Lab No: 1 **Date:** 2081/12/18

Write a program implementing Caesar Cipher.

Source Code:

```
def caesar_cipher(text, key):
    result = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + key) % 26 + base)
        else:
            result += char  # Keep non-alphabet characters unchanged

    return result

def main():
    print("\t\t=================")
    print("\t\t  Caesar Cipher  ")
    print("\t\t=================")

    key = int(input("Enter the key: "))

    while True:
        print("\n1. Encrypt Text \n2. Decrypt Text \n3. Exit")
        choice = input("Enter your choice (1/2/3): ")

        if choice == '1':
            text = input("Enter the text to encrypt: ")
            encrypted = caesar_cipher(text, key)    #Encryption
            print("Encrypted text:", encrypted)
        elif choice == '2':
            text = input("Enter the text to decrypt: ")
            decrypted = caesar_cipher(text, -key)   #Decryption
```

```
        print("Decrypted text:", decrypted)
    elif choice == '3':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice.")
        if __name__ == "__main__":
                main()
```

OUTPUT:

```
● PS D:\Anupam Nepal> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Caesar Cipher.py"
             =================
               Caesar Cipher
             =================
  Enter the key: 4

  1. Encrypt Text
  2. Decrypt Text
  3. Exit
  Enter your choice (1/2/3): 1
  Enter the text to encrypt: Hello
  Encrypted text: Lipps

  1. Encrypt Text
  2. Decrypt Text
  3. Exit
  Enter your choice (1/2/3): 2
  Enter the text to decrypt: Lipps
  Decrypted text: Hello

  1. Encrypt Text
  2. Decrypt Text
  3. Exit
  Enter your choice (1/2/3): 3
  Exiting the program.
○ PS D:\Anupam Nepal> 
```

Lab No: 2 **Date:** 2081/12/18

Write a program implementing Hill Cipher.

Source Code:

```
import numpy as np
# Find modular inverse of a matrix under mod 26
def mod_matrix_inverse(matrix):
    det = int(np.round(np.linalg.det(matrix)))  # Determinant
    det_inv = pow(det % 26, -1, 26)             # Modular inverse
    adj = np.round(np.linalg.inv(matrix) * det).astype(int) % 26  # Adjoint
    return (det_inv * adj) % 26

# Convert text to numbers (A=0 ... Z=25) and numbers to text
def text_to_nums(text):
    return [ord(c.upper()) - ord('A') for c in text if c.isalpha()]

def nums_to_text(nums):
    return ''.join(chr(n + ord('A')) for n in nums)

# Encrypt text using Hill Cipher
def hill_encrypt(text, key):
    nums = text_to_nums(text)
    n = key.shape[0]

    if len(nums) % n != 0:
        nums.append(23)  # 'X' = 23 for padding

    result = []
    for i in range(0, len(nums), n):
        block = nums[i:i+n]
        enc = np.dot(key, block) % 26
        result.extend(enc)
    return nums_to_text(result)

# Decrypt text using Hill Cipher
def hill_decrypt(text, key):
    inv_key = mod_matrix_inverse(key)
    nums = text_to_nums(text)
    n = inv_key.shape[0]
    result = []
    for i in range(0, len(nums), n):
        block = nums[i:i+n]
        dec = np.dot(inv_key, block) % 26
        result.extend(dec)
    return nums_to_text(result)
def main():
    print("\t\t===============")
    print("\t\t  Hill Cipher  ")
    print("\t\t===============")
```

```
    key_matrix = input("Enter the key matrix (space-separated): ")
    key_matrix = np.array([int(num) for num in key_matrix.split()]).reshape(2, 2)

    while True:
        print("\n1. Encrypt Text \n2. Decrypt Text \n3. Exit")
        choice = input("Enter your choice (1/2/3): ")

        if choice == '1':
            plaintext = input("Enter the text to encrypt: ")
            encrypted = hill_encrypt(plaintext, key_matrix)
            print("Encrypted text:", encrypted)

        elif choice == '2':
            ciphertext = input("Enter the text to decrypt: ")
            decrypted = hill_decrypt(ciphertext, key_matrix)
            print("Decrypted text:", decrypted)

        elif choice == '3':
            print("Exiting the program.")
            break

        else:
            print("Invalid choice.")

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

OUTPUT:

```
PS D:\Anupam Nepal> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Hill Cipher.py"
                ===============
                  Hill Cipher
                ===============
Enter the key matrix (space-separated): 3 3 2 5

