Lab Assignment No.01

Lab Assignment No.01 RSA

Lab questions:

q1) RSA algorithm:

- is a public_key encryption technique used to securely transmit data
- Asymmetric key Encryption based on prime factorization it involves using a key pair Public key for encryption, Private key for decryption

*Steps of RSA algorithm

① Choose ~~two~~ two prime numbers p and q

② Compute modulus (n) $n = q * p$

③ Compute $(\phi n) = (p-1)(q-1)$

④ Choose the public exponent (e) gcd (e, $\phi(n)$)

⑤ calculate $d = (d \times e) \bmod \text{phi}(n) = 1$

⑥ public key = (e,n), private key = (d,n)

⑦ Encryt: $c = m^e \bmod n$

Decrypt: $m = c^d \bmod n$.

Lab Assignment No.01

Q2) allows performing mathematical operations on encrypted data such that the decrypted result result equals the operation's result on the original plaintexts

---

Q3) multiplying two encrypted values is equivalent to encrypting the multiplication of the two original messages
$*E(m_1) \times E(m_2) \bmod n = E(m_1 \times m_2) \bmod n$

Q4) Advantages:
① Privacy-Preserving computation
② Secure outsourcing

Risks:
① Performance and complexity are slow and complex
② Security exploitation

Q5) ① to Keep Numbers Bounded
② to create a finite group

Q6) $(p=3, q=11, e=7, m_1=2, m_2=5)$

$n = q \times p = 3 \times 11 = 33$

$\phi(n) = (q-1)(p-1) = 2 \times 10 = 20$

$*d \Rightarrow (d \times e) \bmod \phi(n) = 1 \Rightarrow d = 3$

$*E(m) = m^e \bmod n$

- $E(m_1) \Rightarrow E(2) = 2^7 \bmod 33 = 29$ ✱

$\Rightarrow E(5) = 5^7 \bmod 33 = 14$ ✱

$d(C) = C^d \bmod n$

$d(29) = 29^3 \bmod 33 = 2$ ✱

$d(14) = 14^3 \bmod 33 = 5$ ✱

Q7) to Difficlty of factoring and Brute_force resistance

Q8) then e has no modulor $\phi(n)$, So you connot computed, so dcryption fails and RSA won't work,

---

Q9) Secure cloud computation

```python
# RSA Homomorphic Property Demonstration
# ---------------------------------------

# Step 1: Helper functions
def gcd(a, b):
    """Compute the greatest common divisor."""
    while b:
        a, b = b, a % b
    return a

def choose_e(phi):
    """Choose an 'e' that is coprime with phi."""
    for e in range(3, phi):
        if gcd(e, phi) == 1:
            return e
    return None

# Step 2: Key generation
p = 11
q = 17
n = p * q
phi = (p - 1) * (q - 1)

e = choose_e(phi)
if e is None:
    raise ValueError("No suitable e found!")

# Compute private key d (modular inverse of e mod phi)
d = pow(e, -1, phi)

print("---- RSA Key Generation ----")
print(f"p = {p}")
print(f"q = {q}")
print(f"n = {n}")
print(f"ϕ(n) = {phi}")
print(f"Public Key (e, n) = ({e}, {n})")
print(f"Private Key (d, n) = ({d}, {n})")

# Step 3: Encryption and Decryption Functions
def encrypt(m, e, n):
    """Encrypt message m using public key (e, n)."""
    return pow(m, e, n)

def decrypt(c, d, n):
    """Decrypt ciphertext c using private key (d, n)."""
    return pow(c, d, n)

# Step 4: Choose two plaintext messages
m1 = 9
m2 = 12
print("\n---- Plaintexts ----")
print(f"m1 = {m1}, m2 = {m2}")

# Step 5: Encrypt both messages
E_m1 = encrypt(m1, e, n)
E_m2 = encrypt(m2, e, n)
print("\n---- Encryption ----")
print(f"E(m1) = {E_m1}")
print(f"E(m2) = {E_m2}")

# Step 6: Demonstrate Homomorphic Property
# (E(m1) * E(m2)) mod n = E(m1 * m2 mod n)
product_cipher = (E_m1 * E_m2) % n
decrypted_result = decrypt(product_cipher, d, n)
expected_result = (m1 * m2) % n

print("\n---- Homomorphic Property ----")
print(f"(E(m1) * E(m2)) mod n = {product_cipher}")
print(f"Decrypted result = {decrypted_result}")
print(f"Expected (m1 * m2 mod n) = {expected_result}")

# Step 7: Verify
if decrypted_result == expected_result:
    print("\n Homomorphic property verified successfully!")
else:
    print("\n Homomorphic property failed!")
```

```
---- RSA Key Generation ----
p = 11
q = 17
n = 187
ϕ(n) = 160
Public Key (e, n) = (3, 187)
Private Key (d, n) = (107, 187)

---- Plaintexts ----
m1 = 9, m2 = 12

---- Encryption ----
E(m1) = 168
E(m2) = 45

---- Homomorphic Property ----
(E(m1) * E(m2)) mod n = 80
Decrypted result = 108
Expected (m1 * m2 mod n) = 108

 Homomorphic property verified successfully!
```