

Assignment 1

PRN: 21510017

Name: Onkar Anand Yemul

1. Perform encryption, decryption using the following substitution techniques:

a. Ceaser cipher

Ans:

The Caesar Cipher is a simple encryption technique where each letter in a message is shifted by a fixed number of positions in the alphabet. For example, with a shift of 3, "A" becomes "D," "B" becomes "E," and so on. It's one of the oldest known ciphers and is easy to implement but also easy to break.

Python Code:

```
def caesar_encrypt(text, shift):  
    """  
    Encrypt the plain text using Caesar cipher.  
  
    Parameters:  
    text (str): The input text to be encrypted.  
    shift (int): The number of positions to shift each  
character.  
  
    Returns:  
    str: The encrypted text.  
    """  
    encrypted_text = ""  
    for char in text:
```

```
        if char.isalpha():
            shift_amount = shift % 26
            if char.islower():
                new_char = chr((ord(char) - ord('a') +
shift_amount) % 26 + ord('a'))
            else:
                new_char = chr((ord(char) - ord('A') +
shift_amount) % 26 + ord('A'))
            encrypted_text += new_char
        else:
            encrypted_text += char
    return encrypted_text
```

```
def caesar_decrypt(text, shift):
    """
    Decrypt the encrypted text using Caesar cipher.

    Parameters:
    text (str): The input text to be decrypted.
    shift (int): The number of positions to shift each
character back.

    Returns:
    str: The decrypted text.
    """
    return caesar_encrypt(text, -shift)
```

```
def main():
    """
    The main function to run the menu-driven program.
    """
    while True:
        print("\nCaesar Cipher Program")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")
        choice = input("Enter your choice: ")
```

```
        if choice == '1':
            plain_text = input("\nEnter the plain text: ")
            shift = int(input("Enter the shift value: "))
            encrypted_text = caesar_encrypt(plain_text,
shift)
            print(f"\nEncrypted Text: {encrypted_text}")
        elif choice == '2':
            encrypted_text = input("Enter the encrypted
text: ")
            shift = int(input("Enter the shift value: "))
            decrypted_text = caesar_decrypt(encrypted_text,
shift)
            print(f"Decrypted Text: {decrypted_text}")
        elif choice == '3':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

```

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 1
$ python caesar_cipher.py

Caesar Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text: HELLO
Enter the shift value: 3

Encrypted Text: KHOOR

Caesar Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter the encrypted text: KHOOR
Enter the shift value: 3
Decrypted Text: HELLO

```

Advantages:

- **Simplicity:** Easy to understand and implement.
- **Efficiency:** Fast encryption and decryption.

Disadvantages:

- **Weak Security:** Vulnerable to frequency analysis and brute-force attacks (only 25 possible shifts).
- **Predictability:** Does not change much between different texts.

b. Playfair cipher

Ans:

The Playfair Cipher is a digraph substitution cipher that encrypts pairs of letters. It uses a 5x5 matrix of letters created from a keyword. To encrypt, locate each letter pair in the matrix and swap or substitute based on their

positions. It's more secure than simple substitution ciphers because it encodes pairs of letters rather than individual letters.

Python code:

```
def generate_playfair_matrix(key):
    """
    Generate a 5x5 matrix for the Playfair cipher based on
    the provided key.

    Parameters:
    key (str): The key to generate the matrix.

    Returns:
    list: A 5x5 matrix for the Playfair cipher.
    """
    key = key.upper().replace("J", "I")
    matrix = []
    used = set()

    for char in key:
        if char not in used and char.isalpha():
            used.add(char)
            matrix.append(char)

    for char in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        if char not in used:
            used.add(char)
            matrix.append(char)

    return [matrix[i:i + 5] for i in range(0, 25, 5)]

def find_position(matrix, char):
    """
    Find the row and column of a character in the Playfair
    matrix.

    Parameters:
```

```

matrix (list): The 5x5 matrix for the Playfair cipher.
char (str): The character to find in the matrix.

Returns:
tuple: The row and column of the character in the
matrix.
"""
for row in range(5):
    for col in range(5):
        if matrix[row][col] == char:
            return row, col
return None

def playfair_encrypt(text, key):
    """
    Encrypt the plain text using the Playfair cipher.

    Parameters:
    text (str): The input text to be encrypted.
    key (str): The key for the Playfair cipher.

    Returns:
    str: The encrypted text.
    """
    text = text.upper().replace("J", "I").replace(" ", "")
    if len(text) % 2 != 0:
        text += "X"

    matrix = generate_playfair_matrix(key)
    encrypted_text = ""

    for i in range(0, len(text), 2):
        char1, char2 = text[i], text[i + 1]

        if char1 == char2:
            char2 = 'X'

        row1, col1 = find_position(matrix, char1)

```

```

        row2, col2 = find_position(matrix, char2)

        if row1 == row2:
            encrypted_text += matrix[row1][(col1 + 1) % 5]
            encrypted_text += matrix[row2][(col2 + 1) % 5]
        elif col1 == col2:
            encrypted_text += matrix[(row1 + 1) % 5][col1]
            encrypted_text += matrix[(row2 + 1) % 5][col2]
        else:
            encrypted_text += matrix[row1][col2]
            encrypted_text += matrix[row2][col1]

    return encrypted_text

def playfair_decrypt(text, key):
    """
    Decrypt the encrypted text using the Playfair cipher.

    Parameters:
    text (str): The input text to be decrypted.
    key (str): The key for the Playfair cipher.

    Returns:
    str: The decrypted text.
    """
    text = text.upper().replace("J", "I").replace(" ", "")
    matrix = generate_playfair_matrix(key)
    decrypted_text = ""

    for i in range(0, len(text), 2):
        char1, char2 = text[i], text[i + 1]

        row1, col1 = find_position(matrix, char1)
        row2, col2 = find_position(matrix, char2)

        if row1 == row2:
            decrypted_text += matrix[row1][(col1 - 1) % 5]
            decrypted_text += matrix[row2][(col2 - 1) % 5]

```

```

        elif col1 == col2:
            decrypted_text += matrix[(row1 - 1) % 5][col1]
            decrypted_text += matrix[(row2 - 1) % 5][col2]
        else:
            decrypted_text += matrix[row1][col2]
            decrypted_text += matrix[row2][col1]

    return decrypted_text

def main():
    """
    The main function to run the menu-driven program.
    """
    while True:
        print("\nPlayfair Cipher Program")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            plain_text = input("\nEnter the plain text: ")
            key = input("Enter the key: ")
            encrypted_text = playfair_encrypt(plain_text,
key)
            print(f"\nEncrypted Text: {encrypted_text}")
        elif choice == '2':
            encrypted_text = input("\nEnter the encrypted
text: ")
            key = input("Enter the key: ")
            decrypted_text =
playfair_decrypt(encrypted_text, key)
            print(f"\nDecrypted Text: {decrypted_text}")
        elif choice == '3':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")

```



```
if __name__ == "__main__":  
    main()
```

Output:

```
Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 1  
○ $ python playfair_cipher.py  
  
Playfair Cipher Program  
1. Encrypt  
2. Decrypt  
3. Exit  
Enter your choice: 1  
  
Enter the plain text: MEET ME AT NOON  
Enter the key: MONARCHY  
  
Encrypted Text: CLKLCLRSANNA  
  
Playfair Cipher Program  
1. Encrypt  
2. Decrypt  
3. Exit  
Enter your choice: 2  
  
Enter the encrypted text: CLKLCLRSANNA  
Enter the key: MONARCHY  
  
Decrypted Text: MEETMEATNOON
```

Advantages:

- **Improved Security:** More secure than Caesar Cipher as it encrypts digraphs (pairs of letters).
- **Simplicity:** Slightly more complex but still relatively easy to implement.

Disadvantages:

- **Key Management:** Requires a good keyword and matrix setup.

- **Vulnerability:** Can still be broken with modern techniques like frequency analysis of digraphs.

c. Hill Cipher

Ans:

The Hill Cipher is a polygraphic substitution cipher that uses linear algebra. It encrypts blocks of text (usually 2x2 or 3x3 matrices) by multiplying them with a key matrix. The key matrix must be invertible for decryption. This method allows for more complex encryption compared to simple substitution ciphers.

Python code:

```
import numpy as np

def mod_inverse(matrix, modulus):
    """
    Calculate the modular inverse of a matrix under a given
    modulus.

    Parameters:
    matrix (numpy.ndarray): The matrix to invert.
    modulus (int): The modulus value.

    Returns:
    numpy.ndarray: The modular inverse of the matrix.
    """
    det = int(np.round(np.linalg.det(matrix)))
    det_inv = pow(det, -1, modulus)
    matrix_modulus_inv = (
        det_inv * np.round(det *
np.linalg.inv(matrix)).astype(int) % modulus
    )
    return matrix_modulus_inv
```

```

def hill_encrypt(text, key):
    """
    Encrypt the plain text using the Hill cipher.

    Parameters:
    text (str): The input text to be encrypted.
    key (list of int): The key for the Hill cipher as a flat
list.

