Ex. No. 5	**IMPLEMENT ASYMMETRIC CIPHER**
Date: 18- 09 - 2024	

Aim

To write a program to implement the RSA encryption algorithm.

Description

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integer m: (m^e) d = m (mod n) The public key is represented by the integers n and e; and, the private key, by the integer d. m represents the message. RSA involves a public key and a private key.

The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.

1. **Implement RSA encryption and decryption**

   Algorithm

   Step-1: Select two co-prime numbers as p and q.

   Step-2: Compute n as the product of p and q.

   Step-3: Compute (p-1)*(q-1) and store it in z.

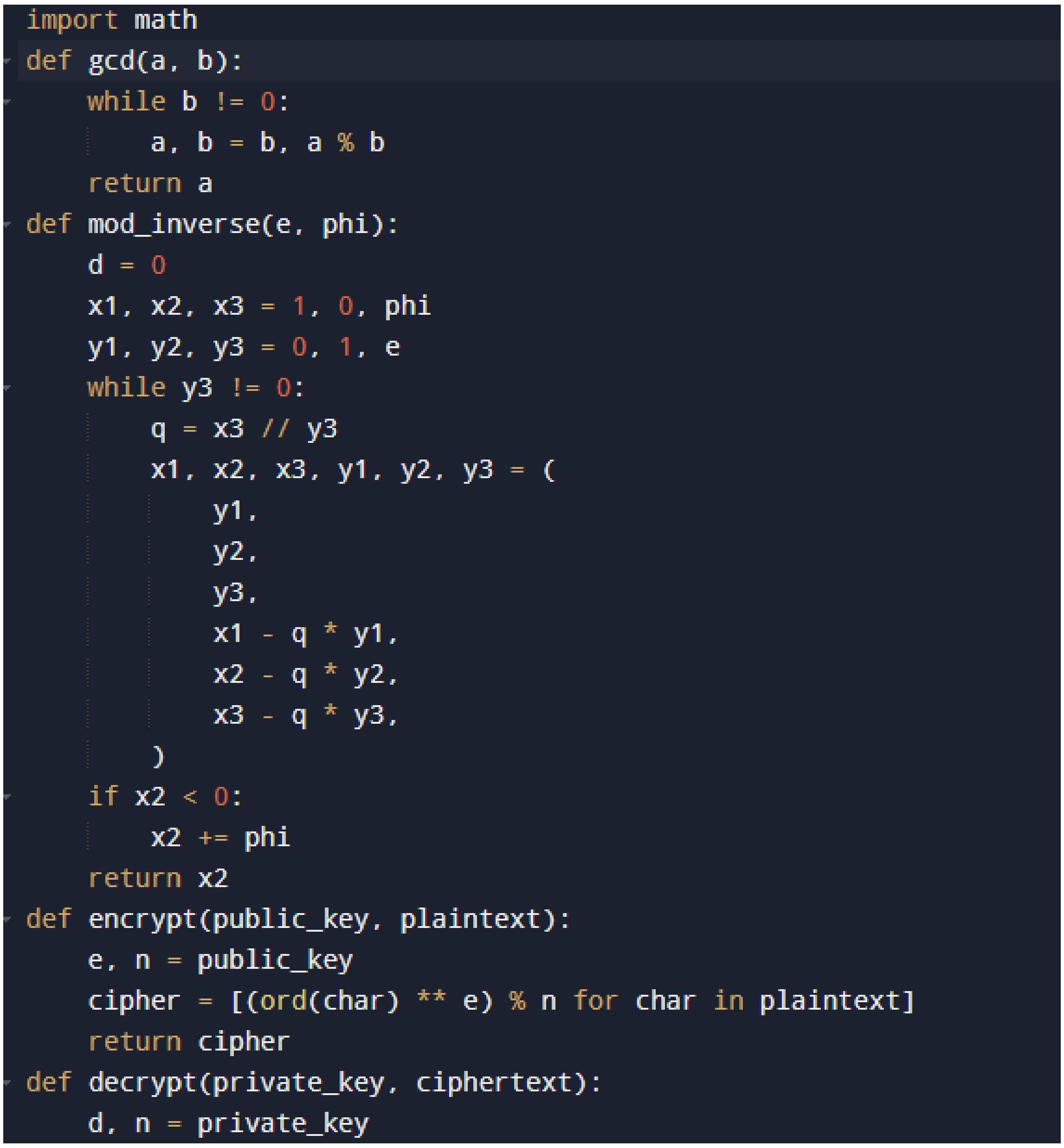
   Step-4: Select a random prime number e that is less than that of z.

