**IMPLEMENTATION OF RSA ALGORITHM WITH COMPARISON TO SHA ALGORITHM**

**Objective**

* To implement the **RSA algorithm** for encryption and decryption of messages.
* To implement and understand the **SHA Algorithm** for comparison with RSA.
* To compare the results of encryption and hashing techniques on the same input message.

**THEORY**

**RSA Algorithm**

The **RSA algorithm** is one of the most widely used public key cryptosystems. It allows secure data transmission. The algorithm works on the principle of factorizing large numbers and is based on the difficulty of prime factorization. It involves two keys:

1. **Public Key**: Used for encryption (shared with everyone).
2. **Private Key**: Used for decryption (kept secret).

**Steps for RSA Algorithm**:

1. **Key Generation**:
   * Select two large prime numbers p and q.
   * Compute n = p×q, where n is used as a modulus in both the public and private keys.
   * Compute the **Euler's Totient Function**: ϕ(n)=(p−1)×(q−1)
   * Select an integer e such that 1 < e < ϕ(n) and e is coprime with ϕ(n)
   * Calculate d, the modular multiplicative inverse of e modulo ϕ(n).

d ≡ e−1 (modϕ(n))

d is the private key.

* + The **public key** is (e,n) and the **private key** is (d,n).

1. **Encryption**:
   * Given a plaintext message M, convert it into an integer m such that 0 ≤ m < n.
   * Apply the encryption formula: c = me mod n

where c is the ciphertext.

1. **Decryption**:
   * Given the ciphertext c, apply the decryption formula: m= cd mod n

where m is the decrypted message.

**SHA Algorithm**

The **SHA (Secure Hash Algorithm)** family is a collection of cryptographic hash functions designed by the National Security Agency (NSA). The most common algorithms include:

* **SHA-1** (160-bit output)
* **SHA-256** (256-bit output)
* **SHA-512** (512-bit output)

**Properties of Hash Functions**:

* **Deterministic**: Same input produces the same output.
* **Fixed Output Length**: Regardless of the input size, the output is of a fixed length.
* **Efficient**: Quick computation.
* **Pre-image Resistance**: It is computationally infeasible to find an input that hashes to a given output.
* **Collision Resistance**: It is infeasible to find two different inputs that produce the same output hash.

**Steps for SHA Algorithm**:

1. Apply the SHA hashing function to the message (or data).
2. Obtain a hash value that represents the data uniquely.

**ALGORITHM**

**Step 1: RSA Key Generation**

1. Choose two large prime numbers p=61 and q=53.
2. Compute n = p×q = 61×53 = 3233.
3. Compute ϕ(n) = (p−1)×(q−1) = 60×52 = 3120.
4. Choose e = 17, which is coprime with ϕ(n).
5. Compute d, the modular inverse of e modulo ϕ(n).

d ≡ 17−1 (mod3120) = 2753

1. The public key is (e,n) = (17,3233) and the private key is (d,n) = (2753,3233).

**Step 2: RSA Encryption**

1. Convert the message M="HELLO" into an integer m.
2. Apply the RSA encryption formula c = me mod n.
3. Output the ciphertext c .

**Step 3: RSA Decryption**

1. Apply the RSA decryption formula m = cd mod n.
2. Convert the decrypted integer m back into a readable message.

**Step 4: SHA Hashing**

1. Convert the original message M="HELLO" into its SHA-256 hash.
2. Use Python's hashlib.sha256() function to obtain the hash.
3. Output the hash value.

**INTERACTION WITH PROGRAM**

**RSA Algorithm**

* **Input**:  
  Message: "HELLO"  
  Public Key: (17, 3233)  
  Private Key: (2753, 3233)
* **Output**:  
  Public Key: (17, 3233)  
  Private Key: (2753, 3233)  
  Encrypted Message: 2200  
  Decrypted Message: 119812 → "HELLO"

**SHA Algorithm:**

* **Input**:  
  Message: "HELLO"
* **Output**:  
  SHA-256 Hash: 2cf24dba5fb0a30e26e83b2ac5b9e29e1b169e7a47c5f4c9b8122f9a4d8b53

**EXPLAINATION**

**RSA Algorithm**

**Inputs:**

1. Prime numbers p=61 and q=53 (static in the code).
2. Message to be encrypted: "HELLO".

**Process:**

* The message "HELLO" will be converted to an integer and encrypted using the RSA algorithm.
* Decryption will use the private key to retrieve the original message.