    Returns:
    str: The encrypted text.
    """
    size = int(len(key) ** 0.5)
    key_matrix = np.array(key).reshape(size, size)
    modulus = 26
    text_vector = np.array([ord(char) - ord('A') for char in
text])
    text_vector = text_vector.reshape(-1, size).T
    encrypted_vector = (np.dot(key_matrix, text_vector) %
modulus).T
    encrypted_text = ''.join(chr(num + ord('A')) for num in
encrypted_vector.flatten())
    return encrypted_text

def hill_decrypt(text, key):
    """
    Decrypt the encrypted text using the Hill cipher.

    Parameters:
    text (str): The input text to be decrypted.
    key (list of int): The key for the Hill cipher as a flat
list.

    Returns:
    str: The decrypted text.
    """
    size = int(len(key) ** 0.5)
    key_matrix = np.array(key).reshape(size, size)

```

```

    modulus = 26
    key_matrix_inv = mod_inverse(key_matrix, modulus)
    text_vector = np.array([ord(char) - ord('A') for char in
text])
    text_vector = text_vector.reshape(-1, size).T
    decrypted_vector = (np.dot(key_matrix_inv, text_vector)
% modulus).T
    decrypted_text = ''.join(chr(int(num) + ord('A')) for
num in decrypted_vector.flatten())
    return decrypted_text

def main():
    """
    The main function to run the menu-driven program.
    """
    while True:
        print("\nHill Cipher Program")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            plain_text = input("\nEnter the plain text
(length multiple of key matrix size): ").upper().replace("
", "")

            key = input("Enter the key matrix (comma-
separated integers, e.g., '2,4,5,9' for 2x2 matrix): ")
            key_matrix = list(map(int, key.split(',')))
            size = int(len(key_matrix) ** 0.5)
            if len(plain_text) % size != 0:
                print("Error: The length of the plain text
must be a multiple of the key matrix size.")
                continue
            encrypted_text = hill_encrypt(plain_text,
key_matrix)
            print(f"\nEncrypted Text: {encrypted_text}")
        elif choice == '2':

```

```

        encrypted_text = input("\nEnter the encrypted
text: ").upper().replace(" ", "")
        key = input("Enter the key matrix (comma-
separated integers, e.g., '2,4,5,9' for 2x2 matrix): ")
        key_matrix = list(map(int, key.split(',')))
        size = int(len(key_matrix) ** 0.5)
        if len(encrypted_text) % size != 0:
            print("Error: The length of the encrypted
text must be a multiple of the key matrix size.")
            continue
        decrypted_text = hill_decrypt(encrypted_text,
key_matrix)
        print(f"\nDecrypted Text: {decrypted_text}")
    elif choice == '3':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()

```

Output:

```

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 1
$ python hill_cipher.py

Hill Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text (length multiple of key matrix size): WILL MEET TOMORROW
Enter the key matrix (comma-separated integers, e.g., '2,4,5,9' for 2x2 matrix): 6, 1, 3, 2

Encrypted Text: KEZDYSRYHIMPHCI

Hill Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2

Enter the encrypted text: KEZDYSRYHIMPHCI
Enter the key matrix (comma-separated integers, e.g., '2,4,5,9' for 2x2 matrix): 6, 1, 3, 2

Decrypted Text: WILLMEETTOMORROW

```

Advantages:

- **Polyalphabetic:** Uses linear algebra for encryption, making it stronger than monoalphabetic ciphers.
- **Higher Complexity:** More resistant to frequency analysis due to the use of matrices.

Disadvantages:

- **Complexity:** Requires matrix inversion and modular arithmetic, which can be cumbersome.
- **Key Size:** Key matrix must be invertible, and the length of the plaintext must be a multiple of the matrix size.

d. Vigenere cipher

Ans:

The Vigenère Cipher is a method of encrypting text using a keyword. It works by shifting each letter in the plaintext by an amount determined by

the corresponding letter in the keyword. The key repeats itself if it's shorter than the plaintext.

How It Works:

1. **Keyword:** Choose a keyword (e.g., "KEY").
2. **Encryption:**
 - Write the keyword repeatedly above the plaintext.
 - Shift each letter in the plaintext by the position of the corresponding letter in the keyword (A=0, B=1, ..., Z=25).
3. **Decryption:**
 - Use the same keyword to reverse the shifts and recover the plaintext.

Python Code:

```
def vigenere_encrypt(plain_text, key):  
    """  
    Encrypt the plain text using the Vigenere cipher.  
  
    Parameters:  
    plain_text (str): The input text to be encrypted.  
    key (str): The key for the Vigenere cipher.  
  
    Returns:  
    str: The encrypted text.  
    """  
    plain_text = plain_text.upper().replace(" ", "")  
    key = key.upper().replace(" ", "")  
    key_length = len(key)  
    encrypted_text = ""  
  
    for i, char in enumerate(plain_text):  
        if char.isalpha():  
            shift = ord(key[i % key_length]) - ord('A')  
            encrypted_text += chr(ord(char) + shift)
```

```

        encrypted_char = chr((ord(char) - ord('A') +
shift) % 26 + ord('A'))
        encrypted_text += encrypted_char
    else:
        encrypted_text += char

    return encrypted_text

def vigenere_decrypt(cipher_text, key):
    """
    Decrypt the cipher text using the Vigenere cipher.

    Parameters:
    cipher_text (str): The input text to be decrypted.
    key (str): The key for the Vigenere cipher.

    Returns:
    str: The decrypted text.
    """
    cipher_text = cipher_text.upper().replace(" ", "")
    key = key.upper().replace(" ", "")
    key_length = len(key)
    decrypted_text = ""

    for i, char in enumerate(cipher_text):
        if char.isalpha():
            shift = ord(key[i % key_length]) - ord('A')
            decrypted_char = chr((ord(char) - ord('A') -
shift + 26) % 26 + ord('A'))
            decrypted_text += decrypted_char
        else:
            decrypted_text += char

    return decrypted_text

def main():
    """

```



```

The main function to run the menu-driven program.
"""
while True:
    print("\nVigenere Cipher Program")
    print("1. Encrypt")
    print("2. Decrypt")
    print("3. Exit")
    choice = input("Enter your choice: ")

    if choice == '1':
        plain_text = input("\nEnter the plain text: ")
        key = input("Enter the key: ")
        encrypted_text = vigenere_encrypt(plain_text,
key)
        print(f"\nEncrypted Text: {encrypted_text}")
    elif choice == '2':
        encrypted_text = input("\nEnter the encrypted
text: ")
        key = input("Enter the key: ")
        decrypted_text =
vigenere_decrypt(encrypted_text, key)
        print(f"\nDecrypted Text: {decrypted_text}")
    elif choice == '3':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()

```

Output:

```

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 1
$ python vigenere_cipher.py

Vigenere Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text: MEET ME AT MORNING
Enter the key: CODE

Encrypted Text: OSHXOSDXOCURKBJ

Vigenere Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2

Enter the encrypted text: OSHXOSDXOCURKBJ
Enter the key: CODE

Decrypted Text: MEETMEATMORNING

```

Advantages:

- **Polyalphabetic:** Uses a keyword to shift letters, making it more secure than Caesar Cipher.
- **Improved Security:** Harder to crack with frequency analysis if the keyword is long and complex.

Disadvantages:

- **Keyword Management:** Security depends on the keyword length and complexity.
- **Vulnerabilities:** Can be broken with techniques like the Kasiski examination or frequency analysis if the keyword is short.

Assignment 2

PRN: 21510017

Name: Onkar Anand Yemul

1. Perform encryption and decryption using following transposition techniques

a. Rail fence

Ans:

The Rail Fence Cipher is a type of transposition cipher where the plain text is written in a zigzag pattern across multiple "rails" (rows) and then read row by row to create the cipher text. Decryption involves reconstructing the zigzag pattern to retrieve the original message.

Python code:

```
def rail_fence_encrypt(plain_text, key):  
    """  
    Encrypt the plain text using the Rail Fence cipher.  
  
    Parameters:  
    plain_text (str): The input text to be encrypted.  
    key (int): The number of rails (rows) for the Rail Fence cipher.  
  
    Returns:  
    str: The encrypted text.  
    """  
    # Create a list of strings to represent each rail  
    rail = ['' for _ in range(key)]  
    row, direction = 0, 1
```

```

    # Distribute the characters across the rails in a zigzag
pattern
    for char in plain_text:
        rail[row] += char
        row += direction

    # Reverse direction when we reach the top or bottom
rail
    if row == 0 or row == key - 1:
        direction *= -1

    # Concatenate all the rails to get the encrypted text
    return ''.join(rail)

def rail_fence_decrypt(cipher_text, key):
    """
    Decrypt the cipher text using the Rail Fence cipher.

    Parameters:
    cipher_text (str): The input text to be decrypted.
    key (int): The number of rails (rows) for the Rail Fence
cipher.