   Step-5: Compute the private key, d as e * mod-1(z).

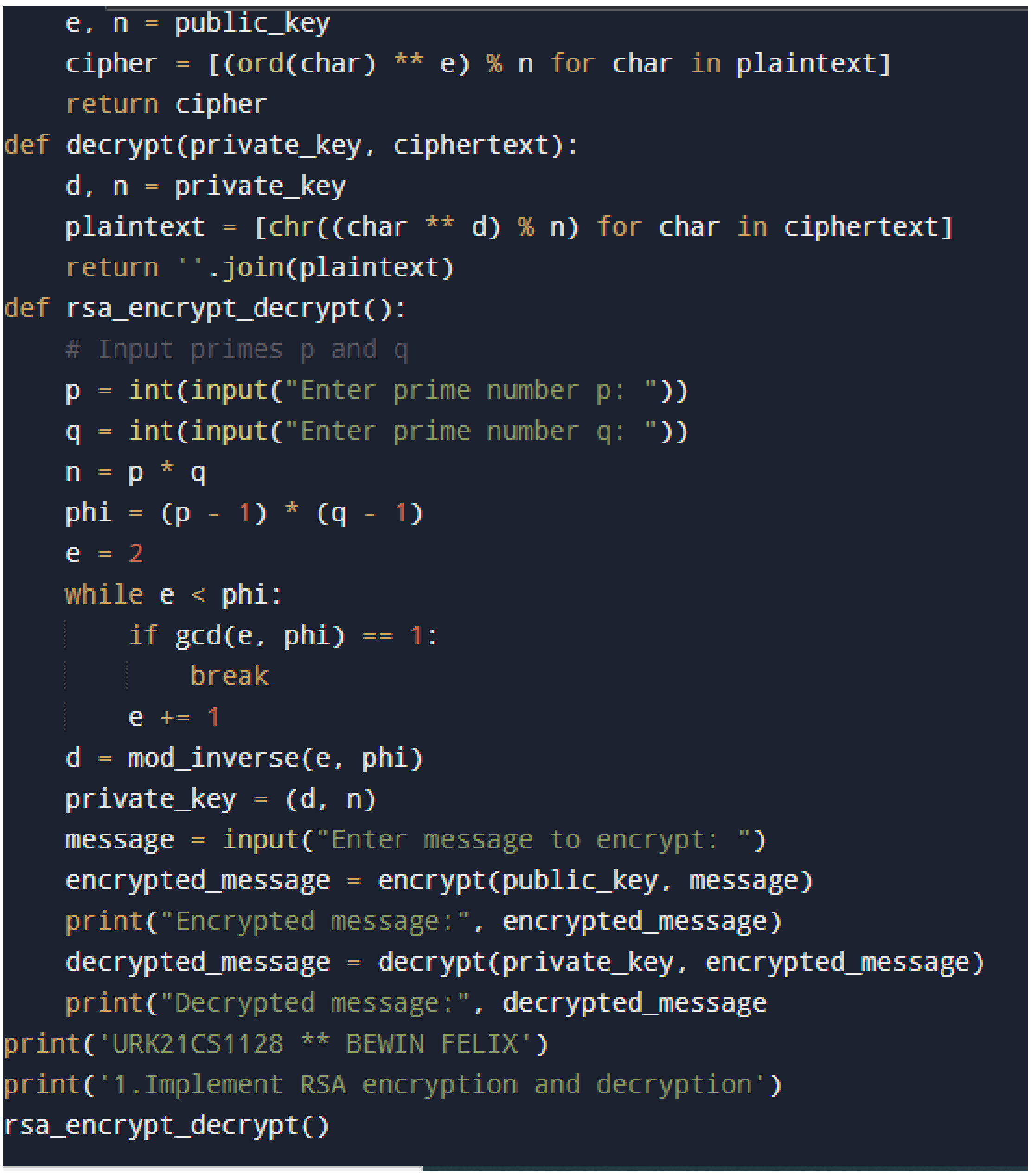
   Step-6: The cipher text is computed as message e * mod n.

