**Final Year B.Tech. (CSE) – VII [ 2024-25]**

**6CS451: Cryptography and Network Security Lab (C&NS Lab)**

**Date: 26/08/2024**

**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 φ(n) = (p-1) \* (q-1).
   * Choose an encryption key e such that 1 < e < φ(n) and gcd(e, φ(n)) = 1. The integer e is the public key exponent.
   * Calculate the decryption key d such that d \* e ≡ 1 (mod φ(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 mod n.

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

M = C^d mod 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 gcd(a, b):

    """Compute the greatest common divisor using Euclid's algorithm."""

    while b != 0:

        a, b = b, a % b

    return a

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 (prime):", p)

    print("q (prime):", q)

    # Compute n = p \* q

    n = p \* q

    print("n (p \* q):", n)

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

    phi = (p - 1) \* (q - 1)

    print("phi (Euler's Totient):", phi)

    # 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)

    print("e (public exponent):", e)

    # Compute d, the modular inverse of e

    d = mod\_inverse(e, phi)

    print("d (private exponent):", d)

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

    return ((e, n), (d, n))

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]

    print("\nIntermediate Encryption Steps:")

    for i, char in enumerate(plaintext):

        print(f"Character '{char}' -> Cipher Value: {cipher[i]}")

    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]

    print("\nIntermediate Decryption Steps:")

    for i, val in enumerate(ciphertext):

        print(f"Cipher Value {val} -> Character: {plain[i]}")

    return ''.join(plain)

def main():

    """Run RSA algorithm with menu-driven interface."""

    public\_key, private\_key = None, None

    while True:

        print("\nMenu:")

        print("1. Generate Key Pair")

        print("2. Encrypt a Message")

        print("3. Decrypt a Message")

        print("4. Quit")

        choice = input("\nEnter your choice: ")

        if choice == '1':

            keysize = int(input("Enter key size (e.g., 1024, 2048): "))

            public\_key, private\_key = generate\_keypair(keysize)

            print(f"\nPublic key: {public\_key}")

            print(f"Private key: {private\_key}")

        elif choice == '2':

            if not public\_key:

                print("Please generate keys first (Option 1).")

                continue

            plaintext = input("Enter a message to encrypt: ")

            encrypted\_msg = encrypt(public\_key, plaintext)

            print(f"\nEncrypted Message: {encrypted\_msg}")

        elif choice == '3':

            if not private\_key:

                print("Please generate keys first (Option 1).")

                continue

            encrypted\_msg = input("Enter the encrypted message (as a list of integers): ")

            try:

                encrypted\_msg = list(map(int, encrypted\_msg.strip('[]').split(',')))

            except ValueError:

                print("Invalid input. Please enter integers separated by commas.")

                continue

            decrypted\_msg = decrypt(private\_key, encrypted\_msg)

            print(f"\nDecrypted Message: {decrypted\_msg}")

        elif choice == '4':

            print("Exiting the program.")

            break

        else:

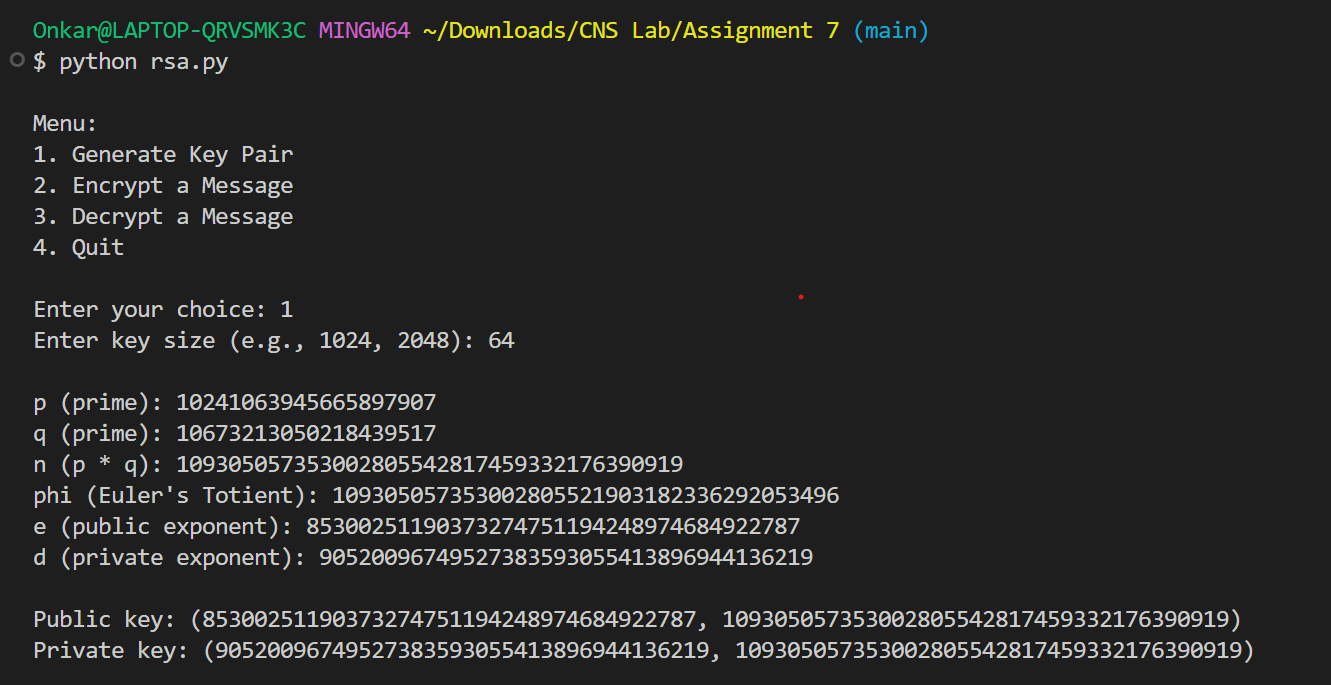
            print("Invalid choice. Please enter 1, 2, 3, or 4.")