    Returns:
    str: The decrypted text.
    """
    # Determine the length of each rail in the zigzag
pattern
    pattern = [0] * len(cipher_text)
    row, direction = 0, 1

    for i in range(len(cipher_text)):
        pattern[i] = row
        row += direction

    # Reverse direction when we reach the top or bottom
rail
    if row == 0 or row == key - 1:

```

```

        direction *= -1

    # Reconstruct the rails from the cipher text
    rail_lengths = [pattern.count(i) for i in range(key)]
    rail_chars = ['' for _ in range(key)]
    pos = 0

    for i in range(key):
        rail_chars[i] = cipher_text[pos:pos +
rail_lengths[i]]
        pos += rail_lengths[i]

    # Reconstruct the original message by following the
zigzag pattern
    result = []
    row_pointers = [0] * key
    for i in range(len(cipher_text)):
        result.append(rail_chars[pattern[i]][row_pointers[pattern[i]]])
        row_pointers[pattern[i]] += 1

    return ''.join(result)

def main():
    """
    The main function to run the menu-driven program.
    """
    while True:
        print("\nRail Fence Cipher Program")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            plain_text = input("\nEnter the plain text: ")
            key = int(input("Enter the number of rails: "))

```

```
        encrypted_text = rail_fence_encrypt(plain_text,
key)
        print(f"\nEncrypted Text: {encrypted_text}")
    elif choice == '2':
        cipher_text = input("\nEnter the encrypted text:
").replace(" ", "")
        key = int(input("Enter the number of rails: "))
        decrypted_text = rail_fence_decrypt(cipher_text,
key)
        print(f"\nDecrypted Text: {decrypted_text}")
    elif choice == '3':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

```

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 2
$ python rail_fence.py

Rail Fence Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text: HELLO FROM THE OTHER SIDE
Enter the number of rails: 3

Encrypted Text: HOMOREELFOTETESDLRHHI

Rail Fence Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2

Enter the encrypted text: HOMOREELFOTETESDLRHHI
Enter the number of rails: 3

Decrypted Text: HELLOFROMTHEOTHERSIDE

```

Advantages:

- **Simplicity:** Easy to understand and implement.
- **Low Computation:** Requires minimal computational resources for encryption and decryption.

Disadvantages:

- **Weak Security:** Very easy to break with simple analysis or known-plaintext attacks.
- **Pattern Recognition:** The regular zigzag pattern makes it susceptible to pattern recognition, which can be exploited to decode the message.

b. row and Column Transformation

Ans:

Row and column transformation is a type of transposition cipher where the message is written in a grid (matrix) and the order of rows and columns is changed according to a key.

Row Transposition: Encrypts text by writing it into rows of a grid, then permuting the columns according to a specific key.

Column Transposition: Encrypts text by writing it into columns of a grid, then permuting the rows according to a specific key.

How It Works:

1. **Write** the plaintext into a grid according to the number of rows or columns.
2. **Permute** the rows or columns based on the key.
3. **Read** off the text in the new order to get the ciphertext.

Python code:

```
import math

def create_matrix(text, key_len):
    """
    Create a matrix from the text with the specified number
    of columns (key length).
    """
    rows = math.ceil(len(text) / key_len)
    matrix = [['' for _ in range(key_len)] for _ in range(rows)]
    k = 0

    for i in range(rows):
        for j in range(key_len):
            if k < len(text):
                matrix[i][j] = text[k]
                k += 1
            else:
```



```
        matrix[i][j] = 'X' # Padding with 'X' if
the matrix is not full
```

```
    return matrix
```

```
def row_column_encrypt(plain_text, row_key, col_key):
    """
```

```
    Encrypt the plain text using row and column
transformation.
```

```
    Parameters:
```

```
    plain_text (str): The input text to be encrypted.
```

```
    row_key (list): The key to rearrange rows.
```

```
    col_key (list): The key to rearrange columns.
```

```
    Returns:
```

```
    str: The encrypted text.
```

```
    """
```

```
    plain_text = plain_text.replace(" ", "")
```

```
    key_len = len(col_key)
```

```
    # Create the matrix from the plain text
```

```
    matrix = create_matrix(plain_text, key_len)
```

```
    # Apply the row key
```

```
    row_matrix = [matrix[i] for i in row_key]
```

```
    # Apply the column key
```

```
    encrypted_text = ""
```

```
    for row in row_matrix:
```

```
        encrypted_row = [row[j] for j in col_key]
```

```
        encrypted_text += ''.join(encrypted_row)
```

```
    return encrypted_text
```

```
def row_column_decrypt(cipher_text, row_key, col_key):
    """
```

Decrypt the cipher text using row and column transformation.

Parameters:

cipher_text (str): The input text to be decrypted.

row_key (list): The key to rearrange rows.

col_key (list): The key to rearrange columns.

Returns:

str: The decrypted text.

"""

key_len = len(col_key)

rows = len(cipher_text) // key_len

Create the matrix to store the rearranged cipher text

matrix = [['' for _ in range(key_len)] for _ in range(rows)]

k = 0

Arrange the cipher text in the matrix based on the column key

for i in range(len(row_key)):

for j in col_key:

matrix[row_key[i]][j] = cipher_text[k]

k += 1

Read the decrypted text row by row

decrypted_text = ""

for i in range(rows):

decrypted_text += ''.join(matrix[i])

return decrypted_text

def main():

"""

The main function to run the menu-driven program.

"""

while True:

```

        print("\nRow and Column Transformation Cipher
Program")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            plain_text = input("\nEnter the plain text: ")
            row_key = list(map(int, input("Enter the row key
as a sequence of numbers (e.g., 2 0 1): ").split()))
            col_key = list(map(int, input("Enter the column
key as a sequence of numbers (e.g., 1 0 2): ").split()))
            encrypted_text = row_column_encrypt(plain_text,
row_key, col_key)
            print(f"\nEncrypted Text: {encrypted_text}")
        elif choice == '2':
            cipher_text = input("\nEnter the encrypted text:
")
            row_key = list(map(int, input("Enter the row key
as a sequence of numbers (e.g., 2 0 1): ").split()))
            col_key = list(map(int, input("Enter the column
key as a sequence of numbers (e.g., 1 0 2): ").split()))
            decrypted_text = row_column_decrypt(cipher_text,
row_key, col_key)
            print(f"\nDecrypted Text: {decrypted_text}")
        elif choice == '3':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()

```

Output:

```

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 2
$ python row_column_transformation.py

Row and Column Transformation Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text: HELLO WORLD
Enter the row key as a sequence of numbers (e.g., 2 0 1): 2 0 1
Enter the column key as a sequence of numbers (e.g., 1 0 2): 1 0 2

Encrypted Text: ROLEHLOLW

Row and Column Transformation Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2

Enter the encrypted text: ROLEHLOLW
Enter the row key as a sequence of numbers (e.g., 2 0 1): 2 0 1
Enter the column key as a sequence of numbers (e.g., 1 0 2): 1 0 2

Decrypted Text: HELLOWORL

```

Advantages:

- **Increased Security:** More complex than simple transpositions.
- **Flexibility:** Key-based rearrangement can add security.

Disadvantages:

- **Complexity:** Can be more complex to implement and manage compared to simple ciphers.
- **Pattern Recognition:** Still susceptible to pattern analysis if not combined with other encryption methods.

Assignment 3

PRN: 21510017

Name: Onkar Anand Yemul

1. Implementation of Euclidean and Extended Euclidean Algorithm

Ans:

The Euclidean and Extended Euclidean algorithms are essential for finding the greatest common divisor (GCD) of two integers. The Extended Euclidean algorithm also finds the coefficients of Bézout's identity, which are useful in solving linear Diophantine equations and in modular arithmetic.

Euclidean Algorithm

The Euclidean algorithm finds the GCD of two numbers by repeatedly applying the following rule: $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$ until b becomes zero. The GCD is then the non-zero remainder.

Extended Euclidean Algorithm

The Extended Euclidean algorithm not only computes the GCD of two integers a and b , but also finds integers x and y such that $ax + by = \text{gcd}(a, b)$.

Python Code:

```
def euclidean_algorithm(a, b):  
    """  
        Compute the GCD of a and b using the Euclidean  
        algorithm.  
    """
```

```

Parameters:
a (int): First integer.
b (int): Second integer.

Returns:
int: The GCD of a and b.
"""

while b != 0:
    a, b = b, a % b
return a

def extended_euclidean_algorithm(a, b):
    """
    Compute the GCD of a and b, as well as the coefficients
    x and y
    such that  $ax + by = \text{gcd}(a, b)$  using the Extended
    Euclidean algorithm.

    Parameters:
    a (int): First integer.
    b (int): Second integer.

    Returns:
    tuple: (gcd, x, y) where gcd is the GCD of a and b, and
    x, y are
    the coefficients of Bézout's identity.
    """
    if b == 0:
        return a, 1, 0
    else:
        gcd, x1, y1 = extended_euclidean_algorithm(b, a % b)
        x = y1
        y = x1 - (a // b) * y1
        return gcd, x, y

def main():
    """

