| | | | |
|---|---|---|---|
| Course Name: | Applied Cryptography | Semester: | V |
| Date of Performance: | __10_ / _10__ / __25____ | DIV/ Batch No: | D-2 |
| Student Name: | Shreyans Tatiya | Roll No: | 16010123325 |

## Experiment No: 7

**Title: Implementation of Homomorphic Encryption**

## Aim and Objective of the Experiment:

- Study and implementation of multiplicative homomorphic encryption- using homomorphic property of RSA.

## COs to be achieved:

**CO5: Explore and investigate advances in the field of cryptography**

## Books/ Journals/ Websites referred:

1. https://www.geeksforgeeks.org/ethical-hacking/homomorphic-encryption/

## Theory:

Definition of homomorphism

A **homomorphism** is a structure-preserving mapping between two algebraic structures (like groups, rings, etc.).
It keeps the operation of the structure the same after mapping.

**Example:**
If $f:G1 \rightarrow G2$ is a homomorphism between two groups, then

$f(a*b)=f(a)*f(b)$ for all $a,b \in G1$

List and Explain Types of homomorphism

1. **Monomorphism:**
   - A *one-to-one* (injective) homomorphism.
   - Example: No two different elements in the first structure map to the same element in the second.
2. **Epimorphism:**
   - An *onto* (surjective) homomorphism.

- Every element in the target structure has a preimage.

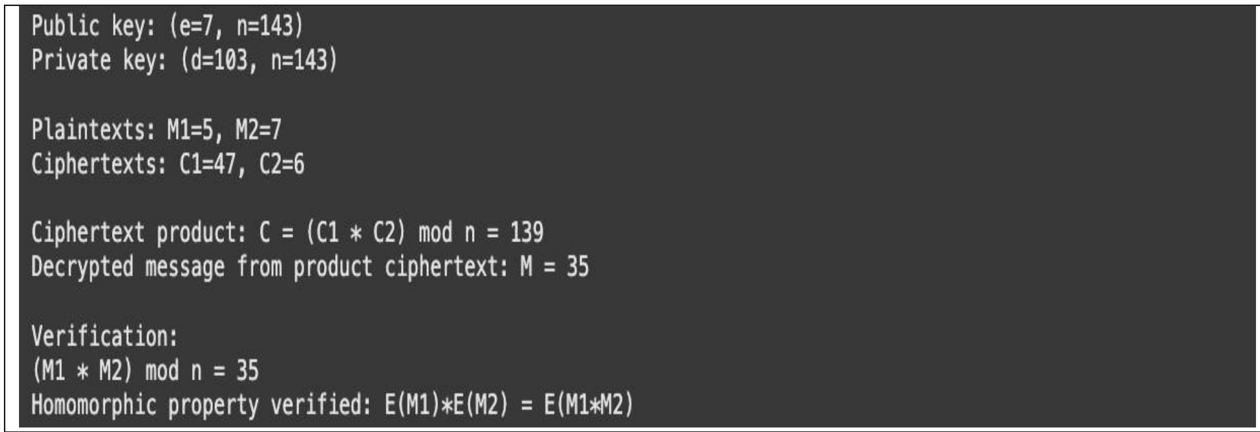
3. **Isomorphism:**
    - *One-to-one and onto* homomorphism (both injective and surjective).
    - Shows the two structures are essentially the same.
4. **Automorphism:**
    - An *isomorphism from a structure to itself*.
    - It's both bijective and structure-preserving.

## Code :

```
# RSA Encryption-Decryption and Homomorphic Property Demonstration

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

# Function to compute modular inverse (for finding d)
def mod_inverse(e, phi):
    for d in range(1, phi):
        if (e * d) % phi == 1:
            return d
    return None

# Function for modular exponentiation
def mod_pow(base, exp, mod):
    result = 1
    base = base % mod
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        exp = exp // 2
        base = (base * base) % mod
    return result

# RSA Key Generation (small values for demonstration)
p = 11
q = 13
n = p * q                   # n = 143
phi = (p - 1) * (q - 1)  # phi = 120
e = 7                       # Choose e coprime to phi
```

```python
d = mod_inverse(e, phi) # Compute private key
print(f"Public key: (e={e}, n={n})")
print(f"Private key: (d={d}, n={n})")

# Step 1: Choose plaintext messages
M1 = 5
M2 = 7
print(f"\nPlaintexts: M1={M1}, M2={M2}")

# Step 2: Encrypt messages
C1 = mod_pow(M1, e, n)
C2 = mod_pow(M2, e, n)
print(f"Ciphertexts: C1={C1}, C2={C2}")

# Step 3: Multiply ciphertexts
C = (C1 * C2) % n
print(f"\nCiphertext product: C = (C1 * C2) mod n = {C}")

# Step 4: Decrypt the product
M = mod_pow(C, d, n)
print(f"Decrypted message from product ciphertext: M = {M}")

# Verification of homomorphic property
print("\nVerification:")
print(f"(M1 * M2) mod n = {(M1 * M2) % n}")
if M == (M1 * M2) % n:
    print(" Homomorphic property verified: E(M1)*E(M2) = E(M1*M2)")
else:
    print("Property not satisfied.")
```

**Output:**



```
Public key: (e=7, n=143)
Private key: (d=103, n=143)

Plaintexts: M1=5, M2=7
Ciphertexts: C1=47, C2=6

Ciphertext product: C = (C1 * C2) mod n = 139
Decrypted message from product ciphertext: M = 35

Verification:
(M1 * M2) mod n = 35
Homomorphic property verified: E(M1)*E(M2) = E(M1*M2)
```

## Post Lab Subjective/Objective type Questions:

List and discuss the alternate privacy preserving methods other than HME

1. **Data Anonymization:**
   Removes personal details (like name, address) so no one can identify a person.
2. **Data Masking:**
   Replaces real data with fake but similar data (e.g., hiding parts of credit card numbers).
3. **Differential Privacy:**
   Adds small random noise to data so individual information can't be guessed.
4. **Secure Multi-Party Computation (SMPC):**
   Many parties work together to compute results without sharing their private data.
5. **Federated Learning:**
   Data stays on user devices; only model updates are shared to train a common model.
6. **Data Perturbation:**
   Slightly changes data values (like adding random numbers) before sharing.
7. **k-Anonymity:**
   Groups people so that each person looks like at least *k–1* others in the dataset.

## Conclusion:

The experiment successfully demonstrates **RSA encryption and decryption** along with its **multiplicative homomorphic property**. It shows that multiplying two ciphertexts gives a result equivalent to encrypting the product of the original plaintexts, proving that RSA supports homomorphic multiplication while keeping data secure.