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 1
Enter the text to encrypt: ROGUE
Encrypted text: PAAIDT

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 2
Enter the text to decrypt: PAAIDT
Decrypted text: ROGUEX

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 3
Exiting the program.
PS D:\Anupam Nepal>
```

Madan Bhandari Memorial College

Lab No: 3 **Date:** 2081/12/18

Write a program implementing Playfair Cipher.

Source Code:

```
# Function to generate the key matrix
def generate_key_matrix(key):
    key = key.upper().replace("J", "I")
    key = "".join(sorted(set(key), key=key.index))  # Remove duplicates, preserve order
    alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"
    key += ''.join(c for c in alphabet if c not in key)

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

# Function to find the position of a character in the matrix
def find_position(matrix, char):
    for i in range(5):
        for j in range(5):
            if matrix[i][j] == char:
                return i, j
    return -1, -1

# Function to prepare the text for encryption/decryption
def prepare_text(text):
    text = text.upper().replace("J", "I")
    prepared = ""
    i = 0
    while i < len(text):
        a = text[i]
        b = text[i+1] if i+1 < len(text) else 'X'
        if a == b:
            prepared += a + 'X'
            i += 1
        else:
```

```
            prepared += a + b
            i += 2
    if len(prepared) % 2 == 1:
        prepared += 'X'
    return prepared

# Function to encrypt the text using Playfair Cipher
def playfair_encrypt(text, matrix):
    text = prepare_text(text)
    result = ""

    for i in range(0, len(text), 2):
        a, b = text[i], text[i+1]
        row1, col1 = find_position(matrix, a)
        row2, col2 = find_position(matrix, b)

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

# Function to decrypt the text using Playfair Cipher
def playfair_decrypt(text, matrix):
    result = ""

    for i in range(0, len(text), 2):
        a, b = text[i], text[i+1]
        row1, col1 = find_position(matrix, a)
        row2, col2 = find_position(matrix, b)

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

```
            result += matrix[(row1 - 1) % 5][col1] +  matrix[(row2 - 1) % 5][col2]
        else:
            result += matrix[row1][col2] + matrix[row2][col1]
    return result

def main():
    print("\t\t===================")
    print("\t\t  Playfair Cipher  ")
    print("\t\t===================")

    key = input("Enter key: ")
    matrix = generate_key_matrix(key)
    while True:
        print("\n1. Encrypt Text \n2. Decrypt Text \n3. Exit")
        choice = input("Enter your choice (1/2/3): ")

        if choice == '1':
            text = input("Enter plaintext: ")
            encrypted = playfair_encrypt(text, matrix)      #Encryption
            print("Encrypted text:", encrypted)

        elif choice == '2':
            text = input("Enter ciphertext: ")
            decrypted = playfair_decrypt(text, matrix)      #Decryption
            print("Decrypted text:", decrypted)

        elif choice == '3':
            print("Exiting the program.")
            break

        else:
            print("Invalid choice!")

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

OUTPUT:

```
PS D:\Anupam Nepal> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Playfair Cipher.py"
                ===================
                  Playfair Cipher
                ===================
Enter key: MONARCHY

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 1
Enter plaintext: HELLO
Encrypted text: CFSUPM

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 2
Enter ciphertext: CFSUPM
Decrypted text: HELXLO

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 3
Exiting the program.
PS D:\Anupam Nepal>
```

Lab No: 4 **Date:** 2081/12/18

Write a program implementing Vigenere Cipher.

Source Code:

```
# Function to repeat the key to match the length of the text
def generate_full_key(text, key):
    key = key.upper()
    key = (key * (len(text) // len(key))) + key[:len(text) % len(key)]
    return key

# Function to encrypt text
def vigenere_encrypt(text, key):
    text = text.upper().replace(" ", "")
    key = generate_full_key(text, key)
    result = ""

    for t, k in zip(text, key):
        if t.isalpha():
            enc_char = chr(((ord(t) - ord('A')) + (ord(k) - ord('A'))) % 26 + ord('A'))
            result += enc_char
        else:
            result += t
    return result

# Function to decrypt text
def vigenere_decrypt(text, key):
    text = text.upper().replace(" ", "")
    key = generate_full_key(text, key)
    result = ""

    for t, k in zip(text, key):
        if t.isalpha():
            dec_char = chr(((ord(t) - ord('A')) - (ord(k) - ord('A')) + 26) % 26 + ord('A'))
```