   Step-7: Decryption is done as cipher d mod n.

Program

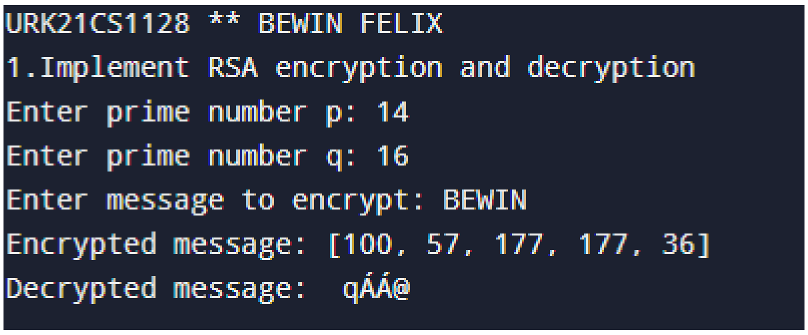


```python
import math
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
def mod_inverse(e, phi):
    d = 0
    x1, x2, x3 = 1, 0, phi
    y1, y2, y3 = 0, 1, e
    while y3 != 0:
        q = x3 // y3
        x1, x2, x3, y1, y2, y3 = (
            y1,
            y2,
            y3,
            x1 - q * y1,
            x2 - q * y2,
            x3 - q * y3,
        )
    if x2 < 0:
        x2 += phi
    return x2
def encrypt(public_key, plaintext):
    e, n = public_key
    cipher = [(ord(char) ** e) % n for char in plaintext]
    return cipher
def decrypt(private_key, ciphertext):
    d, n = private_key
```



```
    e, n = public_key
    cipher = [(ord(char) ** e) % n for char in plaintext]
    return cipher
def decrypt(private_key, ciphertext):
    d, n = private_key
    plaintext = [chr((char ** d) % n) for char in ciphertext]
    return ''.join(plaintext)
def rsa_encrypt_decrypt():
    # Input primes p and q
    p = int(input("Enter prime number p: "))
    q = int(input("Enter prime number q: "))
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 2
    while e < phi:
        if gcd(e, phi) == 1:
            break
        e += 1
    d = mod_inverse(e, phi)
    private_key = (d, n)
    message = input("Enter message to encrypt: ")
    encrypted_message = encrypt(public_key, message)
    print("Encrypted message:", encrypted_message)
    decrypted_message = decrypt(private_key, encrypted_message)
    print("Decrypted message:", decrypted_message
print('URK21CS1128 ** BEWIN FELIX')
print('1.Implement RSA encryption and decryption')
rsa_encrypt_decrypt()
```

Output Screenshot



```
URK21CS1128 ** BEWIN FELIX
1.Implement RSA encryption and decryption
Enter prime number p: 14
Enter prime number q: 16
Enter message to encrypt: BEWIN
Encrypted message: [100, 57, 177, 177, 36]
Decrypted message:  qÁÁ@
```

2. **In a Diffie-Hellman Key Exchange, Alice and Bob have chosen prime value q = 17 and primitive root = 5. If Alice's secret key is 4 and Bob's secret key is 6, Compute the secret key exchanged between them.**

Algorithm

Step 1: Select a large prime number p and a primitive root g modulo p (both are public).

Step 2: Alice selects a private key a such that 1 < a < p − 1.

Step 3: Bob selects a private key b such that 1 < b < p – 1

Step 4: Alice computes her public key A = g^a mod p and sends A to Bob.

