## Name: **Ketan Kunkalikar**

## Roll: **21CSE1016**

## Lab Subject: **Security Lab**

## Topic: **Diffie-Hellman and RSA algorithm**

## Instructor: **Dr. Meenakshi Panda**

## Date: **23th October 2024**

## **Q1. What is the Diffie-Hellman algorithm? How does it work?**

## Diffie-Hellman is a method for securely exchanging cryptographic keys over a public channel, allowing two parties to establish a shared secret without ever transmitting the actual secret.

## **Main points:**

## 1. Key Type: Generates symmetric keys for both parties

## 2. Security Basis: Based on the discrete logarithm problem

## 3. Parameters: Uses a prime modulus (p) and a generator (g)

## 4. Communication: Requires exchange of public values only

## **Key Components of Diffie-Hellman:**

## 1. System Parameters:

## - Prime modulus (p): A large prime number

## - Generator (g): A primitive root modulo p

## - Both are public and can be reused

## 2. Key Generation:

## private\_key = generate\_private\_key(2, p-2);

## 

## public\_key = mod\_pow(g, private\_key, p);

## 

## 3. Key Exchange Process:

## - Each party generates their private key

## - Each party calculates their public key using g^(private\_key) mod p

## - Parties exchange public keys

## - Each party calculates the shared secret using (other\_public\_key)^(private\_key) mod p

## The security of Diffie-Hellman relies on the computational difficulty of the discrete logarithm problem. Even if an attacker intercepts the public values (g^a mod p and g^b mod p), they cannot feasibly determine the shared secret (g^(ab) mod p) without knowing either private key.

// Q1. IMPLEMENT THE DIFFIE-HELLMAN KEY EXCHANGE MECHANISM.

#include <iostream>

#include <cmath>

#include <random>

#include <iomanip>

using namespace std;

// Function for modular exponentiation (a^b mod p)

long long mod\_pow(long long base, long long exponent, long long modulus) {

    cout << "\nCalculating: " << base << "^" << exponent << " mod " << modulus << endl;

    long long result = 1;

    base = base % modulus;

    while (exponent > 0) {

        // If exponent is odd, multiply base with result

        if (exponent & 1)

            result = (result \* base) % modulus;

        // Exponent must be even now

        base = (base \* base) % modulus;

        exponent = exponent >> 1; // Divide exponent by 2

    }

    cout << "Result: " << result << endl;

    return result;

}

// Generate a random private key

long long generate\_private\_key(long long min\_val, long long max\_val) {

    random\_device rd;

    mt19937 gen(rd());

    uniform\_int\_distribution<long long> dis(min\_val, max\_val);

    return dis(gen);

}

class DiffieHellman {

private:

    long long p; // Prime modulus

    long long g; // Generator

    long long private\_key;

    long long public\_key;

    long long shared\_secret;

public:

    // Constructor

    DiffieHellman(long long prime, long long generator) : p(prime), g(generator) {

        cout << "\nInitializing Diffie-Hellman with:" << endl;

        cout << "Prime (p): " << p << endl;

        cout << "Generator (g): " << g << endl;

    }

    void generate\_keys() {

        // Generate private key

        private\_key = generate\_private\_key(2, p-2);

        cout << "\nGenerated private key: " << private\_key << endl;

        // Calculate public key: g^private\_key mod p

        cout << "\nCalculating public key (g^private\_key mod p):" << endl;

        public\_key = mod\_pow(g, private\_key, p);

        cout << "Generated public key: " << public\_key << endl;

    }

    long long get\_public\_key() const {

        return public\_key;

    }

    void calculate\_shared\_secret(long long other\_public\_key) {

        cout << "\nCalculating shared secret with received public key: " << other\_public\_key << endl;

        shared\_secret = mod\_pow(other\_public\_key, private\_key, p);

        cout << "Generated shared secret: " << shared\_secret << endl;

    }

    // Get shared secret

    long long get\_shared\_secret() const {

        return shared\_secret;

    }

};

int main() {

    long long p = 23;        // Prime modulus

    long long g = 5;         // Generator

    cout << "System Parameters: ";

    cout << "Prime (p): " << p << "; ";

    cout << "Generator (g): " << g << endl;