```
            result += dec_char
        else:
            result += t
    return result

# Main program
def main():
    print("\t\t===================")
    print("\t\t  Vigenère Cipher  ")
    print("\t\t===================")

    key = input("Enter key: ")

    while True:
        print("\n1. Encrypt Text \n2. Decrypt Text \n3. Exit")
        choice = input("Enter your choice (1/2/3): ")

        if choice == '1':
            text = input("Enter plaintext: ")
            encrypted = vigenere_encrypt(text, key)
            print("Encrypted text:", encrypted)

        elif choice == '2':
            text = input("Enter ciphertext: ")
            decrypted = vigenere_decrypt(text, key)
            print("Decrypted text:", decrypted)

        elif choice == '3':
            print("Exiting the program.")
            break

        else:
            print("Invalid choice!")

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

OUTPUT:

```
PS D:\Anupam Nepal> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Vigenere Cipher.py"
                    ====================
                      Vigenère Cipher
                    ====================
Enter key: FAD

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 1
Enter plaintext: CAPSULE
Encrypted text: HASXUOJ

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 2
Enter ciphertext: HASXUOJ
Decrypted text: CAPSULE

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 3
Exiting the program.
PS D:\Anupam Nepal>
```

Lab No: 5 **Date:** 2081/12/18

Write a program implementing Rail Fence Cipher.

Source Code:

```
from tabulate import tabulate
# Function to display the rails in table design
def display_rails_table(matrix):
    table = [[" " if ch == " " else ch for ch in row] for row in matrix]
    headers = [str(i) for i in range(1, len(matrix[0]) + 1)]
    print(tabulate(table, headers=headers, tablefmt="fancy_grid"))

# Function to encrypt using Rail Fence Cipher
def rail_fence_encrypt(text, key):
    text = text.replace(" ", "")
    n = len(text)
    rail = [[' ' for _ in range(n)] for _ in range(key)]

    dir_down = False
    row, col = 0, 0

    for char in text:
        rail[row][col] = char
        if row == 0 or row == key - 1:
            dir_down = not dir_down
        row += 1 if dir_down else -1
        col += 1

    print("\nRail Pattern (Encryption):")
    display_rails_table(rail)
    result = ''
    for r in rail:
        result += ''.join([c for c in r if c != ' '])
    return result
```

```
# Function to decrypt using Rail Fence Cipher
def rail_fence_decrypt(cipher, key):
    n = len(cipher)
    rail = [[' ' for _ in range(n)] for _ in range(key)]

    # Mark the zigzag path
    dir_down = None
    row, col = 0, 0
    for _ in range(n):
        if row == 0:
            dir_down = True
        elif row == key - 1:
            dir_down = False
        rail[row][col] = '*'
        row += 1 if dir_down else -1
        col += 1

    # Fill the path with the cipher characters
    index = 0
    for i in range(key):
        for j in range(n):
            if rail[i][j] == '*' and index < len(cipher):
                rail[i][j] = cipher[index]
                index += 1
    print("\nRail Pattern (Decryption):")
    display_rails_table(rail)
    # Read characters following the zigzag path
    result = ''
    row, col = 0, 0
    dir_down = None
    for _ in range(n):
        if row == 0:
            dir_down = True
        elif row == key - 1:
            dir_down = False
        result += rail[row][col]
```

```
        row += 1 if dir_down else -1
        col += 1
    return result

def main():
    print("\t\t====================")
    print("\t\t  Rail Fence Cipher  ")
    print("\t\t====================")

    key = int(input("Enter key (number of rails): "))

    while True:
        print("\n1. Encrypt Text \n2. Decrypt Text \n3. Exit")
        choice = input("Enter your choice (1/2/3): ")

        if choice == '1':
            text = input("Enter plaintext: ")
            encrypted = rail_fence_encrypt(text, key)
            print("Encrypted text:", encrypted)

        elif choice == '2':
            text = input("Enter ciphertext: ")
            decrypted = rail_fence_decrypt(text, key)
            print("Decrypted text:", decrypted)

        elif choice == '3':
            print("Exiting the program.")
            break

        else:
            print("Invalid choice!")
if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS D:\Anupam Nepal> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Railfence Cipher.py"
                    =====================
                      Rail Fence Cipher
                    =====================
Enter key (number of rails): 3

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 1
Enter plaintext: CSIT PROGRAM IS A HOT CAKE

Rail Pattern (Encryption):
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
C				P				R				S				T				E
	S		T		R		G		A		I		A		O		C		K	
		I				O				M				H				A		