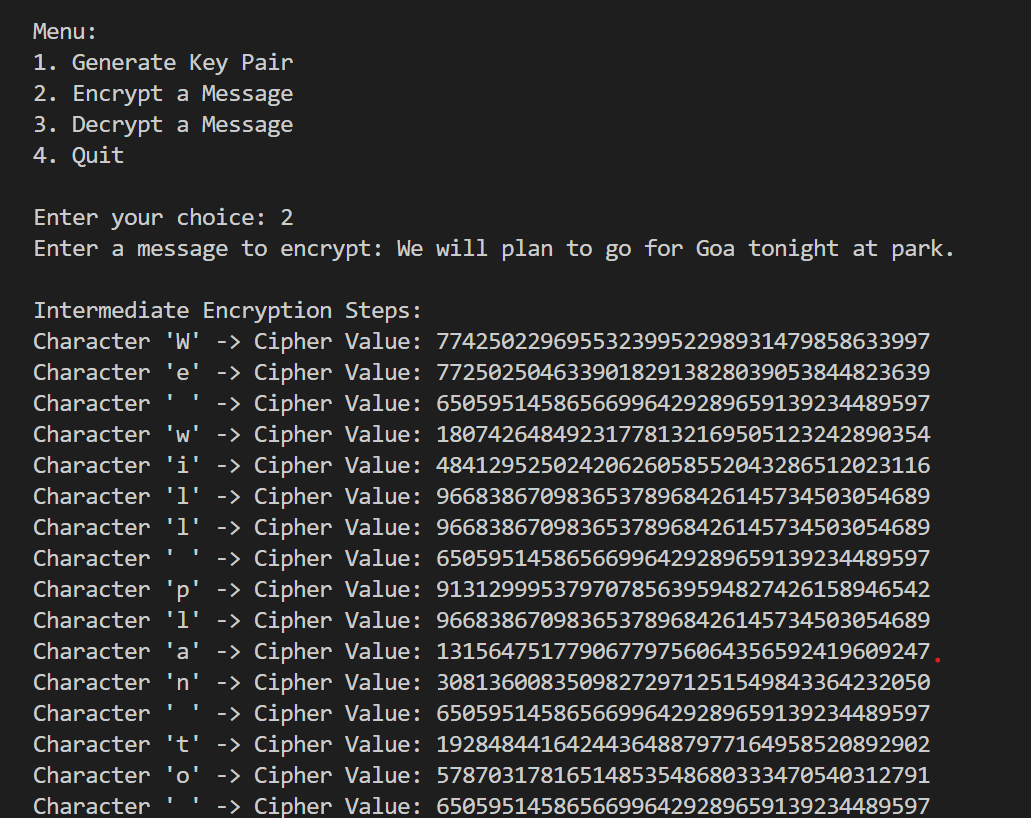
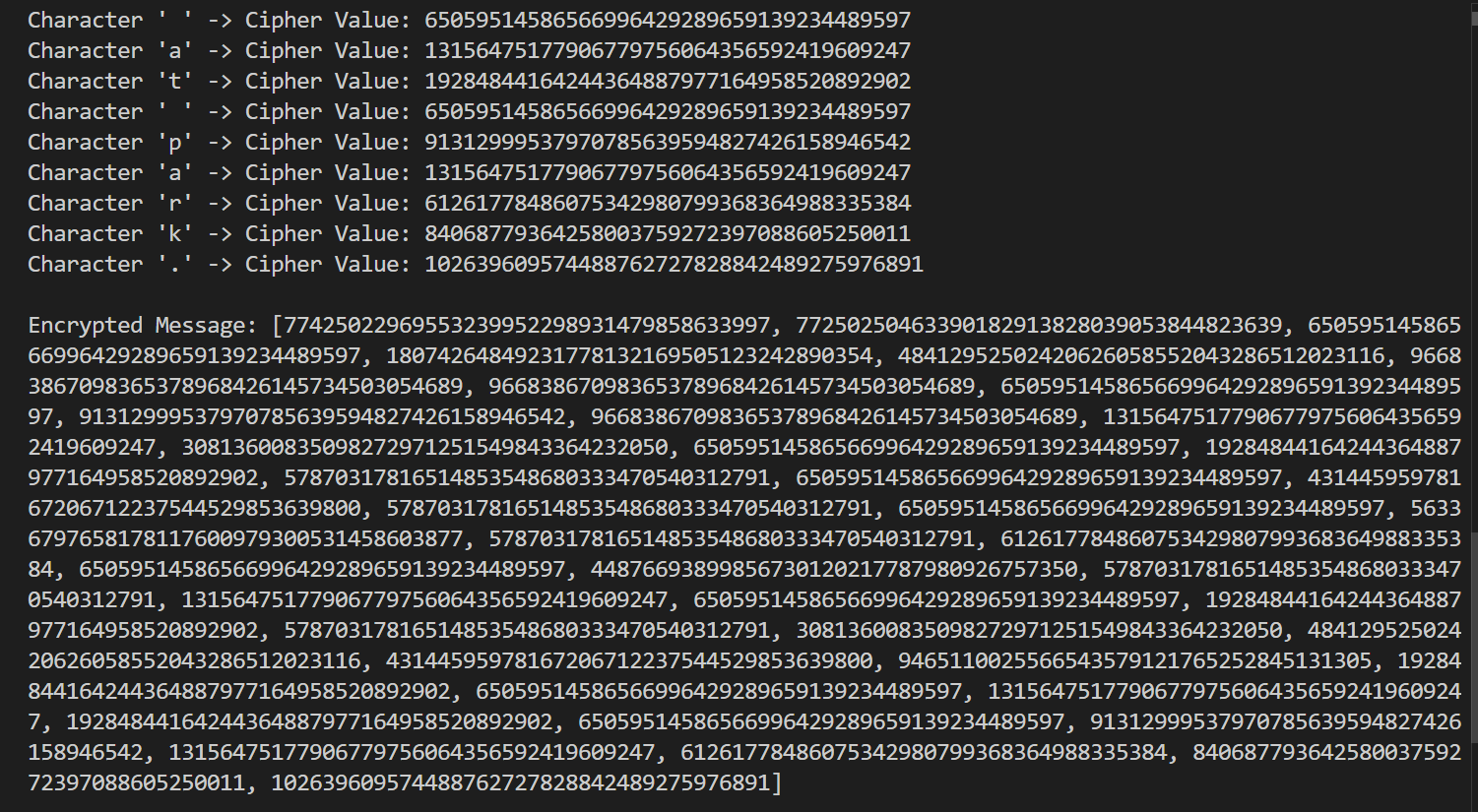
if \_\_name\_\_ == "\_\_main\_\_":