```

```

The main function to run the program.
"""
while True:
    print("\nEuclidean and Extended Euclidean
Algorithm")
    print("1. Compute GCD using Euclidean Algorithm")
    print("2. Compute GCD and coefficients using
Extended Euclidean Algorithm")
    print("3. Exit")
    choice = input("Enter your choice: ")

    if choice == '1':
        a = int(input("\nEnter the first integer (a):
"))
        b = int(input("Enter the second integer (b): "))
        gcd = euclidean_algorithm(a, b)
        print(f"\nGCD of {a} and {b} is: {gcd}")
    elif choice == '2':
        a = int(input("\nEnter the first integer (a):
"))
        b = int(input("Enter the second integer (b): "))
        gcd, x, y = extended_euclidean_algorithm(a, b)
        print(f"\nGCD of {a} and {b} is: {gcd}")
        print(f"Coefficients x and y are: x = {x}, y =
{y}")
        print(f"\nBézout's identity: {a}*({x}) +
{b}*({y}) = {gcd}")
    elif choice == '3':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()

```

Output:

```
Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 3
$ python euclidean.py

Euclidean and Extended Euclidean Algorithm
1. Compute GCD using Euclidean Algorithm
2. Compute GCD and coefficients using Extended Euclidean Algorithm
3. Exit
Enter your choice: 1

Enter the first integer (a): 48
Enter the second integer (b): 18

GCD of 48 and 18 is: 6

Euclidean and Extended Euclidean Algorithm
1. Compute GCD using Euclidean Algorithm
2. Compute GCD and coefficients using Extended Euclidean Algorithm
3. Exit
Enter your choice: 2

Enter the first integer (a): 55
Enter the second integer (b): 15

GCD of 55 and 15 is: 5
Coefficients x and y are: x = -1, y = 4

Bézout's identity:  $55*(-1) + 15*(4) = 5$ 
```

This implementation of the Euclidean and Extended Euclidean algorithms is fundamental in cryptography, number theory, and algorithms related to modular arithmetic.

Assignment 4

PRN: 21510017

Name: Onkar Anand Yemul

1. Implementation of Chinese Remainder Theorem (CRT)

Ans:

The Chinese Remainder Theorem (CRT) is a powerful tool in number theory that provides a solution to a system of simultaneous congruences with pairwise coprime moduli. Given a system of congruences, the CRT allows us to find a unique solution modulo the product of the moduli.

Problem Description

Given n congruences: $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$: $x \equiv a_n \pmod{m_n}$

Where the moduli m_1, m_2, \dots, m_n are pairwise coprime, the CRT provides a unique solution modulo $M = m_1 \times m_2 \times \dots \times m_n$.

For each congruence $x \equiv a_i \pmod{m_i}$, it calculates the partial solution using the formula: $x \equiv a_i \times M_i \times \text{inverse}(M_i, m_i) \pmod{M}$ where $M_i = M/m_i$

The final solution is obtained by summing all partial solutions modulo M .

Python code:

```
def extended_euclidean_algorithm(a, b):  
    """  
        Compute the GCD of a and b, as well as the coefficients  
x and y  
        such that  $ax + by = \text{gcd}(a, b)$  using the Extended  
Euclidean algorithm.  
  
Parameters:
```

```

    a (int): First integer.
    b (int): Second integer.

    Returns:
    tuple: (gcd, x, y) where gcd is the GCD of a and b, and
    x, y are
    the coefficients of Bézout's identity.
    """
    if b == 0:
        return a, 1, 0
    else:
        gcd, x1, y1 = extended_euclidean_algorithm(b, a % b)
        x = y1
        y = x1 - (a // b) * y1
        return gcd, x, y

def chinese_remainder_theorem(a, m):
    """
    Solve the system of congruences using the Chinese
    Remainder Theorem.

    Parameters:
    a (list): List of remainders.
    m (list): List of moduli (must be pairwise coprime).

    Returns:
    int: The smallest non-negative solution to the system of
    congruences.
    """
    assert len(a) == len(m), "The number of remainders and
    moduli must be the same"

    # Calculate the product of all moduli
    M = 1
    for mi in m:
        M *= mi

    # Initialize the solution

```

```

x = 0

# Apply the CRT
for ai, mi in zip(a, m):
    Mi = M // mi # M_i = M / m_i
    gcd, inverse, _ = extended_euclidean_algorithm(Mi,
mi)
    if gcd != 1:
        raise ValueError("Moduli are not pairwise
coprime")
    x += ai * inverse * Mi

return x % M

def main():
    """
    The main function to run the program.
    """
    while True:
        print("\nChinese Remainder Theorem (CRT)")
        print("1. Solve System of Congruences")
        print("2. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            n = int(input("\nEnter the number of
congruences: "))
            a = []
            m = []
            for i in range(n):
                ai = int(input(f"\nEnter remainder a[{i+1}]:
"))
                mi = int(input(f"Enter modulus m[{i+1}]: "))
                a.append(ai)
                m.append(mi)

            solution = chinese_remainder_theorem(a, m)

```

```
        print(f"\nThe solution to the system of
congruences is: {solution}")
    elif choice == '2':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

```
Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 4
$ python chinese_remainder_theorem.py

Chinese Remainder Theorem (CRT)
1. Solve System of Congruences
2. Exit
Enter your choice: 1

Enter the number of congruences: 3

Enter remainder a[1]: 2
Enter modulus m[1]: 3

Enter remainder a[2]: 3
Enter modulus m[2]: 5

Enter remainder a[3]: 2
Enter modulus m[3]: 7

The solution to the system of congruences is: 23
```

Assignment 5

PRN: 21510017

Name: Onkar Anand Yemul

1. Apply DES algorithm for practical applications

Ans:

The Data Encryption Standard (DES) is a symmetric-key algorithm for the encryption of digital data. Although DES is now considered insecure for many applications due to its small key size, it is still an important algorithm for understanding the basics of cryptography.

Practical Application of DES Algorithm

To apply the DES algorithm in a practical application, we can use the **pycryptodome** library in Python, which provides an implementation of DES. Below is an example that demonstrates how to use DES to encrypt and decrypt a message.

Python Code:

```
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

def des_encrypt(plain_text, key):
    """
    Encrypt the plain text using DES algorithm.

    Parameters:
    plain_text (str): The text to be encrypted.
    key (bytes): The encryption key (must be 8 bytes long).
```

```

Returns:
bytes: The encrypted cipher text.
"""

cipher = DES.new(key, DES.MODE_ECB)
padded_text = pad(plain_text.encode(), DES.block_size)
encrypted_text = cipher.encrypt(padded_text)
return encrypted_text

def des_decrypt(cipher_text, key):
    """
    Decrypt the cipher text using DES algorithm.

    Parameters:
    cipher_text (bytes): The encrypted text to be decrypted.
    key (bytes): The decryption key (must be 8 bytes long).

    Returns:
    str: The decrypted plain text.
    """
    cipher = DES.new(key, DES.MODE_ECB)
    decrypted_text = unpad(cipher.decrypt(cipher_text),
DES.block_size)
    return decrypted_text.decode()

def main():
    """
    The main function to run the program.
    """
    print("\nDES Encryption and Decryption")

    # Generate a random 8-byte key for DES
    key = get_random_bytes(8)
    print(f"\nGenerated Key (in hexadecimal): {key.hex()}")

    # Input plaintext
    plain_text = input("Enter the plain text to encrypt: ")

```

```

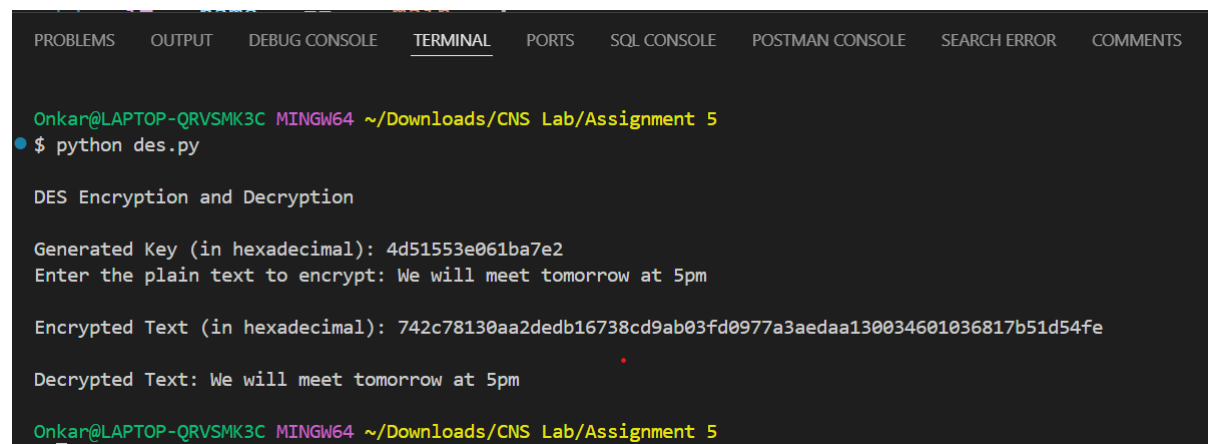
# Encrypt the plaintext
encrypted_text = des_encrypt(plain_text, key)
print(f"\nEncrypted Text (in hexadecimal):
{encrypted_text.hex()}")

# Decrypt the ciphertext
decrypted_text = des_decrypt(encrypted_text, key)
print(f"\nDecrypted Text: {decrypted_text}")

if __name__ == "__main__":
    main()

```

Output:



The screenshot shows a terminal window with the following output:

```

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 5
$ python des.py

DES Encryption and Decryption

Generated Key (in hexadecimal): 4d51553e061ba7e2
Enter the plain text to encrypt: We will meet tomorrow at 5pm

Encrypted Text (in hexadecimal): 742c78130aa2dedb16738cd9ab03fd0977a3aadaa130034601036817b51d54fe

Decrypted Text: We will meet tomorrow at 5pm

```

Practical Applications:

- **File Encryption:** DES can be used to encrypt sensitive files before storing them in insecure locations.
- **Secure Communication:** DES ensures that messages sent over a network are unreadable to unauthorized parties.
- **Password Storage:** Encrypting passwords before storing them in databases (though modern standards recommend stronger algorithms like AES).

While DES itself is outdated and not recommended for secure applications, understanding how it works is crucial for grasping more advanced encryption algorithms like AES.

Assignment 6

PRN: 21510017

Name: Onkar Anand Yemul

1. Apply AES algorithm for practical applications

Ans:

The Advanced Encryption Standard (AES) is a widely used symmetric encryption algorithm that is both fast and secure. It is the standard encryption algorithm used by governments, financial institutions, and many other organizations. Unlike DES, which is now considered insecure, AES is robust and provides a high level of security.

Practical Application of AES Algorithm

We can use the **pycryptodome** library in Python to implement AES encryption and decryption. The AES algorithm can work with key sizes of 128, 192, or 256 bits, and it operates on 128-bit blocks. In this example, we'll use AES with a 256-bit key in Cipher Block Chaining (CBC) mode.

Python Code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

def aes_encrypt(plain_text, key):
    """
    Encrypt the plain text using AES algorithm.

    Parameters:
    plain_text (str): The text to be encrypted.
```

key (bytes): The encryption key (must be 16, 24, or 32 bytes long).

Returns:

bytes: The initialization vector (IV) and the encrypted cipher text.

"""

```
cipher = AES.new(key, AES.MODE_CBC)
iv = cipher.iv # Initialization vector
padded_text = pad(plain_text.encode(), AES.block_size)
encrypted_text = cipher.encrypt(padded_text)
return iv, encrypted_text
```

```
def aes_decrypt(iv, cipher_text, key):
```

"""

Decrypt the cipher text using AES algorithm.

Parameters:

iv (bytes): The initialization vector used during encryption.

cipher_text (bytes): The encrypted text to be decrypted.

key (bytes): The decryption key (must be 16, 24, or 32 bytes long).

Returns:

str: The decrypted plain text.

"""

```
cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted_text = unpad(cipher.decrypt(cipher_text),
AES.block_size)
return decrypted_text.decode()
```

```
def main():
```

"""

The main function to run the program.

"""

```
print("\nAES Encryption and Decryption")
```

```

# Generate a random 32-byte key for AES (256-bit)
key = get_random_bytes(32)
print(f"\nGenerated Key (in hexadecimal): {key.hex()}")

# Input plaintext
plain_text = input("\nEnter the plain text to encrypt:
")

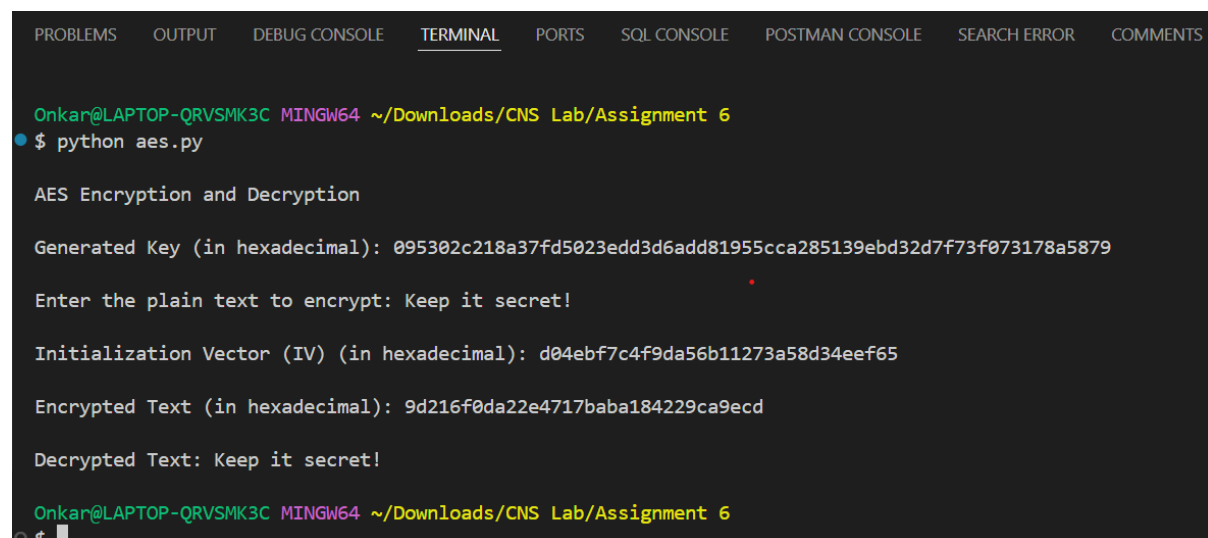
# Encrypt the plaintext
iv, encrypted_text = aes_encrypt(plain_text, key)
print(f"\nInitialization Vector (IV) (in hexadecimal):
{iv.hex()}")
print(f"\nEncrypted Text (in hexadecimal):
{encrypted_text.hex()}")

# Decrypt the ciphertext
decrypted_text = aes_decrypt(iv, encrypted_text, key)
print(f"\nDecrypted Text: {decrypted_text}")

if __name__ == "__main__":
    main()

```

Output:



The screenshot shows a terminal window with the following content:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE  POSTMAN CONSOLE  SEARCH ERROR  COMMENTS

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 6
$ python aes.py

AES Encryption and Decryption

Generated Key (in hexadecimal): 095302c218a37fd5023edd3d6add81955cca285139ebd32d7f73f073178a5879

Enter the plain text to encrypt: Keep it secret!

Initialization Vector (IV) (in hexadecimal): d04ebf7c4f9da56b11273a58d34eef65

Encrypted Text (in hexadecimal): 9d216f0da22e4717baba184229ca9ecd

Decrypted Text: Keep it secret!

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 6
$

```

Practical Applications of AES:

- **File Encryption:** Encrypting sensitive files before storing them on disk.
- **Secure Communication:** Ensuring that data sent over the network remains confidential.
- **Data Protection in Applications:** Encrypting user data, such as passwords, to protect them from unauthorized access.

AES is widely adopted due to its strength and efficiency, and it remains the standard for securing digital data across various industries.

Assignment 7

PRN: 21510017

Name: Onkar Anand Yemul

1. Implementation of RSA Algorithm

Ans:

The RSA algorithm is one of the first public-key cryptosystems and is widely used for secure data transmission. It is an asymmetric cryptographic algorithm, meaning it uses a pair of keys: a public key for encryption and a private key for decryption. It relies on the mathematical properties of prime numbers.

How RSA Works:

1. Key Generation:

- Choose two large prime numbers p and q .
- Compute $n = p * q$.
- Compute the totient $\phi(n) = (p-1) * (q-1)$.
- Choose an encryption key e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. The integer e is the public key exponent.
- Calculate the decryption key d such that $d * e \equiv 1 \pmod{\phi(n)}$. The integer d is the private key exponent.

2. Encryption:

- The public key is (n, e) .
- Given a plaintext message M , the ciphertext C is computed as:
$$C = M^e \pmod{n}$$

3. Decryption:

- The private key is (n, d).
- Given a ciphertext C, the plaintext M is recovered as:

$$M = C^d \bmod n$$

To implement the RSA algorithm using large prime numbers with 2048 bits and converting plaintext into numbers, we'll use the **Crypto** library in Python, which provides the necessary tools to handle such large prime numbers and perform RSA encryption and decryption.

The large primes and the strong key sizes make RSA secure against most attacks when implemented correctly.

Python Code:

```
import random
from sympy import isprime, mod_inverse

def generate_prime_candidate(length):
    """Generate an odd integer randomly."""
    p = random.getrandbits(length)
    # Ensure p is odd
    p |= (1 << length - 1) | 1
    return p

def generate_prime_number(length):
    """Generate a prime number."""
    p = 4
    while not isprime(p):
        p = generate_prime_candidate(length)
    return p

def generate_keypair(keysize):
    """Generate RSA public and private keys."""
    # Generate two large primes p and q
    p = generate_prime_number(keysize)
    q = generate_prime_number(keysize)

    print("\np: ", p)
```

```

print("\nq: ", q)

# Compute n = p * q
n = p * q

# Compute Euler's Totient  $\phi(n) = (p-1)*(q-1)$ 
phi = (p - 1) * (q - 1)

# Choose an integer e such that  $1 < e < \phi(n)$  and  $\gcd(e, \phi(n)) = 1$ 
e = random.randrange(2, phi)
g = gcd(e, phi)
while g != 1:
    e = random.randrange(2, phi)
    g = gcd(e, phi)