```
Encrypted text: CPRSTESTRGAIAOCKIOMHA

1. Encrypt Text
2. Decrypt Text
3. Exit
Enter your choice (1/2/3): 2
Enter ciphertext: CPRSTESTRGAIAOCKIOMHA

Rail Pattern (Decryption):
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
C				P				R				S				T				E
	S		T		R		G		A		I		A		O		C		K	
		I				O				M				H				A		

```
Decrypted text: CSITPROGRAMISAHOTCAKE
```

Lab No: 6 **Date:** 2081/12/24

Write a program implementing Euclidean Algorithm.

Source Code:

```
def euclidean_algorithm(a, b):
    steps = []
    while b != 0:
        q = a // b
        r = a % b
        steps.append((a, b, q, r))
        a, b = b, r

    # Final step when remainder is 0
    steps.append((a, 0, '', ''))
    return a, steps

def display_table(steps):
    print("\nSteps of Euclidean Algorithm:")
    print("+--------+--------+--------+-------+")
    print("│   A    │   B    │ Q=A//B │ R=A%B │")
    print("+--------+--------+--------+-------+")
    for a, b, q, r in steps:
        print(f"│ {str(a).center(6)} │ {str(b).center(6)} │ {str(q).center(6)} │ {str(r).center(5)} │")
    print("+--------+--------+--------+-------+")

def main():

    a = int(input("Enter first number (A): "))
    b = int(input("Enter second number (B): "))

    # Perform Euclidean Algorithm
    gcd, steps = euclidean_algorithm(a, b)
    display_table(steps)
```

```
    print(f"\nGCD of {a} and {b} is: {gcd}")


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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Euclidean.py"
Enter first number (A): 252
Enter second number (B): 105

Steps of Euclidean Algorithm:
+--------+--------+--------+-------+
|   A    |   B    | Q=A//B | R=A%B |
+--------+--------+--------+-------+
|  252   |  105   |   2    |  42   |
|  105   |   42   |   2    |  21   |
|   42   |   21   |   2    |  0    |
|   21   |   0    |        |       |
+--------+--------+--------+-------+

GCD of 252 and 105 is: 21
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 7 **Date:** 2081/12/24

Write a program implementing Extended Euclidean Algorithm.

Source Code:

```
def extended_euclidean_algorithm(a, n):
    steps = []
    r1, r2 = n, a
    t1, t2 = 0, 1

    while r2 != 0:
        q = r1 // r2
        r = r1 - q * r2
        t = t1 - q * t2
        steps.append((q, r1, r2, r, t1, t2, t))
        r1, r2 = r2, r
        t1, t2 = t2, t

    steps.append((", r1, 0, ", t1, ", "))
    return r1, t1, steps

def display_table(steps):
    print("\nSteps of Extended Euclidean Algorithm:")
    print("+--------+--------+--------+--------+--------+--------+--------+")
    print("|   q    |   r1   |   r2   |   r    |   t1   |   t2   |   t    |")
    print("+--------+--------+--------+--------+--------+--------+--------+")
    for q, r1, r2, r, t1, t2, t in steps:
        print (f"| {str(q).center(6)} | {str(r1).center(6)} | {str(r2).center(6)} | {str(r).center(6)} |
{str(t1).center(6)} | {str(t2).center(6)} | {str(t).center(6)} |")
    print("+--------+--------+--------+--------+--------+--------+--------+")

def main():

    a = int(input("Enter number (A): "))
```

```
    n = int(input("Enter modulo (N): "))

    # Perform the Extended Euclidean Algorithm
    gcd, inverse, steps = extended_euclidean_algorithm(a, n)
    display_table(steps)

    print(f"\nGCD of {a} and {n} is: {gcd}")
    if gcd == 1:
        print(f"Multiplicative Inverse of {a} mod {n} is: {inverse % n}")
    else:
        print(f"{a} has no multiplicative inverse modulo {n} since GCD ≠ 1.")