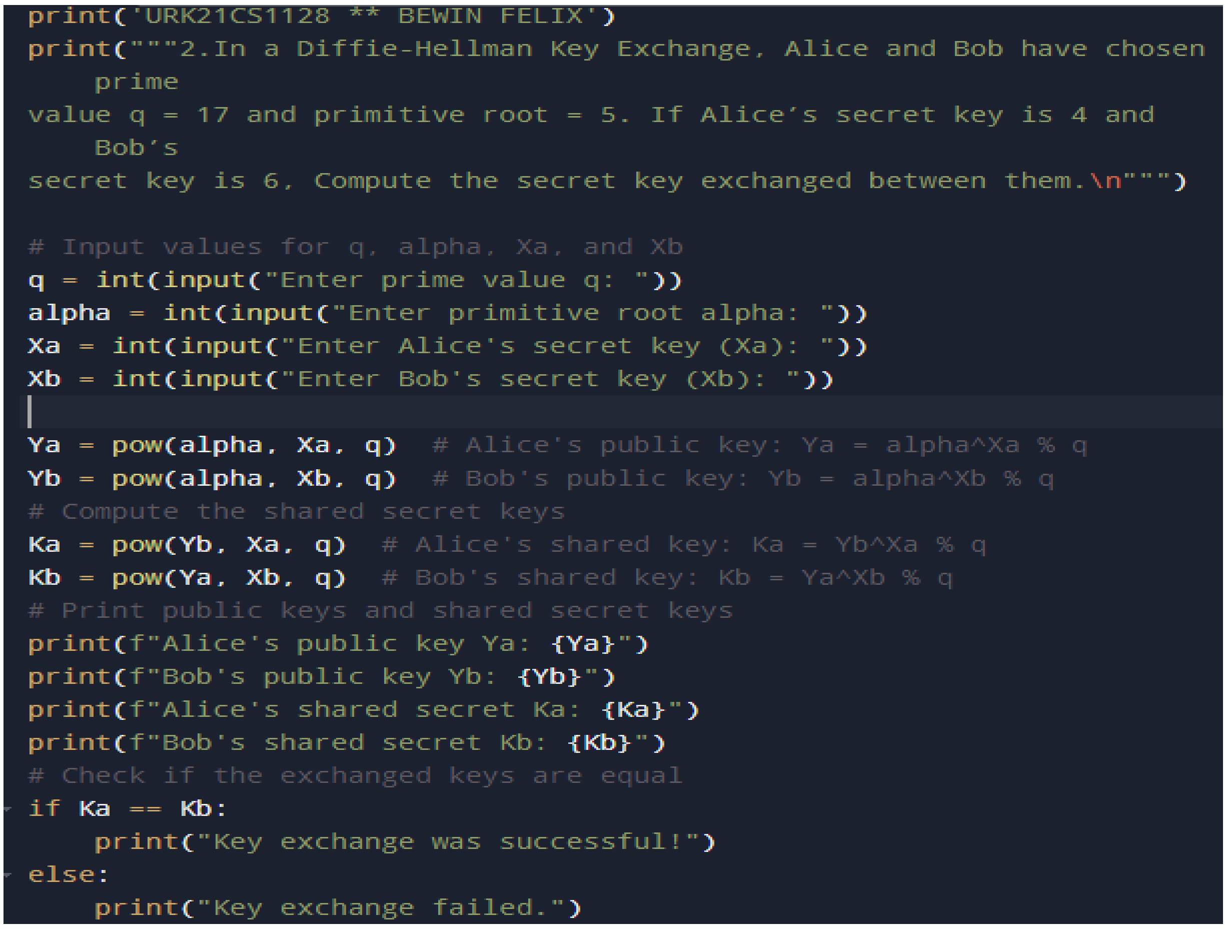
Step 5: Bob computes his public key B = g^b mod p and sends B to Alice.

Step 6: Alice computes the shared secret key SA=B^a mod p.

Step 7: Bob computes the shared secret key SB=A^b mod p.

Step 8: Since SA=SB, the shared secret key S = g^ab mod p is established.