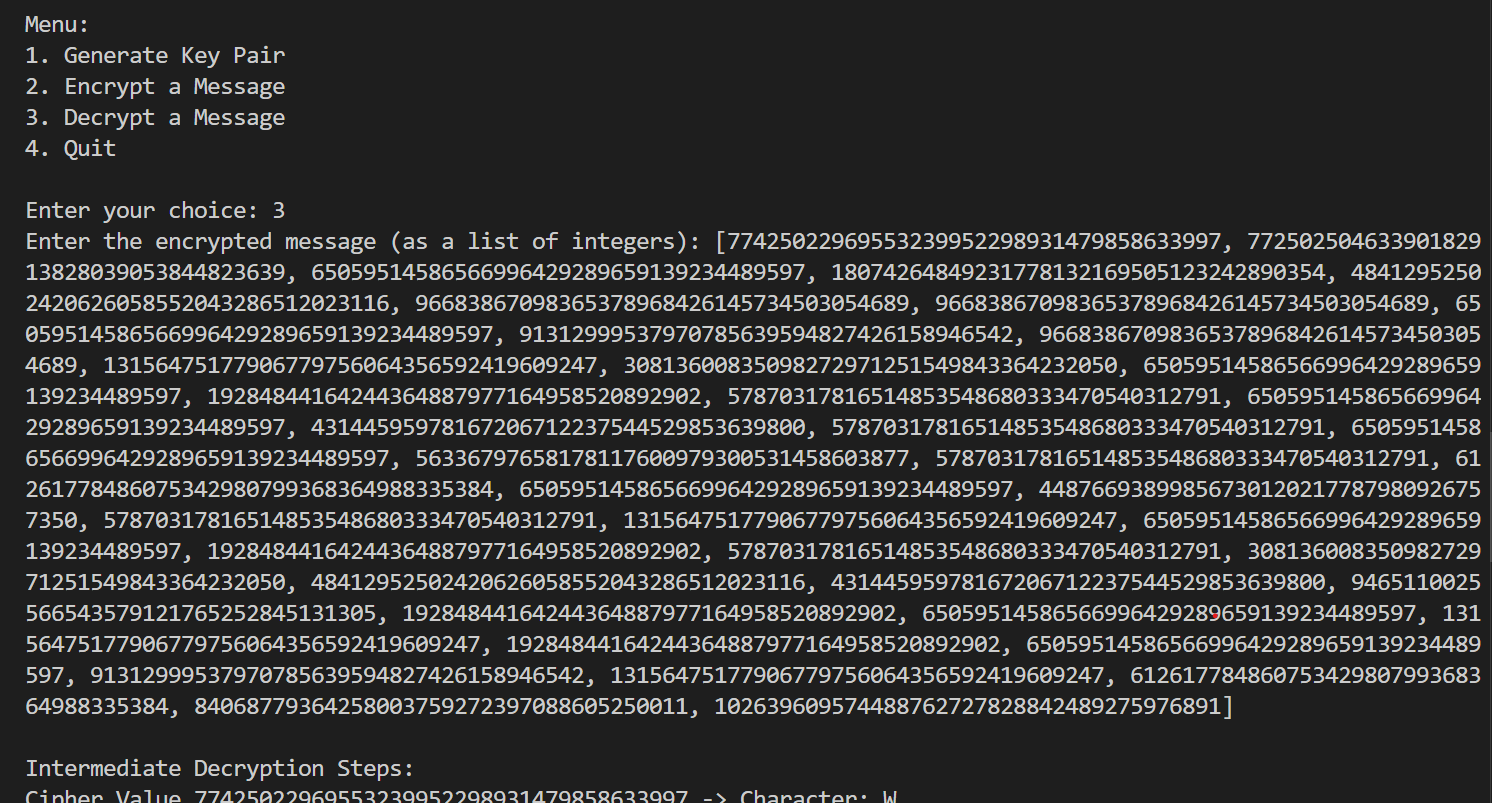
    main()

