CNS LAB 8

SAHIL BONDRE: U18CO021

Write a program to implement the following RSA functions with large prime numbers.
(a) Key Generation
(b) Encryption
(c) Decryption
User can input plaintext as a number or text.

Code:

```python
import random



def recursive_read(allowed_input, message=""):
    # Recursively reads user input until input is not in allowd_input
    while True:
        user_input = input(message)
        if user_input in allowed_input:
            return user_input



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



'''
```

```
Euclid's extended algorithm for finding the multiplicative inverse of two
numbers

'''



def multiplicative_inverse(e, phi):

    d = 0

    x1 = 0

    x2 = 1

    y1 = 1

    temp_phi = phi


    while e > 0:

        temp1 = temp_phi//e

        temp2 = temp_phi - temp1 * e

        temp_phi = e

        e = temp2


        x = x2 - temp1 * x1

        y = d - temp1 * y1


        x2 = x1

        x1 = x

        d = y1

        y1 = y


    if temp_phi == 1:
```

```python
        return d + phi



def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True



def generate_key_pair(p, q):
    if not (is_prime(p) and is_prime(q)):
        raise ValueError('Both numbers must be prime.')
    elif p == q:
        raise ValueError('p and q cannot be equal')
    # n = pq
    n = p * q

    # Phi is the totient of n
    phi = (p-1) * (q-1)

    # Choose an integer e such that e and phi(n) are coprime
```

```python
    e = random.randrange(1, phi)


    # Use Euclid's Algorithm to verify that e and phi(n) are coprime

    g = gcd(e, phi)

    while g != 1:

        e = random.randrange(1, phi)

        g = gcd(e, phi)


    # Use Extended Euclid's Algorithm to generate the private key

    d = multiplicative_inverse(e, phi)


    # Return public and private key_pair

    # Public key is (e, n) and private key is (d, n)

    return ((e, n), (d, n))




def encrypt(pk, plaintext):

    # Unpack the key into it's components

    key, n = pk

    # Convert each letter in the plaintext to numbers based on the character
using a^b mod m

    cipher = [pow(ord(char), key, n) for char in plaintext]

    # Return the array of bytes

    return cipher




def decrypt(pk, ciphertext):
```

```
    # Unpack the key into its components
    key, n = pk
    # Generate the plaintext based on the ciphertext and key using a^b mod m
    aux = [str(pow(char, key, n)) for char in ciphertext]
    # Return the array of bytes as a string
    plain = [chr(int(char2)) for char2 in aux]
    return ''.join(plain)


code = recursive_read(
    ["g", "e", "d"], "Enter 'g' for key generation, 'e' for encryption or 'd'
for decryption: ")


def perform_encryption():
    public_key = [int(x) for x in input("Enter public key: ").split(" ")]
    msg = input("Enter Message: ")
    encrypted_msg = encrypt(public_key, msg)
    print(
        f"Encrypted Message: {''.join(map(lambda x: str(x) + ' ',
encrypted_msg))}")


def perform_decryption():
    private_key = [int(x) for x in input("Enter private key: ").split(" ")]
    msg = [int(x) for x in input("Enter Cipher Text: ").split(' ')]
    decrypted_msg = decrypt(private_key, msg)
```

```python
    print(f"Decrypted Message: {decrypted_msg}")


def perform_key_gen():
    p = int(input("Enter a prime number: "))
    q = int(input("Enter another prime nubmer: "))
    try:
        public, private = generate_key_pair(p, q)
    except:
        print("Error: Invalid Numbers Supplied")
        exit()
    finally:
        print(f"Public key: {public}")
        print(f"Private key: {private}")


if code == "e":
    perform_encryption()
elif code == "d":
    perform_decryption()
else:
    perform_key_gen()
```

```
PS F:\code\github.com\godcrampy\college-notes\cns\lab-08> python .\rsa.py
Enter 'g' for key generation, 'e' for encryption or 'd' for decryption: g
Enter a prime number: 17
Enter another prime nubmer: 67
Public key: (505, 1139)
Private key: (1033, 1139)
PS F:\code\github.com\godcrampy\college-notes\cns\lab-08> python .\rsa.py
Enter 'g' for key generation, 'e' for encryption or 'd' for decryption: e
Enter public key: 505 1139
Enter Message: hello world @123
Encrypted Message: 104 543 130 130 995 355 527 995 906 130 797 355 625 151 152 731 355
PS F:\code\github.com\godcrampy\college-notes\cns\lab-08> python .\rsa.py
Enter 'g' for key generation, 'e' for encryption or 'd' for decryption: d
Enter private key: 1033 1139
Enter Cipher Text: 104 543 130 130 995 355 527 995 906 130 797 355 625 151 152 731 355
Decrypted Message: hello world @123
PS F:\code\github.com\godcrampy\college-notes\cns\lab-08>
```