    // Create instances for Alice and Bob

    cout << "=== Alice's Setup ===" << endl;

    DiffieHellman alice(p, g);

    alice.generate\_keys();

    cout << "=== Bob's Setup ===" << endl;

    DiffieHellman bob(p, g);

    bob.generate\_keys();

    // Exchange public keys and calculate shared secret

    cout << "=== Key Exchange ===" << endl;

    cout << "Alice's public key: " << alice.get\_public\_key() << endl;

    cout << "Bob's public key: " << bob.get\_public\_key() << endl;

    cout << "=== Alice calculates shared secret ===" << endl;

    alice.calculate\_shared\_secret(bob.get\_public\_key());

    cout << "=== Bob calculates shared secret ===" << endl;

    bob.calculate\_shared\_secret(alice.get\_public\_key());

    // Verify that both parties have the same shared secret

    cout << "=== Verification ===" << endl;

    cout << "Alice's shared secret: " << alice.get\_shared\_secret() << endl;

    cout << "Bob's shared secret: " << bob.get\_shared\_secret() << endl;

    if (alice.get\_shared\_secret() == bob.get\_shared\_secret()) {

        cout << "Success! Shared secrets match." << endl;

    } else {

        cout << "Error! Shared secrets do not match." << endl;

    }

    // Show the mathematical steps

    cout << "=== Steps ===" << endl;

    cout << "1. Alice and Bob agree on p = " << p << " and g = " << g << endl;

    cout << "2. Alice chooses private key a, calculates A = g^a mod p" << endl;

    cout << "3. Bob chooses private key b, calculates B = g^b mod p" << endl;

    cout << "4. Alice calculates shared secret = B^a mod p" << endl;

    cout << "5. Bob calculates shared secret = A^b mod p" << endl;

    cout << "Both arrive at g^(ab) mod p" << endl;

    return 0;

}

## **OUTPUT:**

## **ketan@Arondight MINGW64 ~/Desktop/sem 7/Security Lab/09\_23\_oct\_absent**

## **$ g++ q1.cpp -o q1.exe && ./q1.exe**

## **System Parameters: Prime (p): 23; Generator (g): 5**

## **=== Alice's Setup ===**

## **Initializing Diffie-Hellman with:**

## **Prime (p): 23**

## **Generator (g): 5**

## **Generated private key: 13**

## **Calculating public key (g^private\_key mod p):**

## **Calculating: 5^13 mod 23**

## **Result: 21**

## **Generated public key: 21**

## **=== Bob's Setup ===**

## **Initializing Diffie-Hellman with:**

## **Prime (p): 23**

## **Generated private key: 13**

## **Calculating public key (g^private\_key mod p):**

## **Calculating: 5^13 mod 23**

## **Result: 21**

## **Generated public key: 21**

## **=== Key Exchange ===**

## **Alice's public key: 21**

## **Bob's public key: 21**

## **=== Alice calculates shared secret ===**

## **Calculating shared secret with received public key: 21**

## **Calculating: 21^13 mod 23**

## **Result: 19**

## **Generated shared secret: 19**

## **=== Bob calculates shared secret ===**

## **Calculating shared secret with received public key: 21**

## **Calculating: 21^13 mod 23**

## **Result: 19**

## **Generated shared secret: 19**

## **=== Verification ===**

## **Alice's shared secret: 19**

## **Bob's shared secret: 19**

## **Success! Shared secrets match.**

## **=== Steps ===**

## **1. Alice and Bob agree on p = 23 and g = 5**

## **2. Alice chooses private key a, calculates A = g^a mod p**

## **3. Bob chooses private key b, calculates B = g^b mod p**

## **4. Alice calculates shared secret = B^a mod p**

## **5. Bob calculates shared secret = A^b mod p**

## **Both arrive at g^(ab) mod p**

## **ketan@Arondight MINGW64 ~/Desktop/sem 7/Security Lab/09\_23\_oct\_absent**

## **Q2. What is the RSA algorithm?**

## RSA is an asymmetric cryptographic algorithm that uses a pair of keys (public and private) for encryption and decryption.