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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Extended Euclidean.py"
Enter number (A): 7
Enter modulo (N): 26

Steps of Extended Euclidean Algorithm:
+--------+--------+--------+--------+--------+--------+--------+
|   q    |   r1   |   r2   |   r    |   t1   |   t2   |   t    |
+--------+--------+--------+--------+--------+--------+--------+
|   3    |   26   |   7    |   5    |   0    |   1    |   -3   |
|   1    |   7    |   5    |   2    |   1    |   -3   |   4    |
|   2    |   5    |   2    |   1    |   -3   |   4    |  -11   |
|   2    |   2    |   1    |   0    |   4    |  -11   |   26   |
|        |   1    |   0    |        |  -11   |        |        |
+--------+--------+--------+--------+--------+--------+--------+

GCD of 7 and 26 is: 1
Multiplicative Inverse of 7 mod 26 is: 15
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 8 **Date:** 2081/12/24

Write a program implementing Miller Rabin Algorithm.

Source Code:

```
import random
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False
    # Calculate n-1 as 2^k * m
    m = n - 1
    k = 0
    while m % 2 == 0:
        m //= 2
        k += 1
    print(f"Value of k: {k} and m: {m}")
    a = 2
    b = pow(a, m, n)
    print(f"Value of a: {a}")
    print("\n+------------+-------+---------------+---------------+")
    print("| Iterations |    a    | b = a^m mod n | b = b^2 mod n |")
    print("+------------+-------+---------------+---------------+")
    print(f"| {'0'.center(10)} | {str(a).center(5)} | {str(b).center(13)} | {str(b).center(13)} |")
    print("+------------+-------+---------------+---------------+")
    if b == 1 or b == n - 1:
        return True

    # Loop to square b and check conditions
    for j in range(k - 1):
```

```
            b = pow(b, 2, n)
            print(f"| {str(j + 1).center(10)} | {str(a).center(5)} | {str(b).center(13)} | {str(b).center(13)} |")
            print("+------------+-------+---------------+---------------+")
            if b == n - 1:
                return True
        return False

def main():
    print("\t\t====================")
    print("\t\t   Miller-Rabin Test   ")
    print("\t\t====================")
    num = int(input("Enter a number to test for primality: "))
    if is_prime(num):
        print(f"\n{num} is probably prime.")
    else:
        print(f"\n{num} is composite.")
if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Miller-Rabin.py"
                ====================
                  Miller-Rabin Test
                ====================
Enter a number to test for primality: 29
Value of k: 2 and m: 7
Value of a: 2

+------------+-------+---------------+---------------+
| Iterations |   a   | b = a^m mod n | b = b^2 mod n |
+------------+-------+---------------+---------------+
|     0      |   2   |      12       |      12       |
+------------+-------+---------------+---------------+
|     1      |   2   |      28       |      28       |
+------------+-------+---------------+---------------+

29 is probably prime.
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 9 **Date:** 2081/12/24

Write a a program implementing Primitive Root.

Source Code:

```
def power(a, b, mod):
    result = 1
    a %= mod
    while b:
        if b % 2:
            result = (result * a) % mod
        a = (a * a) % mod
        b //= 2
    return result

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def is_primitive_root(r, n):
    for i in range(1, n - 1):
        if power(r, i, n) == 1:
            return False
    return True

def find_primitive_roots(n):
    return [r for r in range(2, n) if gcd(r, n) == 1 and is_primitive_root(r, n)]

def main():

    num = int(input("Enter a number: "))
    roots = find_primitive_roots(num)
```

```
    if roots:
        print(f"\nPrimitive roots of {num} are: {roots}")
    else:
        print(f"\nNo primitive roots found for {num}.")


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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Primitive Root.py"
Enter a number: 7

Primitive roots of 7 are: [3, 5]
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 10 **Date:** 2081/12/27

Write a program implementing Discrete Log.

Source Code:

```
def discrete_log(a, b, p):
    a = a % p
    b = b % p

    print(f"\nFinding i such that {a}^i ≡ {b} (mod {p}):")
    for i in range(p):
        power = pow(a, i, p)
        print(f"{a}^{i} mod {p} = {power}")
        if power == b:
            return i
    return -1

def main():
    print("\t\t=====================")
    print("\t\t   Discrete Logarithm   ")
    print("\t\t=====================")

    a = int(input("Enter value of a: "))
    b = int(input("Enter value of b: "))
    p = int(input("Enter value of p: "))

    result = discrete_log(a, b, p)

    if result is None:
        print(f"\nNo solution found.")
    else:
        print(f"\ndlog_{a},{p} ({b}) = {result}")
```

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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Discrete Logarithm.py"
                  ======================
                    Discrete Logarithm
                  ======================
Enter value of a: 2
Enter value of b: 3
Enter value of p: 5

Finding i such that 2^i ≡ 3 (mod 5):
2^0 mod 5 = 1
2^1 mod 5 = 2
2^2 mod 5 = 4
2^3 mod 5 = 3

dlog_2,5 (3) = 3
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 11 **Date:** 2081/12/27

Write a program implementing Euler's Theorem.