Program

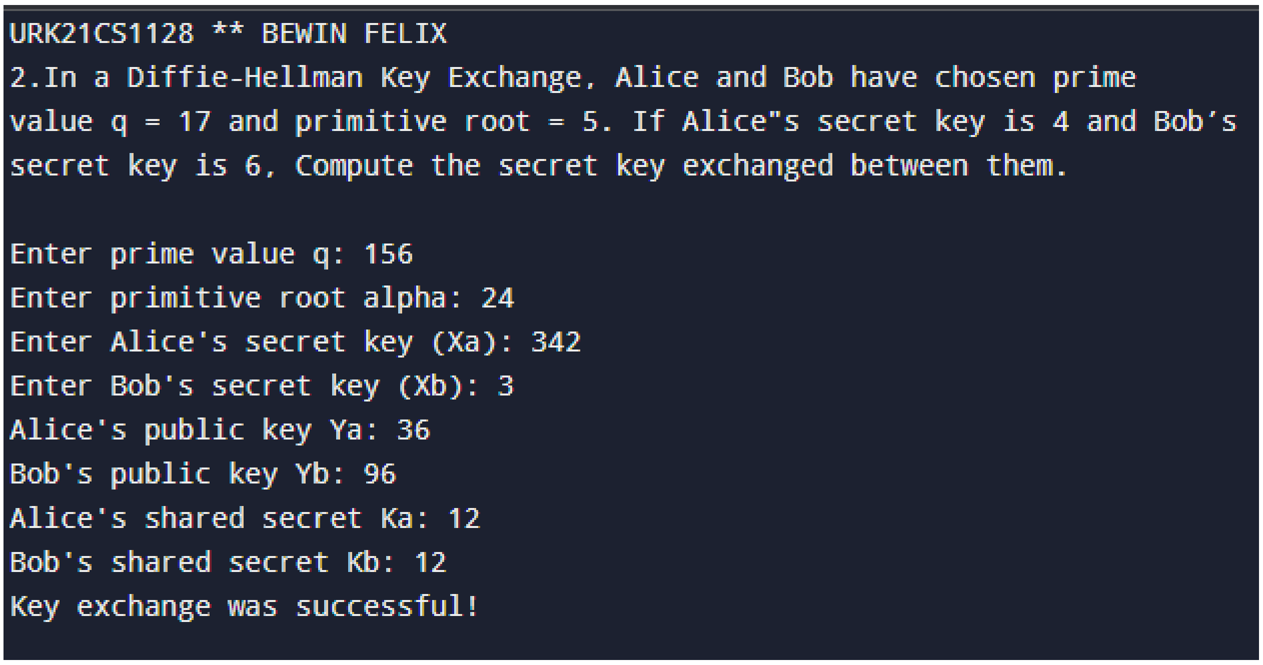


```
print('URK21CS1128 ** BEWIN FELIX')
print("""2.In a Diffie-Hellman Key Exchange, Alice and Bob have chosen
    prime
value q = 17 and primitive root = 5. If Alice’s secret key is 4 and
    Bob’s
secret key is 6, Compute the secret key exchanged between them.\n""")

# Input values for q, alpha, Xa, and Xb
q = int(input("Enter prime value q: "))
alpha = int(input("Enter primitive root alpha: "))
Xa = int(input("Enter Alice's secret key (Xa): "))
Xb = int(input("Enter Bob's secret key (Xb): "))

Ya = pow(alpha, Xa, q)  # Alice's public key: Ya = alpha^Xa % q
Yb = pow(alpha, Xb, q)  # Bob's public key: Yb = alpha^Xb % q
# Compute the shared secret keys
Ka = pow(Yb, Xa, q)  # Alice's shared key: Ka = Yb^Xa % q
Kb = pow(Ya, Xb, q)  # Bob's shared key: Kb = Ya^Xb % q
# Print public keys and shared secret keys
print(f"Alice's public key Ya: {Ya}")
print(f"Bob's public key Yb: {Yb}")
print(f"Alice's shared secret Ka: {Ka}")
print(f"Bob's shared secret Kb: {Kb}")
# Check if the exchanged keys are equal
if Ka == Kb:
    print("Key exchange was successful!")
else:
    print("Key exchange failed.")
```

Output Screenshot



```
URK21CS1128 ** BEWIN FELIX
2.In a Diffie-Hellman Key Exchange, Alice and Bob have chosen prime
value q = 17 and primitive root = 5. If Alice"s secret key is 4 and Bob’s
secret key is 6, Compute the secret key exchanged between them.

Enter prime value q: 156
Enter primitive root alpha: 24
Enter Alice's secret key (Xa): 342
Enter Bob's secret key (Xb): 3
Alice's public key Ya: 36
Bob's public key Yb: 96
Alice's shared secret Ka: 12
Bob's shared secret Kb: 12
Key exchange was successful!
```

3. Demonstrate El gammal’s encryption and decryption

Algorithm

Step 1: Select a large prime number p and a primitive root g modulo p (both are public).

