LAB 3

Write a program using any Object-oriented programming language to show the implementation of RSA. The input p and q should be generated by randomly (15 Marks)

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

To encrypt a message, M, with the public key, create the ciphertext, C, using the

equation: $C = M^e \bmod n$

The receiver then decrypts the ciphertext with the private key using the

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

Evaluation Criteria

1. Correctness of code
2. Random number Generation of p, q
3. Generation of multiple pairs of PU (e,n) and PR Keys (d,n)
4. Encryption and Decryption of input message
5. A lab report showing your code , sample outputs and code explanation

Lab Report: Implementation of RSA Encryption and Decryption in Python

Objective

The objective of this lab is to implement the RSA encryption and decryption algorithm in Python. The program will generate random prime numbers, create public and private keys, and allow the user to encrypt and decrypt a message.

Introduction

RSA (Rivest-Shamir-Adleman) is a public-key cryptosystem that is widely used for secure data transmission. In RSA, a pair of keys is generated: a public key, which is used for encryption, and a private key, which is used for decryption.

Main Program

```
import random

import math


def is_prime(n):

    if n < 2:

        return False

    for i in range(2, int(math.sqrt(n)) + 1):

        if n % i == 0:

            return False

    return True


def generate_prime(min_val, max_val):

    prime = random.randint(min_val, max_val)

    while not is_prime(prime):

        prime = random.randint(min_val, max_val)

    return prime
```

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def mod_inverse(e, phi):
    g, x, y = egcd(e, phi)
    if g != 1:
        raise Exception('Modular inverse does not exist')
    else:
        return x % phi

def generate_keypair():
    # Generate p and q
    p = generate_prime(10, 50)
    q = generate_prime(10, 50)
    while p == q:
        q = generate_prime(10, 50)

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

    # Choose e
    e = random.randint(2, phi - 1)
```

```
    while math.gcd(e, phi) != 1:
        e = random.randint(2, phi - 1)

    # Compute d
    d = mod_inverse(e, phi)

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

def generate_multiple_keypairs(num_pairs):
    keypairs = []
    for _ in range(num_pairs):
        keypairs.append(generate_keypair())
    return keypairs

def encrypt(pk, plaintext):
    e, n = pk
    cipher = [(ord(char) ** e) % n for char in plaintext]
    return cipher

def decrypt(pk, ciphertext):
    d, n = pk
    plain = [chr((char ** d) % n) for char in ciphertext]
    return ''.join(plain)

# Generate multiple keypairs
num_pairs = 3
keypairs = generate_multiple_keypairs(num_pairs)
```

```
# Print the generated keypairs
for i, (public_key, private_key) in enumerate(keypairs, 1):
    print(f"Keypair {i}:")
    print(f"Public Key (e, n): {public_key}")
    print(f"Private Key (d, n): {private_key}")
    print()

# Get user input message for encryption and decryption
message = input("Enter a message to encrypt and decrypt: ")
print(f"Original message: {message}")

# Using the first key pair for encryption and decryption
public_key, private_key = keypairs[0]
encrypted_message = encrypt(public_key, message)
print(f"Encrypted message: {encrypted_message}")