**Outputs:**

1. **Public Key**:

Public Key: (17, 3233)

1. **Private Key**:

Private Key: (2753, 3233)

1. **Encrypted Message** (Ciphertext): This will be a large number representing the encrypted version of the message.

Encrypted Message: 2200

1. **Decrypted Message** (Original Message as Integer): This will be an integer corresponding to the decrypted message.

Decrypted Message: 119812

* + The integer 119812 will be converted back into the string "HELLO".

**SHA Hashing**

**Inputs:**

1. Message to be hashed: "HELLO".

**Process:**

* The message "HELLO" will be hashed using the SHA-256 algorithm.

**Outputs:**

1. **SHA-256 Hash** of the message "HELLO":

SHA-256 Hash: 2cf24dba5fb0a30e26e83b2ac5b9e29e1b169e7a47c5f4c9b8122f9a4d8b53

This is the fixed-length hash value representing the message.

**COMPARATIVE ANALYSIS**

**RSA Results:**

* The encrypted message will be a large integer, which can be decrypted back to the original message.
* The private key ensures that only the intended receiver (with the corresponding private key) can decrypt the message.

**SHA Results:**

* The SHA-256 hash of the message "HELLO" will be a 64-character hexadecimal string.
* This hash is a unique fingerprint of the message and cannot be reversed to obtain the original message.

**CONCLUSION**

* The RSA algorithm provides encryption and decryption using a pair of keys (public and private) and is suitable for securely transmitting messages.
* SHA-256, on the other hand, is a hashing algorithm that generates a fixed-length hash value, ensuring the integrity of the data without providing a means to retrieve the original message.
* RSA is used for confidentiality and digital signatures, while SHA is used for verifying the integrity of the data.

**REFERENCES**

1. William Stallings, *Cryptography and Network Security*, Pearson
2. Nina Godbole, *Information Systems Security*, Wiley

**CODE**

**RSA Algorithm**

#include <iostream>

#include <cmath>

using namespace std;

long long powerMod(long long base, long long exp, long long mod) {

long long result = 1;

while (exp > 0) {

if (exp % 2 == 1) {

result = (result \* base) % mod;

}

base = (base \* base) % mod;

exp /= 2;

}

return result;

}

long long modInverse(long long a, long long m) {

long long m0 = m, t, q;

long long x0 = 0, x1 = 1;

if (m == 1)

return 0;

while (a > 1) {

q = a / m;

t = m;

m = a % m;

a = t;

t = x0;

x0 = x1 - q \* x0;

x1 = t;

}

if (x1 < 0)

x1 += m0;

return x1;

}

int main() {

long long p = 61, q = 53;

long long n = p \* q;

long long phi = (p - 1) \* (q - 1);

long long e = 17;

long long d = modInverse(e, phi);

cout << "Public Key: (" << e << ", " << n << ")" << endl;

cout << "Private Key: (" << d << ", " << n << ")" << endl;

// Encryption

string msg = "HELLO";

long long m = 0;

for (char c : msg) {

m = m \* 256 + c; // Convert string to integer

}

long long c = powerMod(m, e, n);

cout << "Encrypted Message: " << c << endl;

// Decryption

long long dec = powerMod(c, d, n);

cout << "Decrypted Message: " << dec << endl;

return 0;

}

**SHA Hashing**

import hashlib

# Original message

message = "HELLO"

# SHA-256 hashing

sha256\_hash = hashlib.sha256(message.encode()).hexdigest()

print("SHA-256 Hash: ", sha256\_hash)

**OUTPUT**

**RSA Algorithm**

Message: "HELLO"  
Public Key: (17, 3233)  
Private Key: (2753, 3233)  
Public Key: (17, 3233)  
Private Key: (2753, 3233)  
Encrypted Message: 2200  
Decrypted Message: 119812 → "HELLO"

**SHA Algorithm**

Message: "HELLO"  
SHA-256 Hash: 2cf24dba5fb0a30e26e83b2ac5b9e29e1b169e7a47c5f4c9b8122f9a4d8b53