Source Code:

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def euler_totient(n):
    return sum(1 for i in range(1, n) if gcd(i, n) == 1)

def power(a, b, mod):
    result = 1
    a %= mod
    while b:
        if b % 2:
            result = (result * a) % mod
        a = (a * a) % mod
        b //= 2
    return result

def euler_theorem(a, n):
    if gcd(a, n) != 1:
        return None
    phi   = euler_totient(n)
    return power(a, phi, n), phi

def main():

    a = int(input("Enter value of a: "))
    n = int(input("Enter value of n: "))
```

```
    result = euler_theorem(a, n)

    if result is None:
        print(f"\nSince gcd({a}, {n}) ≠ 1, Euler's Theorem is not applicable.")
    else:
        mod_result, phi_n = result
        print(f"\n φ({n}) = {phi_n}")
        print(f"\nAccording to Euler's Theorem: {a}^{phi_n} ≡ {mod_result} mod {n}")
        print("(Verified using modular exponentiation)")

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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Euler's Theorem.py"
Enter value of a: 3
Enter value of n: 10

 φ(10) = 4

According to Euler's Theorem: 3^4 ≡ 1 mod 10
(Verified using modular exponentiation)
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 12 **Date:** 2081/1 2 /27

Write a program implementing Diffie – Helman Key Exchange.

Source Code:

```
def mod_exp(base, exponent, modulus):
    if exponent == 0:
        return 1
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        exponent = exponent >> 1   # Right shift by 1
        base = (base * base) % modulus
    return result

def main():
    print("\t\t===============================")
    print("\t\t   Diffie-Hellman Key Exchange   ")
    print("\t\t===============================")

    p = int(input("\nEnter a prime number (p): "))
    g = int(input(f"Enter a primitive root (g) modulo {p}: "))

    # Alice and Bob's private keys
    a = int(input("Enter Alice's private key (a): "))
    b = int(input("Enter Bob's private key (b): "))

    # Alice and Bob's public keys
    A = mod_exp(g, a, p)
    B = mod_exp(g, b, p)
```

```
    print(f"\nAlice's public key A = g^a mod p = {g}^{a} mod {p} = {A}")
    print(f"Bob's public key B = g^b mod p = {g}^{b} mod {p} = {B}")

    # A and B compute the shared secret key
    s1 = mod_exp(B, a, p)
    s2 = mod_exp(A, b, p)

    print(f"\nAlice computes shared key = B^a mod p = {B}^{a} mod {p} = {s1}")
    print(f"Bob computes shared key = A^b mod p = {A}^{b} mod {p} = {s2}")

    # Check if the shared keys are equal
    if s1 == s2:
        print(f"\nShared secret key: {s1}")
    else:
        print("\n Key exchange failed!")

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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Diffie-Hellman.py"
                ===============================
                  Diffie-Hellman Key Exchange
                ===============================

Enter a prime number (p): 23
Enter a primitive root (g) modulo 23: 5
Enter Alice's private key (a): 6
Enter Bob's private key (b): 15

Alice's public key A = g^a mod p = 5^6 mod 23 = 8
Bob's public key B = g^b mod p = 5^15 mod 23 = 19

Alice computes shared key = B^a mod p = 19^6 mod 23 = 2
Bob computes shared key = A^b mod p = 8^15 mod 23 = 2

Shared secret key: 2
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 13 **Date:** 2081/12/27

Write a program implementing Man in the Middle.

Source Code:

```
def encrypt(message, key):
    return (message + key) % 256
def decrypt(message, key):
    return (message - key + 256) % 256

def main():
    USER1Key = int(input("Enter USER1's secret key: "))
    USER2Key = int(input("Enter USER2's secret key: "))
    eveKey = int(input("Enter Eve's key (attacker): "))
    message = int(input("Enter the message to be sent (0-255): "))
    encryptedByUSER1 = encrypt(message, USER1Key)
    interceptedByEve = encrypt(encryptedByUSER1, eveKey)
    decryptedByUSER2 = decrypt(interceptedByEve, USER2Key)
    print("Message encrypted by USER1:", encryptedByUSER1)
    print("Message intercepted and modified by Eve:", interceptedByEve)
    print("Message decrypted by USER2:", decryptedByUSER2)