# Compute d, the modular inverse of e
d = mod_inverse(e, phi)

# Public key (e, n) and Private key (d, n)
return ((e, n), (d, n))

def gcd(a, b):
    """Compute the greatest common divisor using Euclid's
    algorithm."""
    while b != 0:
        a, b = b, a % b
    return a

def encrypt(public_key, plaintext):
    """Encrypt plaintext using the public key."""
    e, n = public_key
    cipher = [pow(ord(char), e, n) for char in plaintext]
    return cipher

def decrypt(private_key, ciphertext):
    """Decrypt ciphertext using the private key."""
    d, n = private_key
    plain = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(plain)

def main():
    """Run RSA algorithm."""

```

```

print("RSA Encryption/Decryption")

keysize = 2048 # Keysize in bits

# Generate public and private keys
public_key, private_key = generate_keypair(keysize)

print(f"\nPublic key: {public_key}")
print(f"Private key: {private_key}")

# Input plaintext
plaintext = input("\nEnter a message to encrypt: ")

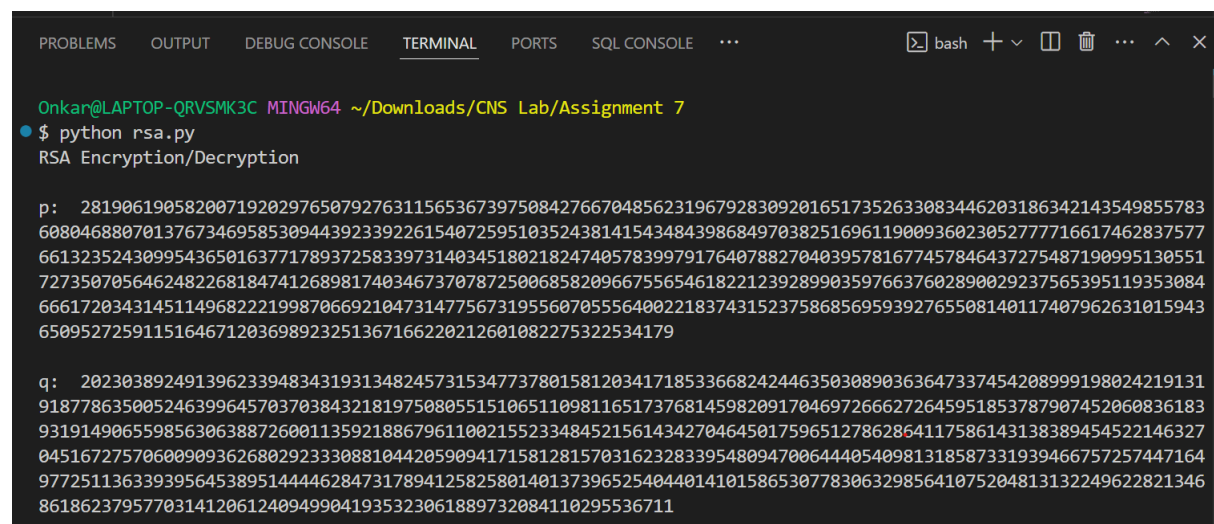
# Encrypt the message
encrypted_msg = encrypt(public_key, plaintext)
print(f"\nEncrypted message: {encrypted_msg}")

# Decrypt the message
decrypted_msg = decrypt(private_key, encrypted_msg)
print(f"\nDecrypted message: {decrypted_msg}")

if __name__ == "__main__":
    main()

```

Output:



```

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 7
$ python rsa.py
RSA Encryption/Decryption

p:  281906190582007192029765079276311565367397508427667048562319679283092016517352633083446203186342143549855783
6080468807013767346958530944392339226154072595103524381415434843986849703825169611900936023052777716617462837577
6613235243099543650163771789372583397314034518021824740578399791764078827040395781677457846437275487190995130551
7273507056462482268184741268981740346737078725006858209667556546182212392899035976637602890029237565395119353084
6661720343145114968222199870669210473147756731955607055564002218374315237586856959392765508140117407962631015943
6509527259115164671203698923251367166220212601082275322534179

q:  202303892491396233948343193134824573153477378015812034171853366824244635030890363647337454208999198024219131
9187786350052463996457037038432181975080551510651109811651737681459820917046972666272645951853787907452060836183
9319149065598563063887260011359218867961100215523348452156143427046450175965127862864117586143138389454522146327
0451672757060090936268029233308810442059094171581281570316232833954809470064440540981318587331939466757257447164
9772511363393956453895144446284731789412582580140137396525404401410158653077830632985641075204813132249622821346
8618623795770314120612409499041935323061889732084110295536711

```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ... bash + - [ ] [ ] ... ^ X

Public key: (401255109074108980465921566640954685015789317040706324808294074793760149164137220756805659517308539
7563577034112088116199484745849904772683471202284494220027855166620075980082304443429479169622060679052470202693
0411787450330773602245241988236305999716273346537348737295969486379632376410752576096797368837081252281928811543
8499809719125842404252765062095777139821878609672956923385374696158715558204716377452238770185059655619694081228
8569422692664375052473888693495808605266702716229478849133929053928420993361232710271417821148142920046097622051
0134751115024305230586833872283163940971126347370493762167683249279350138621051572137038752639163879113599923330
9999697657072046833313723241862776246905259572370283003045826501387901640905965973037275023538843956560231048782
8759233095527964656322074227134589720583964198338472228019412720308801656813417372777422587951157806363092964073
7000204538193785072373882038138315473581186708555176224529929196915106965452203687517287537338945705812840967668
7332279970809149763585237442913932016321933691479760107609057028670610892258054931818693293885854694914264194271
0361221662850020461698721905478118067116265129741979759545727765323602166136205264690785479859636669826727164872
87962702766029, 570307196721614404983350840666356675627763256679820284549479911790970876115008475746256250223776
8146371599321863883031681446691785839946995119840003543052038358925848578190949483904042498591232179970433673786
3589445306990898991311631229107246085574534845388828184939846279982736911759277522580831308890958384873785934556
7943024412606772552334842211021543865528181171093528070864851552788268340687160813485875372279944971937442314267
1650840887039235358518608957535334340650800023834204780615204570195340535432760535656879621533281501204074855757
6008981244138614558131514428825049059946356067898634548024968384367892104527842614286074797008458761962451574170
3969914571785430294272766740765767200855111484313346401331491889184975235214687333063458733121141754274623282320
4348836326088226834620138472537275016568796784555631757676208867466725686541976542881551136845304459722125185642
4825053770689582759790615726013110896003918883091390609161609896702408522605485535599301145413325613447605011217
2061168131488945279238593258247611438875749892420475172049225259552781768690941516700391644950737393961676406343
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ... bash + - [ ] [ ] ... ^ X

4348836326088226834620138472537275016568796784555631757676208867466725686541976542881551136845304459722125185642
4825053770689582759790615726013110896003918883091390609161609896702408522605485535599301145413325613447605011217
2061168131488945279238593258247611438875749892420475172049225259552781768690941516700391644950737393961676406343
3147632394730900198657057615282161393002973042816206874613419199104350758784556088389870521569865445556349211390
67013905446745269)
Private key: (12568402183863569951896883230387076856416531860297403273406475503641345091092906621850466896701745
5237763222302747100768485810321253906129513719927485439658245500815060233076009318429505539356268554915532930490
60159855375128713175530251257160982176841782589050998424651640886604205099304288642717703422422605519653631891348
0419038711175375192089157988909222026942381647912175044143156149711451446394396124094163499604100863082619354679
4050381652933439415839993209100335810729202199952705145831801657214140975820918997608772385794710362251741331564
0415420190340227279456826268515066484514352600144103120407726713356191688125626576937547692379935680417853896567
29680832972042152126293816389457802833135821851959379835852071828871766391773345858160699181123153048860731524750
8273071446397194404794210141362629209309149211874867367262106419101084508944056823929747399575453059691556405723
2911952122481514154313856005959120235648162970773140049770510378174533363942919358060270121213416653283523480329
3765195707877045719050500379139232295766065669589250775087706584783469863130241378310901626224176249734556112108
8873522517005074034966655163209458301838668182077421909028327922614659389175053350444324120072908045229800641110
946515915266009, 57030719672161440498335084066635667562776325667982028454947991179097087611500847574625625022377
6814637159932186388303168144669178583994699511984000354305203835892584857819094948390404249859123217997043367378
6358944530699089899131163122910724608557453484538882818493984627998273691175927752258083130889095838487378593455
6794302441260677255233484221102154386552818117109352807086485155278826834068716081348587537227994497193744231426
7165084088703923535851860895753533434065080002383420478061520457019534053543276053565687962153328150120407485575
7600898124413861455813151442882504905994635606789863454802496838436789210452784261428607479700845876196245157417
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ... bash + - [ ] [ ] ... ^ X
6794302441260677255233484221102154386552818117109352807086485155278826834068716081348587537227994497193744231426
7165084088703923535851860895753533434065080002383420478061520457019534053543276053565687962153328150120407485575
7600898124413861455813151442882504905994635606789863454802496838436789210452784261428607479700845876196245157417
0396991457178543029427276674076576720085511148431334640133149188918497523521468733306345873312114175427462328232
043488363260882268346201384725372750165687967845556317576720886746672568654197654288155113684530445972212518564
2482505377068958275979061572601311089600391888309139060916160989670240852260548553559930114541332561344760501121
7206116813148894527923859325824761143887574989242047517204922525955278176869094151670039164495073739396167640634
3314763239473090019865705761528216139300297304281620687461341919910435075878455608838987052156986544555634921139
067013905446745269)

Enter a message to encrypt: We will meet tomorrow at 5pm at canteen.
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ... bash + - [ ] [ ] ... ^ X

Encrypted message: [51240574717994847921488533151575126509457154796299845527107583540480756402690276029528277377
0833354307176086974022144064598484481349236316528088214300055914053223437485052196289099643104602447423233480064
3371701194809556758988507932489330901631764794030615627926563639891660579784701811596124310510323455685035080779
7799611938239555462850696528381110428562973024594927173461380116035819685915492500597079281408720329666173638677
5916918331271989537471783915042997861503263218688950050491230665963205379247919367521828433283289928212177141468
2232486300300755005035226936209116109937793918905153376388842690888150240430404157102805253429928166829224280722
2186309083599201220743569165262771085717529699970529321860893850917241156451610226841149240430698141777367066809
3372253256335651318886822133515582574514794215894107126462821189415053818693908464164106640330746025557338489549
8807463725536201750993715354377214898815929075078701750205500178593252651094777640806095329007578261761267931110
3772814803304573987162482203022402782603035225647282632582365979240281438777553237597360041960429649807805098635
7872993385128892462917034487731498284941634854762916079316164260234460517751671841579429017277517241086244043085
872883489696078139940, 27635090608112552992037187576122193821105195369672590996782823541108734144696845990626012
860836137784762188122869822462730930914383197330540692115527231375597813421177709391864540524426235685071780799
5886485398994241233398230395264044111077860714989616581667830008486861535467987480922073189040435525321722839971
4767385450669515410905599293882537279185490552092670068685267982301516119429462529318333132389626713898474703098
5152324458283225573020894115508094561792320457723689142650434625106434676447415805633982478691350925888709338391
4509256603909973859995279717155382536306925704599351417810455855828678716644556816244268985677950687921216605337
4299242496072611624947361894130827882528057067123152785234097698096379546037989986332912975770903717632823142331
3560765816500757688740921001040736471281625346617973616385433879610866517280031283010125131455390660162721227309
2334394713328617606302804005819596050895569983314342312737970325239196546613923334214509257510758609134717750158
6517037198718525853392787835657004421012871886761019424828428221068056455519685099170703324692845011594398743062
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ... bash + - [ ] [ ] ... ^ X

2334394713328617606302804005819596050895569983314342312737970325239196546613923334214509257510758609134717750158
6517037198718525853392787835657004421012871886761019424828428221068056455519685099170703324692845011594398743062
5676932057543645102582985716366979453666522127976135239235184165012589333559078431464713963419208722662920153336
033719251332295433791504, 23705036455489225849849620179448513572350923525671099703765122632303400341950610664385
6161458969201215353338733210654974910047709438837478380677023094217893543691452795814046727371125386199112644790
9014653828493382110088305479451368186743454689984284705807087613898626530299508823904408006917562399734423950844
6320933319852761679959677722397576710474516457841163816851046625189198880071350989678808778287424714090050431199
8523881139946825873540590438839893059334666921693398337190929392362636968390756250040297509168387109097461601219
4431795834917362233091699441220279952805313307514070003359793503857605699599264272522953348784442901185825255598
9106322246510672144897452635268972344652223074597614388561110886611818241428709244542032039163404325912179007395
3095283879448122863694126233327082623206941274486540863433290847302705311067388945770485400290942117069901112454
5935215724558544063892849337275973249224914883759644168213759701671674492424855078997046166347895005765252989154
9018078508317402545974233827672863073887836042623236481903318167852397600655145148408930781856695846529122115731
3021611910416891682443080669896949364316204628504324438862277222814535838821775453161102532144304426509691209836
390970794536873163763570784, 55225466416240428439264103353995996065152439023630232771594383579607956153852159964
9327864756075024822266479569334982500107189834339673079544655056813986306561208449356220509488299684270904793743
2118076268659081129970208274991021576165990927312672105879372443181099444720428691869407729844123792702309593419
2496467274299473844652721819439905312233387978982646484211683336223977828803841299976134050813643065377512789454
3303520575331704702321521455711826765642498076264779454328441707711991584359614975748139682734875983555020783455
5220834987946383528774076600639749555019138950366190931383554955562423438939699236984609340612456783997451397246
134805250060848565797121162441480196949346554605708600206035108121134874397302147158276253739626234445628385818
2548374890984592510820178791367073748158705409458606119719392070778017509917946965403343292937388679369272452244
```