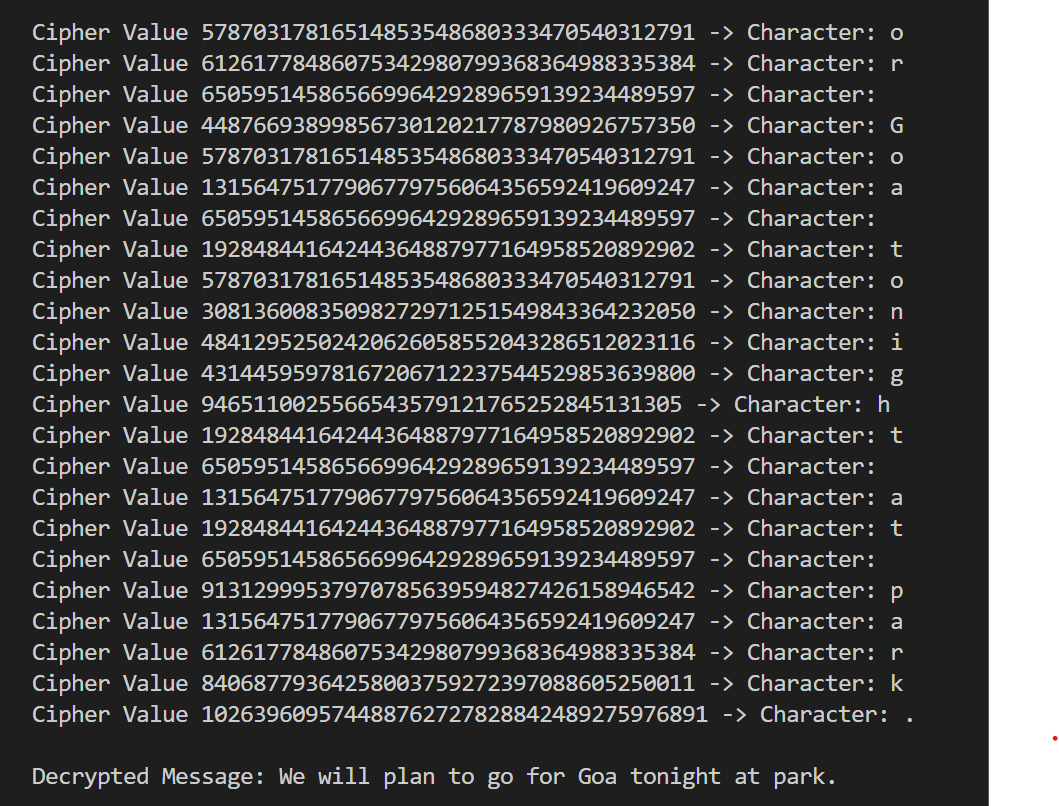
**Output:**

**For Keysize = 64:**







**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.