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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Man-in-the-Middle.py"
Enter USER1's secret key: 13
Enter USER2's secret key: 7
Enter Eve's key (attacker): 5
Enter the message to be sent (0-255): 219
Message encrypted by USER1: 232
Message intercepted and modified by Eve: 237
Message decrypted by USER2: 230
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 14 **Date:** 2081/12/27

Write a program implementing RSA.

Source Code:

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    if m == 1:
        return 0
    while a > 1:
        q = a // m
        a, m = m, a % m
        x0, x1 = x1 - q * x0, x0
    if x1 < 0:
        x1 += m0
    return x1

def power_mod(base, exp, mod):
    result = 1
    base %= mod
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp //= 2
    return result

def is_prime(n):
```

```
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

def main():
    while True:
        p = int(input("Enter prime p: "))
        if is_prime(p) and p > 1:
            break
        print("p must be a prime number. Try again.")

    while True:
        q = int(input("Enter prime q: "))
        if is_prime(q) and q > 1 and q != p:
            break
        print("q must be a prime number and not equal to p. Try again.")

    n = p * q
    phi = (p - 1) * (q - 1)

    while True:
        e = int(input(f"Enter public exponent e (1 < e < {phi}): "))
        if e <= 1 or e >= phi:
            print(f"e must be between 1 and {phi}. Try again.")
        elif gcd(e, phi) != 1:
            print("e and phi(n) must be coprime. Choose another e.")
        else:
```

```
            break

    d = mod_inverse(e, phi)
    print(f"Private exponent d: {d}")
    print(f"Public key (n, e): ({n}, {e})")
    print(f"Private key (n, d): ({n}, {d})")

    while True:
        message = int(input(f"Enter message to encrypt (integer, 0 <= message < {n}): "))
        if 0 <= message < n:
            break
        print(f"Message must be between 0 and {n - 1}. Try again.")

    encrypted = power_mod(message, e, n)
    print(f"Encrypted message: {encrypted}")
    decrypted = power_mod(encrypted, d, n)
    print(f"Decrypted message: {decrypted}")

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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/RSA.py"
Enter prime p: 13
Enter prime q: 3
Enter public exponent e (1 < e < 24): 17
Private exponent d: 17
Public key (n, e): (39, 17)
Private key (n, d): (39, 17)
Enter message to encrypt (integer, 0 <= message < 39): 20
Encrypted message: 11
Decrypted message: 20
PS D:\Anupam Nepal\Cryptography>
```

Lab No: 15 **Date:** 2081/1 2 /27.

Write a program implementing Elgamal Algorithm.

Source Code:

```
def power_mod(base, exp, mod):
    result = 1
    base %= mod
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp //= 2
    return result

def main():
    p = int(input("Enter a prime p: "))
    g = int(input("Enter a primitive root g: "))
    x = int(input("Enter private key x: "))

    h = power_mod(g, x, p)
    print(f"Public key (p, g, h): ({p}, {g}, {h})")
    print(f"Private key x: {x}")

    m = int(input("Enter message m (integer to encrypt): "))
    k = int(input("Enter random integer k (1 < k < p-1): "))

    a = power_mod(g, k, p)
    b = (m * power_mod(h, k, p)) % p
    print(f"Encrypted message (a, b): ({a}, {b})")

    s = power_mod(a, x, p)
    s_inverse = power_mod(s, p - 2, p)   # Using Fermat's Little Theorem
```

```
    decrypted_message = (b * s_inverse) % p
    print(f"Decrypted message: {decrypted_message}")

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

OUTPUT:

```
PS D:\Anupam Nepal\Cryptography> & "C:/Program Files/Python312/python.exe" "d:/Anupam Nepal/Cryptography/Elagamal Algorithm.py"
Enter a prime p: 13
Enter a primitive root g: 5
Enter private key x: 3
Public key (p, g, h): (13, 5, 8)
Private key x: 3
Enter message m (integer to encrypt): 25
Enter random integer k (1 < k < p-1): 3
Encrypted message (a, b): (8, 8)
Decrypted message: 12
PS D:\Anupam Nepal\Cryptography>
```