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ... bash + - [] [X] ... ^ X

```
5220834987946383528774076600639749555019138950366190931383554955562423438939699236984609340612456783997451397246
1348052500608485657971211624414801969493465546055708600206035108121134874397302147158276253739626234445628385818
2548374890984592510820178791367073748158705409458606119719392070778017509917946965403343292937388679369272452244
4621345912288751823994721951600577454030875068179712594433464364593327847688680418650430412192868384373600654027
1039425763609476169261389105622863813926345102724899097962481202476990259823399900998671391966025010350409801284
9665566808996011735199559565140243392615272848195699729622523376861782117613609596333319040679245622753833739532
012538141835731664941168430932, 10563710283181769993643630292710553153977787763781062362663724928805445142603780
7211081884322063316765857863307103609407822364187389046823527850346730660440999638583927227871073441141985571031
7564412538275693404749810638590837962113998924334229422364190342773099842160725511736469396812880425411582313840
2796462929086222947427401739979900046293235497932335061681291629565475423174659263314151322740571597114788723461
9801461582785541713269691872725110469082820687542359192062875714008498040569171198826395506368593583374762924401
5454472934503657991220981350389156290340257516053472614501327629891635253823413738641477122521335891064555228119
9741057412414553354816526436153659923338605006094641158112976675010796471623150008832421958769718256654377209684
5904222603838172823949248286385553075600080563733879310882992751087049603896778171483777554575458304353439161871
6002887552995698636763230486079385568969953011375100411635380240916417881779944112314381057094910316006548815436
9747954497498132583787914119778893963540835433025425667054109611693608233056894713995598847147898881004893584566
1925766623858603521208761347411699541044691705147823094838045891844049497671563662344666281338687764644751699077
610704794668565231284120004666390, 13640175622460002922004732570640946224061761655339613211815867225859848188713
207398821334204751833270314332643624247734614692489305278718827913217832139430806636295035290838410773391263637
439314720942579737952340082834462789689547399915116454827814867846405600848564551336511608635835204414126367561
6831085709391944036290875247195182148482791763109555505328658286619567638996202700387632915944404646609225789077
4715095880870267481943423966599370040173810357530089545342262767925694409827197091465778928905479291714776971397
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ... bash + - [] [X] ... ^ X

```
6831085709391944036290875247195182148482791763109555505328658286619567638996202700387632915944404646609225789077
4715095880870267481943423966599370040173810357530089545342262767925694409827197091465778928905479291714776971397
0803441979126982175194429572029340841190616232209033489705552343914252190126182034283825373289617702148846658128
4319800444654301706583109493394087259428919844046854135586468251792279890277236507490565420984347372556393314361
2919318283687507136237682615166826485346582662229813975823929013069737824714862679018265946187664701473557193467
6697522902992324748751102054746411563333511926299127622072932023796271848479575244796483635450398394361344924930
6778604413262181460175509137633189350638770619774381621997734091407877015614062981685308398144038289582986515013
1090158677770759991656025258370507152782559753886451587103565869561108321856751187567784194646184885179465302077
433737567357161788429753521027903721, 13640175622460002922004732570640946224061761655339613211815867225859848188
713207398821334204751833270314332643624247734614692489305278718827913217832139430806636295035290838410773391263
6374393147209425797379552340082834462789689547399915116454827814867846405600848564551336511608635835204414126367
5616831085709391944036290875247195182148482791763109555505328658286619567638996202700387632915944404646609225789
0774715095880870267481943423966599370040173810357530089545342262767925694409827197091465778928905479291714776971
3970803441979126982175194429572029340841190616232209033489705552343914252190126182034283825373289617702148846658
1284319800444654301706583109493394087259428919844046854135586468251792279890277236507490565420984347372556393314
3612919318283687507136237682615166826485346582662229813975823929013069737824714862679018265946187664701473557193
4676697522902992324748751102054746411563333511926299127622072932023796271848479575244796483635450398394361344924
9306778604413262181460175509137633189350638770619774381621997734091407877015614062981685308398144038289582986515
0131090158677770759991656025258370507152782559753886451587103565869561108321856751187567784194646184885179465302
077433737567357161788429753521027903721, 23705036455489225849849620179448513572350923525671099703765122632303400
341950610664385616145896920121535338733210654974910047709438837478380677023094217893543691452795814046727371125
3861991126447909014653828493382110088305479451368186743454689984284705807087613898626530299508823904408006917562
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE ... bash + - [ ] [X] ... ^ X
9100480330526012800368592246320319169283876858112831560708758954407909890654051114893353868088380143417866859903
4058533708874771037003139795862055837144163552918029834505968755720570121631767834144528464251239783594207362368
9739678726727181784852025386797980699598838700150479223062044480082299967033903344268816170333771533808283622333
4220171600398779468870869247527604748330305662065811275483486463637325210585904229629085207895144378177796771996
2425681943056619355123950535714886258097111836264993523397107647991586316535845384158316049800058885316786044794
82932475958713645, 421368742534090852407339110525209976020477230572793808561639218983973557774442664299570749841
4253895096616010890308139645537599715812445641868854145252180613355701538703539444922063827343960277052904371846
0491867484886849836788968759537678085921026139389766640479525582574837172267224995272121051023499021928719552991
258633994944376252292413438952476944567796214439352781571895241660888115982829717615997546705540902205499871333
4046621079250030283860627197173640409982723145353969557949405488282277944370641863297926346329624120033218319429
0315493607445576677285567346785502966905855166056181144098717895173066936235399761224454727915898391708560654922
8564045796700886836329236624440385177391838863797023274001517601323612331632938503252762608507021610653549535277
1232837102945771269467755521477296778342707489179342120183278791219797857109577296820847891434938513040540067885
7494218680264520957667919823153698236202754945743846157427829404222247071608819909996281603336356237857331614109
7282470056218721417206344725452407424299201360274032752501957588782381754098710689234861287999842680624781774336
9997148442417947615430392067876952103958885760576235395324401519068218885378341552727660067403478750838369716247
91085756018289911292]

