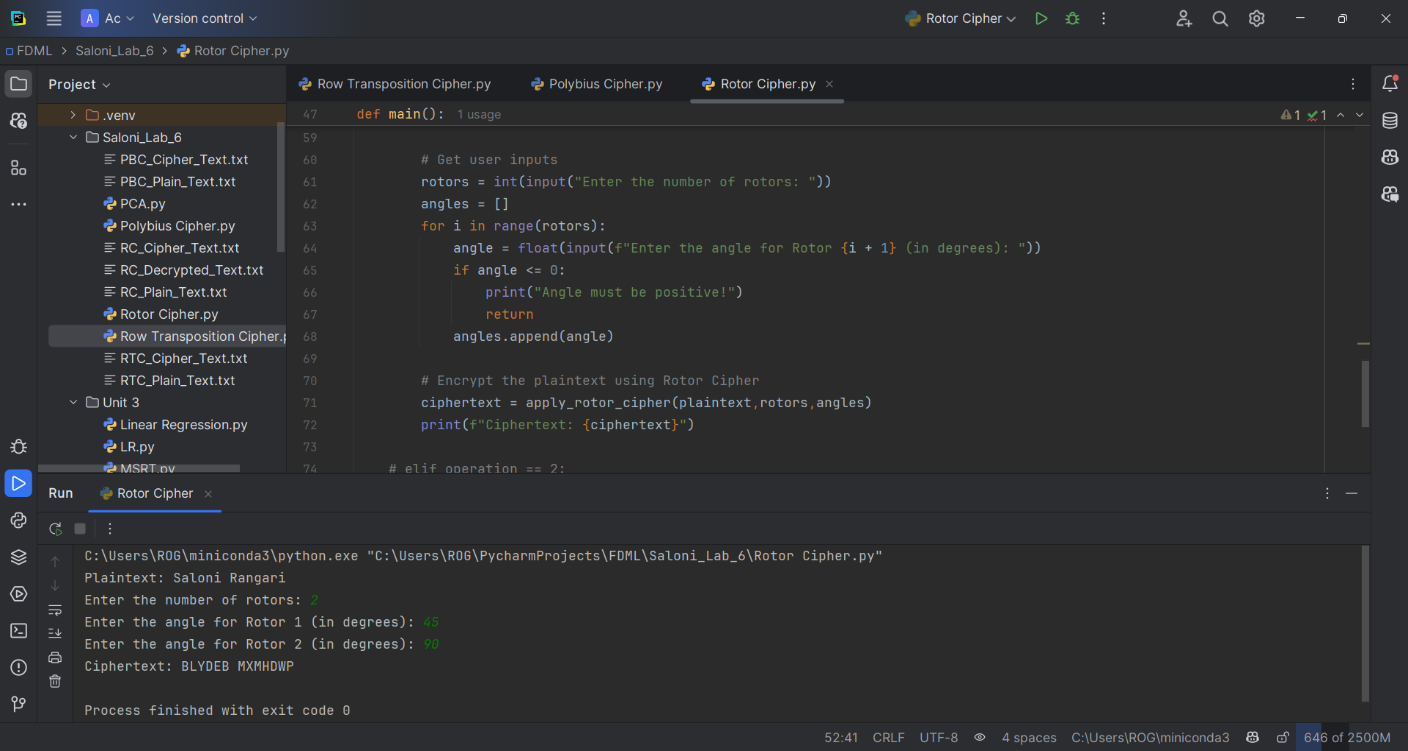
**Assignment 3:  Write a program to demonstrate the Rotor Cipher (as discussed in the classroom).**

**Take the following inputs:**

* **Plaintext**
* **No. of Rotor**
* **Angle of Each Rotor**

import string  
  
  
def get\_shift\_from\_angle(angle):  
 degrees\_per\_shift = 360 / 26  
 return int(angle // degrees\_per\_shift) % 26 # Return the shift value (mod 26)  
  
  
def apply\_rotor\_cipher(plain\_text,rotors,angles):  
 alphabet = string.ascii\_uppercase  
 encrypted\_text = ''  
 plain\_text = plain\_text.upper() # Convert to uppercase for simplicity  
 rotor\_positions = [0] \* rotors # Track the rotor positions  
  
 for i,char in enumerate(plain\_text):  
 if char in alphabet: # Only process alphabetic characters  
 shift = 0  
 # Calculate the total shift based on rotor positions  
 for j in range(rotors):  
 rotor\_shift = get\_shift\_from\_angle(angles[j])  
 shift += (rotor\_positions[j] + rotor\_shift) % 26  
  
 # Find the encrypted character by applying the total shift  
 current\_index = alphabet.index(char)  
 shifted\_index = (current\_index + shift) % 26 # Shift index mod 26  
 encrypted\_text += alphabet[shifted\_index]  
  
 # Rotate each rotor after processing a character  
 for j in range(rotors):  
 rotor\_positions[j] = (rotor\_positions[j] + 1) % 26 # Rotor rotates after each character  
 else:  
 encrypted\_text += char # Non-alphabetic characters remain unchanged  
  
 return encrypted\_text  
  
  
def read\_file(file\_path):  
 with open(file\_path,'r') as file:  
 return file.read().strip()  
  
  
def write\_file(file\_path,content):  
 with open(file\_path,'w') as file:  
 file.write(content)  
  
  
def main():  
 # Input file containing plaintext  
 input\_file = 'RC\_Plain\_Text.txt'  
 # Output file to store ciphertext  
 output\_file = 'RC\_Cipher\_Text.txt'  
  
 # Read plaintext from the input file  
 plaintext = read\_file(input\_file)  
 print(f"Plaintext: {plaintext}")  
  
 # Get user inputs  
 rotors = int(input("Enter the number of rotors: "))  
 angles = []  
 for i in range(rotors):  
 angle = float(input(f"Enter the angle for Rotor {i + 1} (in degrees): "))  
 if angle <= 0:  
 print("Angle must be positive!")  
 return  
 angles.append(angle)  
  
 # Encrypt the plaintext using Rotor Cipher  
 ciphertext = apply\_rotor\_cipher(plaintext,rotors,angles)  
 print(f"Ciphertext: {ciphertext}")  
  
 # Write the encrypted ciphertext to the output file  
 write\_file(output\_file,ciphertext)  
  
 # Optional: Decrypt to verify  
 decrypted\_text = apply\_rotor\_cipher(ciphertext,rotors,angles)  
 print(f"Decrypted Text: {decrypted\_text}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Output:**

****