decrypted_message = decrypt(private_key, encrypted_message)
print(f"Decrypted message: {decrypted_message}")
```

Code Explanation

Prime Number Generation

The first step in RSA is to generate two large prime numbers. For simplicity and ease of testing, I have generate small prime numbers in the range of 10 to 50.

is_prime(n): This function checks if a number n is a prime number by testing divisibility from 2 to the square root of n.

generate_prime(min_val, max_val): This function generates a random prime number within the specified range by repeatedly generating random numbers and checking for primality.

Extended Euclidean Algorithm

The Extended Euclidean Algorithm is used to find the modular inverse, which is crucial for calculating the private key.

`egcd(a, b)`: This function implements the Extended Euclidean Algorithm to find the greatest common divisor (gcd) of `a` and `b`, along with the coefficients `x` and `y` such that `ax + by = gcd(a, b)`.

Modular Inverse

The modular inverse is used to compute the private key `d` from `e` and `phi`.

`mod_inverse(e, phi)`: This function uses the Extended Euclidean Algorithm to find the modular inverse of `e` modulo `phi`. If the gcd is not 1, it raises an exception, indicating that the modular inverse does not exist.

Key Pair Generation

`generate_keypair()`: This function generates two distinct prime numbers `p` and `q`, computes `n = p * q` and `phi = (p - 1) * (q - 1)`. It then selects a public exponent `e` such that `1 < e < phi` and `gcd(e, phi) = 1`, and computes the private key `d` as the modular inverse of `e` modulo `phi`.

Multiple Key Pair Generation

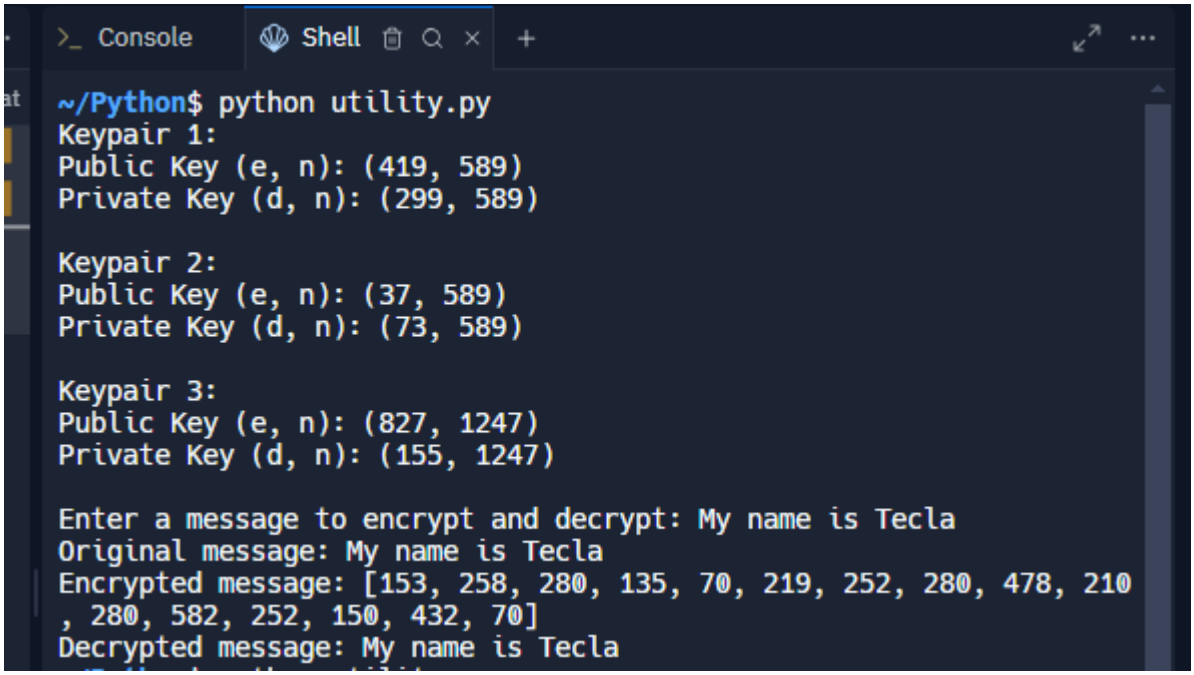
`generate_multiple_keypairs(num_pairs)`: This function generates the specified number of RSA key pairs and returns them in a list.

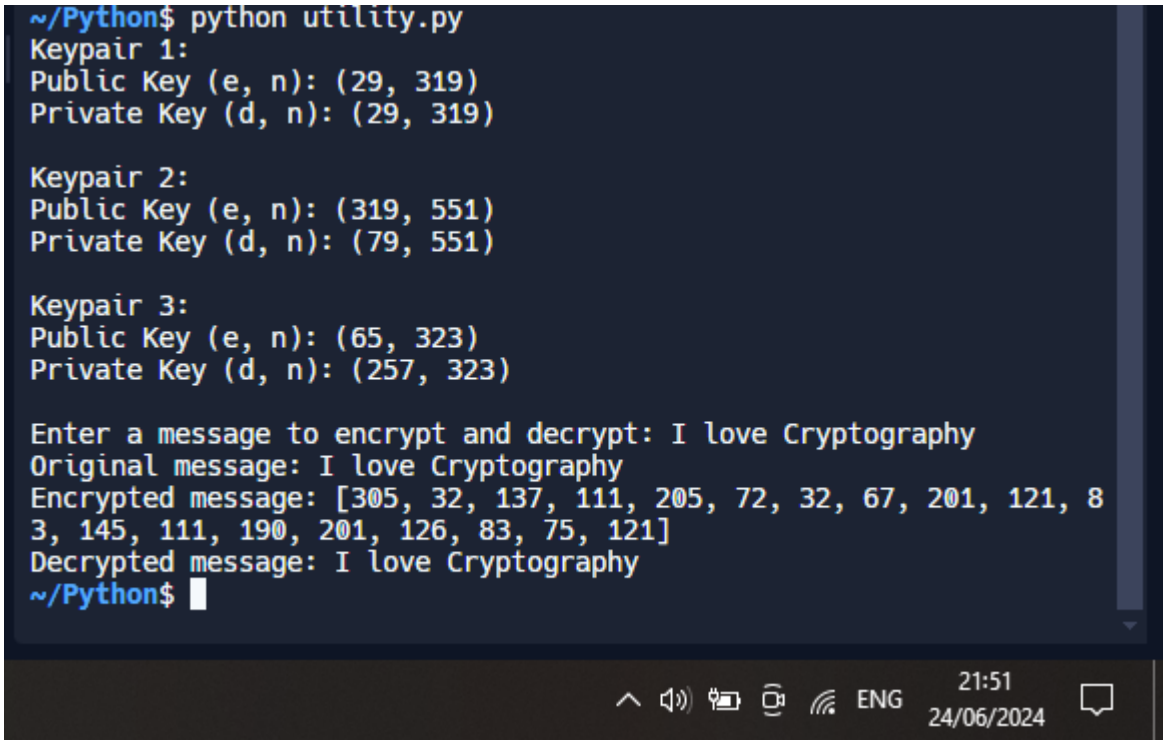
Encryption and Decryption

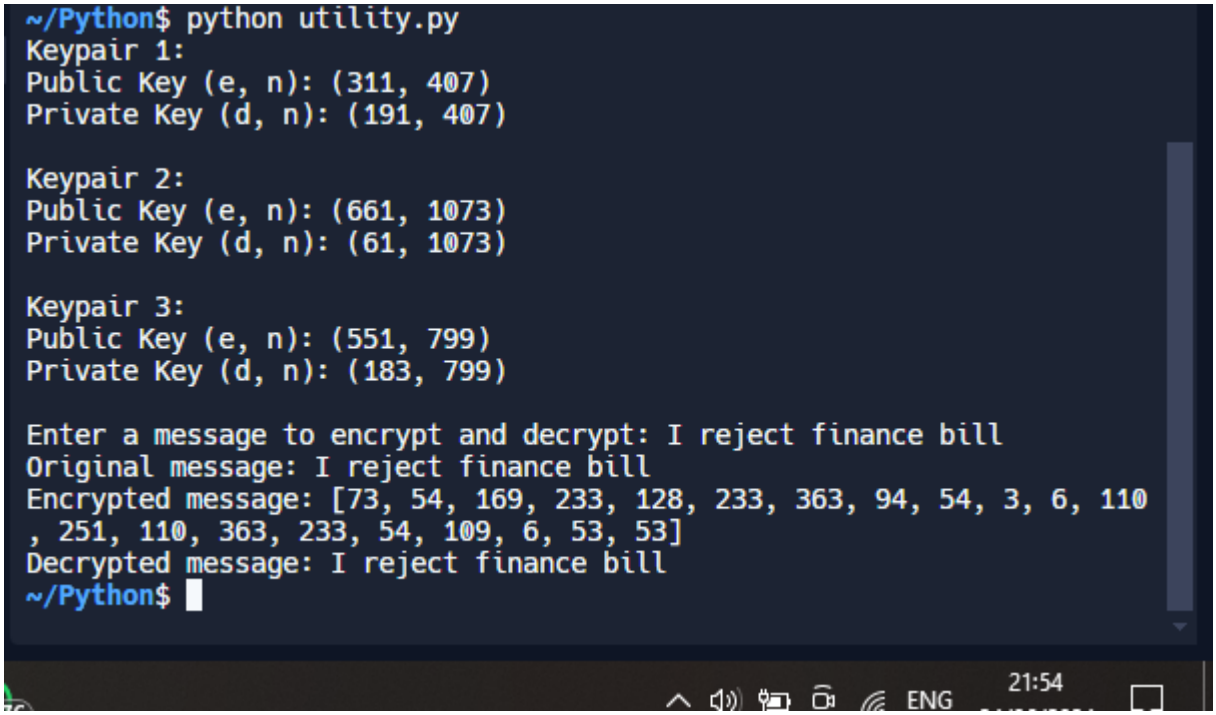
encrypt(pk, plaintext): This function encrypts the plaintext message using the public key pk by raising each character's ASCII value to the power of e and taking modulo n.

decrypt(pk, ciphertext): This function decrypts the ciphertext using the private key pk by raising each encrypted value to the power of d and taking modulo n, converting the resulting values back to characters.

SAMPLE OUTPUTS






~/Python$ python utility.py
Keypair 1:
Public Key (e, n): (311, 407)
Private Key (d, n): (191, 407)
Keypair 2:
Public Key (e, n): (661, 1073)
Private Key (d, n): (61, 1073)
Keypair 3:
Public Key (e, n): (551, 799)
Private Key (d, n): (183, 799)
Enter a message to encrypt and decrypt: I reject finance bill
Original message: I reject finance bill
Encrypted message: [73, 54, 169, 233, 128, 233, 363, 94, 54, 3, 6, 110
, 251, 110, 363, 233, 54, 109, 6, 53, 53]
Decrypted message: I reject finance bill
~/Python$
ENG
21:54