Decrypted message: We will meet tomorrow at 5pm at canteen.

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 7
```

Practical Applications of RSA

- **Secure Communication:** Encrypting emails and messages.
- **Digital Signatures:** Verifying the authenticity of a message or document.
- **Key Exchange:** Securely exchanging keys for symmetric encryption algorithms.

RSA is widely used in various security protocols, including SSL/TLS for secure internet communications.

RSA ensures security through the difficulty of factoring large numbers. It is commonly used for securing sensitive data, digital signatures, and in SSL/TLS protocols.

Assignment 8

PRN: 21510017

Name: Onkar Anand Yemul

1. Implement the Diffie–Hellman Key Exchange algorithm for a given problem

Ans:

The Diffie–Hellman Key Exchange is a cryptographic algorithm that allows two parties to securely share a secret key over a public channel. This shared key can then be used for encrypted communication. The algorithm allows two parties to generate a shared secret key that can be used for subsequent encryption and decryption, even if the exchange itself is observed by an eavesdropper.

How Diffie–Hellman Works:

1. Public Parameters:

- Both parties agree on a large prime number p and a base g (a primitive root modulo p).

2. Key Exchange Process:

- Party A** selects a private key ' a ' and computes $A = g^a \text{ mod } p$, then sends A to Party B.
- Party B** selects a private key ' b ' and computes $B = g^b \text{ mod } p$, then sends B to Party A.

3. Shared Secret:

- Party A** computes the shared secret as $S = B^a \text{ mod } p$.
- Party B** computes the shared secret as $S = A^b \text{ mod } p$.

Since both calculations result in the same value, S becomes the shared secret key, even though an eavesdropper only knows p , g , A , and B .

The Diffie–Hellman algorithm securely establishes a shared secret key without transmitting it directly, making it fundamental for secure communications in protocols like SSL/TLS.

To implement the Diffie–Hellman Key Exchange algorithm for client–server communication across two different machines, we will create two Python programs: one for the client and one for the server. The server will generate its public key and share it with the client, and vice versa. Both will then calculate the shared secret key independently.

Python Code:

Server–side program:

```
import socket
import random

def generate_private_key(p):
    """Generate a private key."""
    return random.randint(2, p-2)

def calculate_public_key(g, private_key, p):
    """Calculate the public key."""
    return pow(g, private_key, p)

def calculate_shared_secret(public_key, private_key, p):
    """Calculate the shared secret."""
    return pow(public_key, private_key, p)
```

```

def start_server(host='localhost', port=5000):
    # p = 23
    # g = 5
    p = 104729 # Shared prime number --> 104729 (which is
the 10000th prime number)
    g = 2 # Shared base --> Primitive Root g: 2

    # Generate server's private and public keys
    private_key = generate_private_key(p)
    public_key = calculate_public_key(g, private_key, p)

    # Create server socket
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server started. Listening on {host}:{port}")

    conn, addr = server_socket.accept()
    print(f"\nConnected by {addr}")

    # Send the server's public key to the client
    conn.sendall(str(public_key).encode())

    # Receive the client's public key
    client_public_key = int(conn.recv(1024).decode())
    print(f"\nReceived Client's Public Key:
{client_public_key}")

    # Calculate the shared secret
    shared_secret =
calculate_shared_secret(client_public_key, private_key, p)
    print(f"\nShared Secret (Server): {shared_secret}")

    conn.close()
    server_socket.close()

if __name__ == "__main__":

```

```
start_server()
```

Client-side program:

```
import socket
import random

def generate_private_key(p):
    """Generate a private key."""
    return random.randint(2, p-2)

def calculate_public_key(g, private_key, p):
    """Calculate the public key."""
    return pow(g, private_key, p)

def calculate_shared_secret(public_key, private_key, p):
    """Calculate the shared secret."""
    return pow(public_key, private_key, p)

def start_client(server_host='localhost', server_port=5000):
    # p = 23
    # g = 5
    p = 104729 # Shared prime number --> 104729 (which is
the 10000th prime number)
    g = 2 # Shared base --> Primitive Root g: 2

    # Generate client's private and public keys
    private_key = generate_private_key(p)
    public_key = calculate_public_key(g, private_key, p)

    # Create client socket
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))
```



```

    # Receive the server's public key
    server_public_key =
int(client_socket.recv(1024).decode())
    print(f"\nReceived Server's Public Key:
{server_public_key}")

    # Send the client's public key to the server
    client_socket.sendall(str(public_key).encode())

    # Calculate the shared secret
    shared_secret =
calculate_shared_secret(server_public_key, private_key, p)
    print(f"\nShared Secret (Client): {shared_secret}")

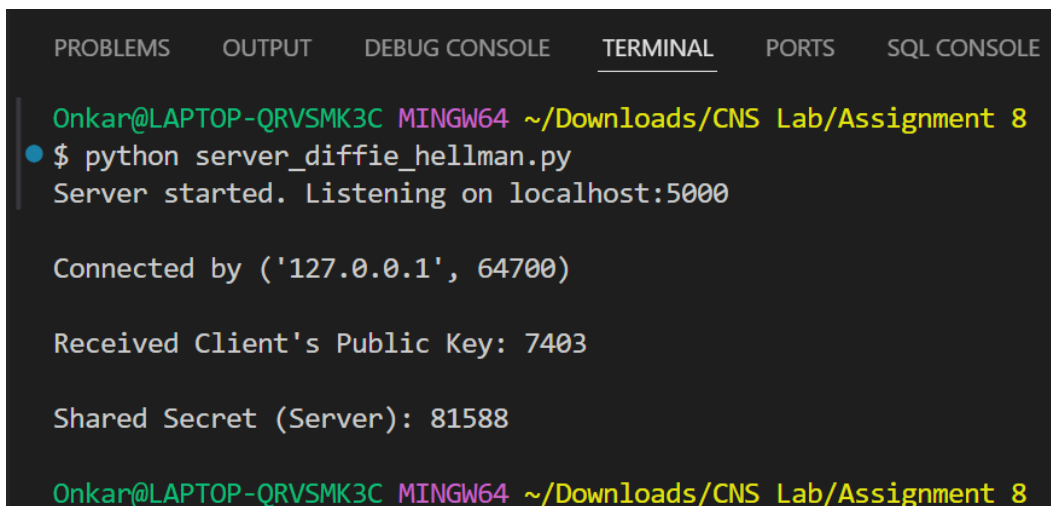
    client_socket.close()

if __name__ == "__main__":
    start_client()

```

Output:

Server output-



The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and SQL CONSOLE. The TERMINAL tab is active, displaying the following output:

```

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8
$ python server_diffie_hellman.py
Server started. Listening on localhost:5000

Connected by ('127.0.0.1', 64700)

Received Client's Public Key: 7403

Shared Secret (Server): 81588

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8

```

Client output-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8
$ python client_diffie_hellman.py
Received Server's Public Key: 23013

Shared Secret (Client): 81588

Onkar@LAPTOP-QRVSMK3C MINGW64 ~/Downloads/CNS Lab/Assignment 8
$
```

Practical Applications of Diffie–Hellman:

- **Secure Communication:** Establishing a shared secret for symmetric encryption over an insecure channel.
- **VPNs:** Secure key exchange for Virtual Private Networks.
- **TLS/SSL:** Part of the key exchange process in securing internet communications.

The Diffie–Hellman algorithm forms the basis of many modern cryptographic protocols and is crucial for secure communication in distributed systems.