Step 2: Choose a private key a such that $1 < a < p – 1$.

Step 3: Compute the public key A = g^a mod p and share A, p, and g.

Step 4: Select a random integer k such that $1 \leq k \leq p – 2$ and compute the shared value K = A^k mod .

Step 5: Encrypt the message M by computing two ciphertext values: C1, C2.

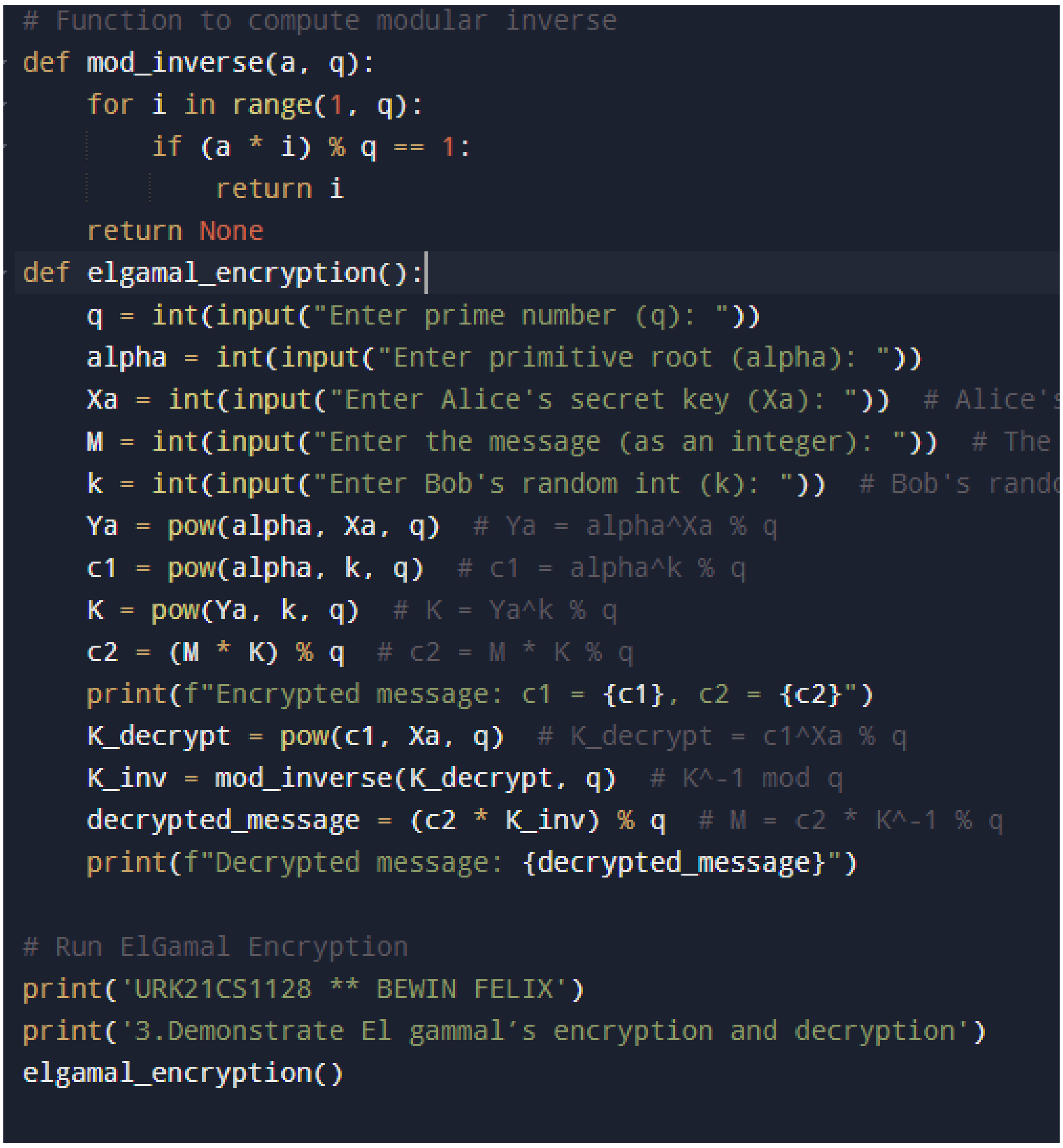
Step 6: Send the ciphertext pair (C1,C2).

