KEY GENERATION
p, q (large primes)
N = p * q
DECRYYPTION
ENCRYPTION
CIPHERPTION
C = M² mod N
M = Cᵈ mod N
PUBLIC KEY (N,e)
PRIVATE KEY (N, d)
CIPHERTEXT (C)
MESSAGE (M)

IS [40] الاسم : عبدالله خليل كرامه العامري
[STIIS] 25/11/11

Answers for Lab Questions:

* Q1:

Asymmetric Key Encryption based on prime factorization it involves using a key pair [Public key for encryption, Private key for decryption].
Steps:
1- Choose primes p, q.
2- Compute $n = pq$, $\phi(n) = (p-1)(q-1)$.
3- Pick e such $\gcd(e, \phi(n)) = 1$.
4- Find d where $\cancel{e}d = e^{-1} (\mod \phi(n))$.
5- Public key $= (n, e)$, Private key $= (n, d)$.
Encryt: $C = m^e \mod n$; Decrypt: $m = C^d \mod n$.

* Q2:

A property that allwos performing operations on ciphertext that correspond to operations on plaintext without decrypting.

* Q3:

Because multiplying two encrypted values is equivalent to encrypting the multiplication of the two original messages.
Mathematically:
$E(m_1) \times E(m_2) \mod n = E(m_1 \times m_2) \mod n$.

* Q4:

- Advantages: privacy-preserving computation, secure data processing, Cloud privacy.
- Risks: Performance overhead (slow), complex (noise mangement), limited operations, possible leakage or misuse.

* Q5:

Keeps numbers bounded, forms a finite group, and links encryption/decryption to number-theory properties of $n = pq$.

Q6:

Ex: $P=3, q=11, e=7, m_1=2, m_2=5, n=33, \phi=20$

$d=3$ * $Enc(m) = m^e \bmod n$

- $Enc(2) = 2^7 \bmod 33 = 29$.
- $Enc(5) = 5^7 \bmod 33 = 14$.

$29 * 14 \bmod 33 = 10 = Enc(5*2) \bmod 33$

- Decrypt: [$Dec(c) = c^d \bmod n$]

$dec(29) = 29^3 \bmod 33 = 2$ #

$dec(14) = 14^3 \bmod 33 = 5$ #

Q7:

Because larger keys make factoring(n) harder so stronger security. Small keys can be broken easily.

Q8:

Then d doesn't exist, so dcryption fails and RSA won't work.

Q9:

used in Privacy-Preserving Machine Learning (PPML) Cloud computing so servers Process encrypted data without seeing it.

```python
# Step 1: Generate key pair
p = 17          # Small prime
q = 23          # Small prime
n = p * q       # n = 391
phi = (p-1)*(q-1)   # 16 * 22 = 352

# Choose e such that 1 < e < phi and gcd(e, phi) = 1
e = 3
while math.gcd(e, phi) != 1:
    e += 2

d = modinv(e, phi)

print(f"Step 1: Key Generation")
print(f"  Primes: p = {p}, q = {q}")
print(f"  n = p*q = {n}")
print(f"  Euler's totient ϕ(n) = {phi}")
print(f"  Public exponent e = {e}")
print(f"  Private exponent d = {d}")
print("-"*40)
```

```python
# Step 2: Encryption/Decryption functions
def encrypt(m, e, n):
    return pow(m, e, n)

def decrypt(c, d, n):
    return pow(c, d, n)

# Step 3: Encrypt two messages m1 and m2
m1 = 42
m2 = 99

c1 = encrypt(m1, e, n)
c2 = encrypt(m2, e, n)

print(f"Step 3: Encrypt messages")
print(f"  m1 = {m1}, E(m1) = {c1}")
print(f"  m2 = {m2}, E(m2) = {c2}")
print("-"*40)
```

```python
# Step 4b: Ciphertext multiplication mod n
c_mult = (c1 * c2) % n

print(f"Step 4: Homomorphic Multiplication")
print(f"  E(m1) * E(m2) mod n = {c_mult}")
print("-"*40)

# Step 4c: Decrypt the multiplied ciphertext
m_product = decrypt(c_mult, d, n)
expected = (m1 * m2) % n

print(f"Step 5: Decrypt product ciphertext")
print(f"  Decrypted result: {m_product}")
print(f"  Expected (m1 * m2) mod n = {expected}")
print(f"  {'\nHomomorphic property verified: E(m1)E(m2) ≡ E(m1*m2) mod n' i
```

```
Step 1: Key Generation
  Primes: p = 17, q = 23
  n = p*q = 391
  Euler's totient ϕ(n) = 352
  Public exponent e = 3
  Private exponent d = 235
----------------------------------------
Step 3: Encrypt messages
  m1 = 42, E(m1) = 189
  m2 = 99, E(m2) = 228
----------------------------------------
Step 4: Homomorphic Multiplication
  E(m1) * E(m2) mod n = 82
----------------------------------------
Step 5: Decrypt product ciphertext
  Decrypted result: 248
  Expected (m1 * m2) mod n = 248

Homomorphic property verified: E(m1)E(m2) ≡ E(m1*m2) mod n
```