USIU AFRICA

United States International University-Africa

SUMMER SEMESTER 2024

Name: Jodick Ndayisenga

ID: 666225

Major: APT

APT3090 CRYPTOGRAPHY AND NETWORK SECURITY

Write a program using any Object oriented programming language to show 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: $M = C^d \bmod n$

RESPONSE

```
import random
from sympy import isprime, mod_inverse

def generate_prime_candidate(length):
    # Generate random number of specified bit length
    p = random.getrandbits(length)
    # Apply a mask to set MSB and LSB to 1
    p |= (1 << length - 1) | 1
    return p

def generate_prime_number(length=256):
    p = 4
    # Keep generating while the number is not prime
    while not isprime(p):
        p = generate_prime_candidate(length)
    return p

def generate_key_pair(bit_length=256):
    # Generate two distinct prime numbers p and q
    p = generate_prime_number(bit_length)
    q = generate_prime_number(bit_length)
    while q == p:
        q = generate_prime_number(bit_length)
    n = p * q
    print(f"These are the two prime numbers: p: {p} and q: {q}")
    print("This is n: {}".format(n))
    # Compute the Euler's totient function phi(n) = (p-1)(q-1)
    phi = (p - 1) * (q - 1)
    print("This is phi, the Euler's totient function phi(n) = (p-1)(q-1): 
{}".format(phi))
    # Choose e such that 1 < e < phi(n) and e is coprime to phi(n)
    e = random.randrange(1, phi)
    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(1, phi)
        g = gcd(e, phi)
    print("e: "+str(e))
    print("g: "+str(g))
    # Compute d, the modular multiplicative inverse of e mod phi(n)
    d = mod_inverse(e, phi)
    print("d, the modular multiplicative inverse of e mod phi(n): "+str(d))
    # Public key (e, n) and private key (d, n)
    return ((e, n), (d, n))
```

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

def encrypt(public_key, plaintext):
    e, n = public_key
    # Convert plaintext to integer
    plaintext_bytes = plaintext.encode('utf-8')
    plaintext_int = int.from_bytes(plaintext_bytes, byteorder='big')
    # Encrypt plaintext
    ciphertext = pow(plaintext_int, e, n)
    return ciphertext

def decrypt(private_key, ciphertext):
    d, n = private_key
    # Decrypt ciphertext
    plaintext_int = pow(ciphertext, d, n)
    # Convert decrypted integer to plaintext
    plaintext_bytes = plaintext_int.to_bytes((plaintext_int.bit_length() + 7) //
8, byteorder='big')
    plaintext = plaintext_bytes.decode('utf-8')
    return plaintext

# Main function to demonstrate RSA encryption and decryption
def main():
    message = "Hello, this is a test message!"

    # Generate RSA key pairs
    public_key, private_key = generate_key_pair()

    print("Public Key: ", public_key)
    print("Private Key: ", private_key)

    # Encrypt the message
    encrypted_msg = encrypt(public_key, message)
    print("Encrypted Message: ", encrypted_msg)

    # Decrypt the message
    decrypted_msg = decrypt(private_key, encrypted_msg)
    print("Decrypted Message: ", decrypted_msg)

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

<u>Output</u>

```
Message:  Hello, this is a test message!

These are the two prime numbers:
 p: 249091004588924034284638637698860294877
and q: 239980046732351671006662739744463288213

This is n: 59776870921858414281559970505029135133895413966234383624990806909638218384801

This is phi, the Euler's totient function phi(n) = (p-1)(q-1): 5977687092185841428155997050502913513340634291491310791969950553219
4894801712

e: 11230431259114372427152710659873968231865175568657642878982366899036839173939

g: 1

d, the modular multiplicative inverse of e mod phi(n): 397995155451117493836986282536908783543147507530930005065892081426257464165
71

Public Key:  (11230431259114372427152710659873968231865175568657642878982366899036839173939, 5977687092185841428155997050502913513
3895413966234383624990806909638218384801)

Private Key:  (39799515545111749383698628253690878354314750753093000506589208142625746416571, 597768709218584142815599705050291351
33895413966234383624990806909638218384801)

Encrypted Message:  6402664020021498090105512596316219309510921378615361870436442953773585157794

Decrypted Message:  Hello, this is a test message!
```