## **Main points:**

## 1. Key Generation: Uses two large prime numbers to create public and private keys

## 2. Block Size: Can handle variable-length data blocks (smaller than n)

## 3. Key Size: Typically 1024 to 4096 bits in practice (though smaller in your example for demonstration)

## 4. Mathematical Basis: Based on the practical difficulty of factoring the product of two large prime numbers

## **Key Components of RSA:**

## 1. Key Generation:

## - Choose two distinct prime numbers (p, q)

## - Calculate modulus n = p × q

## - Calculate Euler's totient φ(n) = (p-1) × (q-1)

## - Choose public exponent e (coprime to φ(n))

## - Calculate private exponent d (modular multiplicative inverse of e mod φ(n))

## 2. Public Key:

## - Consists of (e, n)

## - Used for encryption

## - Can be freely shared

## 3. Private Key:

## - Consists of (d, n)

## - Used for decryption

## - Must be kept secret

## 4. Operations:

## - Encryption: c = m^e mod n (where m is the message)

## - Decryption: m = c^d mod n (where c is the ciphertext)

## The security of RSA relies on the computational difficulty of factoring the product of two large prime numbers (n = p × q).

// Q2. WRITE A PROGRAM TO IMPLEMENT RSA ALGORITHM.

#include <bits/stdc++.h>

using namespace std;

bool is\_prime(long long n) {

    if (n <= 1) return false;

    if (n <= 3) return true;

    if (n % 2 == 0 || n % 3 == 0) return false;

    for (long long i = 5; i \* i <= n; i += 6) {

        if (n % i == 0 || n % (i + 2) == 0)

            return false;

    }

    return true;

}

long long gcd(long long a, long long b) {

    while (b != 0) {

        long long temp = b;

        b = a % b;

        a = temp;

    }

    return a;

}

long long mod\_inverse(long long a, long long m) {

    long long m0 = m;

    long long y = 0, x = 1;

    if (m == 1)

        return 0;

    while (a > 1) {

        long long q = a / m;

        long long t = m;

        m = a % m;

        a = t;

        t = y;

        y = x - q \* y;

        x = t;

    }

    if (x < 0)

        x += m0;

    return x;

}

long long mod\_pow(long long base, long long exponent, long long modulus) {

    long long result = 1;

    base = base % modulus;

    while (exponent > 0) {

        if (exponent & 1)

            result = (result \* base) % modulus;

        base = (base \* base) % modulus;

        exponent = exponent >> 1;

    }

    return result;

}

class RSA {

private:

    long long p, q;      // Prime numbers

    long long n;         // Modulus n = p \* q

    long long phi;       // Euler's totient = (p-1) \* (q-1)

    long long e;         // Public exponent

    long long d;         // Private exponent

public:

    RSA(long long prime1 = 61, long long prime2 = 53) {

        cout << "Initializing RSA with p=" << prime1 << " and q=" << prime2 << endl;

        p = prime1;

        q = prime2;

        n = p \* q;

        phi = (p - 1) \* (q - 1);

        // Choose e

        e = 17;

        while (gcd(e, phi) != 1) {

            e += 2;

        }

        // Calculate private key d

        d = mod\_inverse(e, phi);

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

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

    }

    long long encrypt(long long message) {

        cout << "Encrypting message: " << message << endl;

        return mod\_pow(message, e, n);

    }

    long long decrypt(long long ciphertext) {

        cout << "Decrypting ciphertext: " << ciphertext << endl;

        return mod\_pow(ciphertext, d, n);

    }

    pair<long long, long long> get\_public\_key() {

        return make\_pair(e, n);

    }

    pair<long long, long long> get\_private\_key() {

        return make\_pair(d, n);

    }

};

int main() {

    RSA rsa(61, 53);

    // Example message

    long long message = 123;

    cout << "Original Message: " << message << endl;

    // Encrypt the message

    long long ciphertext = rsa.encrypt(message);

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

    // Decrypt the message

    long long decrypted = rsa.decrypt(ciphertext);

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

    return 0;

}

## **OUTPUT:**

## 