Step 7: Upon receiving the ciphertext, compute the shared value K = C1^a mod p.

Step 8: Decrypt the message by computing M.

Step 9: The original message M is now recovered.

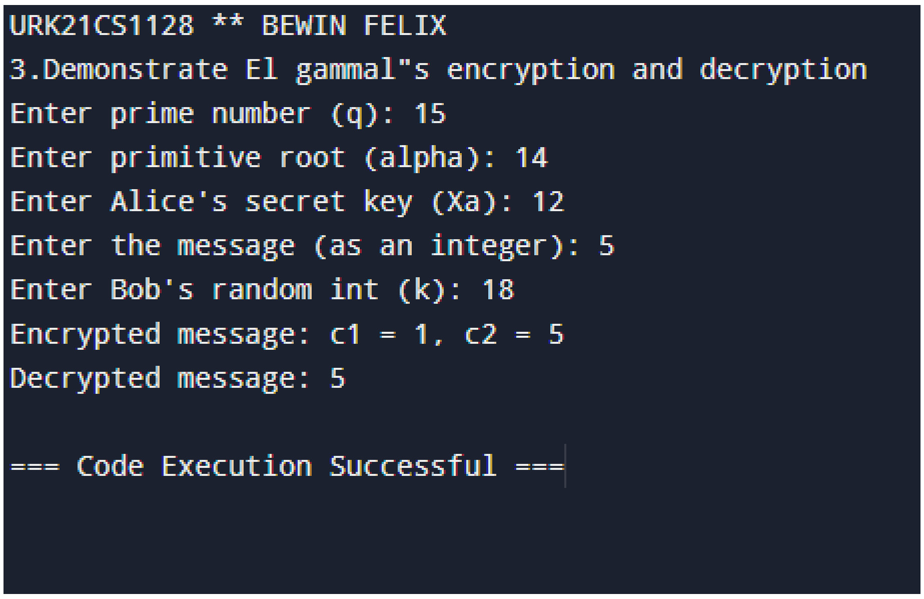
Program



```python
# Function to compute modular inverse
def mod_inverse(a, q):
    for i in range(1, q):
        if (a * i) % q == 1:
            return i
    return None
def elgamal_encryption():
    q = int(input("Enter prime number (q): "))
    alpha = int(input("Enter primitive root (alpha): "))
    Xa = int(input("Enter Alice's secret key (Xa): "))  # Alice'
    M = int(input("Enter the message (as an integer): "))  # The
    k = int(input("Enter Bob's random int (k): "))  # Bob's rand
    Ya = pow(alpha, Xa, q)  # Ya = alpha^Xa % q
    c1 = pow(alpha, k, q)  # c1 = alpha^k % q
    K = pow(Ya, k, q)  # K = Ya^k % q
    c2 = (M * K) % q  # c2 = M * K % q
    print(f"Encrypted message: c1 = {c1}, c2 = {c2}")
    K_decrypt = pow(c1, Xa, q)  # K_decrypt = c1^Xa % q
    K_inv = mod_inverse(K_decrypt, q)  # K^-1 mod q
    decrypted_message = (c2 * K_inv) % q  # M = c2 * K^-1 % q
    print(f"Decrypted message: {decrypted_message}")

# Run ElGamal Encryption
print('URK21CS1128 ** BEWIN FELIX')
print('3.Demonstrate El gammal’s encryption and decryption')
elgamal_encryption()
```

Output Screenshot



```
URK21CS1128 ** BEWIN FELIX
3.Demonstrate El gammal"s encryption and decryption
Enter prime number (q): 15
Enter primitive root (alpha): 14
Enter Alice's secret key (Xa): 12
Enter the message (as an integer): 5
Enter Bob's random int (k): 18
Encrypted message: c1 = 1, c2 = 5
Decrypted message: 5

=== Code Execution Successful ===
```

Result

Thus, the encryption and decryption using the Asymmetric Cipher Algorithms were implemented and the same was verified successfully.