Network Security and Forensics

Lab Session 6

Submitted To:-

Dr. Lokesh Chauhan Sir

Submitted By:-

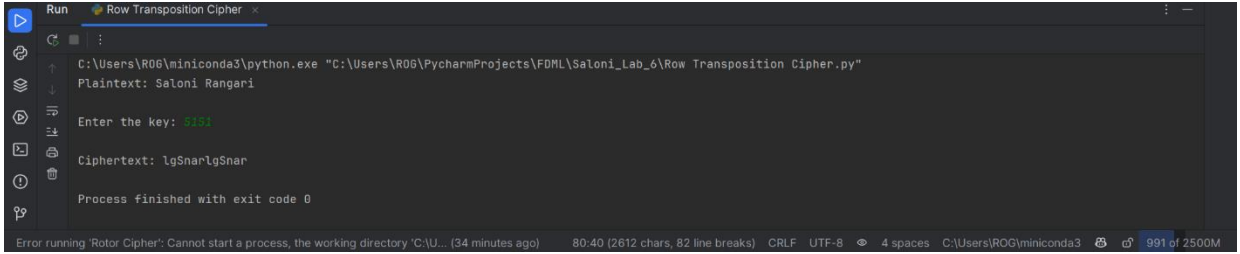
Saloni Rangari

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:


Run
Row Transposition Cipher
C:\Users\ROG\miniconda3\python.exe "C:\Users\ROG\PycharmProjects\FDML\Saloni_Lab_6\Row Transposition Cipher.py"
Plaintext: Saloni Rangari
Enter the key: 5151
Ciphertext: lgSnarlgSnar
Process finished with exit code 0
Error running 'Rotor Cipher': Cannot start a process, the working directory 'C:\U... (34 minutes ago)
80:40 (2612 chars, 82 line breaks)
CRLF
UTF-8
4 spaces
C:\Users\ROG\miniconda3
991 of 2500M

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:

```
Run    Polybius Cipher

C:\Users\ROG\miniconda3\python.exe "C:\Users\ROG\PycharmProjects\FDML\Saloni_Lab_6\Polybius Cipher.py"

Plaintext: Saloni Rangari

Ciphertext: 431131343324 42113322114224

Decrypted Text: SALONI RANGARI

Process finished with exit code 0

Error running 'Rotor Cipher': Cannot start a process, the working directory 'C:\U... (36 minutes ago)   69:38 (2377 chars, 79 line breaks)   CRLF   UTF-8   4 spaces   C:\Users\ROG\miniconda3   1127 of 2500M
```

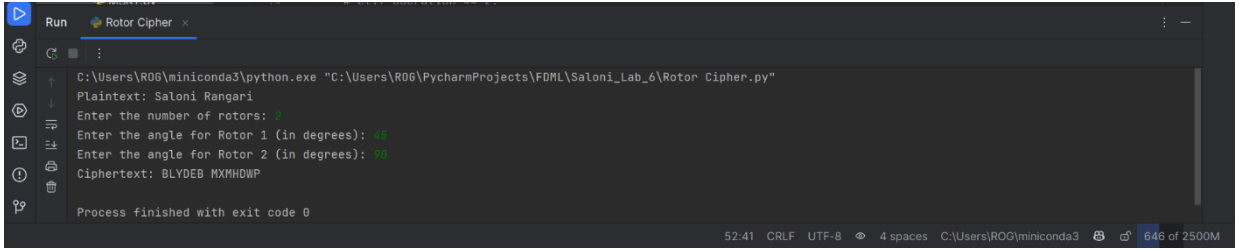
Assignment 3: Write a program to demonstrate the Rotor Cipher (as discussed in the classroom).

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:


Run
Rotor Cipher
C:\Users\ROG\miniconda3\python.exe "C:\Users\ROG\PycharmProjects\FDML\Saloni_Lab_6\Rotor Cipher.py"
Plaintext: Saloni Rangari
Enter the number of rotors: 2
Enter the angle for Rotor 1 (in degrees): 45
Enter the angle for Rotor 2 (in degrees): 90
Ciphertext: BLYDEB MXMHDWP
Process finished with exit code 0
52:41 CRLF UTF-8 4 spaces C:\Users\ROG\miniconda3 646 of 2500M