## Name: **Ketan Kunkalikar**

## Roll: **21CSE1016**

## Lab Subject: **Security Lab**

## Topic: **RSA and Elgamal implementation with Digital Signatures.**

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

## Date: **6th November 2024**

## **What are Digital Signatures? How do they work?**

## Digital signatures are the computer – equivalent of actual signatures, but with cryptographic security benefits. There are three main cryptographic properties:

## 1. **Authentication** - Verifies the identity of the sender/signer

## 2. **Non-repudiation** - The signer cannot later deny having signed the message

## 3. **Integrity** - Ensures the message hasn't been altered since signing

## The signer has a pair of keys: a private key (kept secret) and a public key (shared openly)

## 2. To sign a message:

## - The signer sends the message through a sign generation function (could involve making hashes), to create a signature.

## - The original message and signature are sent together

## 3. To verify a signature:

## - The receiver uses the signer's public key to decrypt the signature

## - They independently hash the received message

## - If the decrypted hash matches their calculated hash, the signature is valid

// RSA with signature

#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;

    }

    // Original encryption/decryption methods

    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);

    }

    // New signature methods

    long long sign(long long message) {

        cout << "Signing message: " << message << endl;

        // In RSA, signing is essentially encrypting with the private key

        long long signature = mod\_pow(message, d, n);

        cout << "Generated signature: " << signature << endl;

        return signature;

    }

    bool verify(long long message, long long signature) {

        cout << "Verifying signature for message: " << message << endl;

        cout << "With signature: " << signature << endl;

        // Log the public key used for verification

        cout << "Using public key (e, n): (" << e << ", " << n << ")" << endl;

        // In RSA, verification is done by "decrypting" the signature with public key

        long long decoded = mod\_pow(signature, e, n);

        // Log the decoded message

        cout << "Decoded message from signature: " << decoded << endl;

        // Compare the decoded message with the original message

        bool is\_valid = (decoded == message);

        // Log the result of the comparison

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

        cout << "Decoded message: " << decoded << endl;

        cout << "Signature is " << (is\_valid ? "valid" : "invalid") << endl;

        return is\_valid;

    }

    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 << "\n=== Encryption/Decryption Demo ===" << endl;

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

    // Encrypt and decrypt

    long long ciphertext = rsa.encrypt(message);

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

    long long decrypted = rsa.decrypt(ciphertext);

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

    // Digital signature demo

    cout << "\n=== Digital Signature Demo ===" << endl;

    // Sign the message

    long long signature = rsa.sign(message);

    cout << "Generated signature: " << signature << endl;

    // Verify the signature

    bool is\_valid = rsa.verify(message, signature);

    // Try verifying with a tampered message

    cout << "\n=== Tampering Demo ===" << endl;

    long long tampered\_message = message + 1;

    cout << "Trying to verify signature with tampered message: " << tampered\_message << endl;

    bool is\_invalid = rsa.verify(tampered\_message, signature);

    return 0;

}

## **Output:**

## 

// Elgamal with digital signature.

#include <iostream>

#include <cmath>

#include <random>

using namespace std;

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

    if (b == 0) return a;

    return gcd(b, a % b);

}

class ElGamal {

private:

    // System parameters

    long long p;     // Prime number

    long long alpha; // Generator

    // Private key

    long long a;     // Private key

    // Public key

    long long beta;  // Public key (beta = alpha^a mod p)

    long long power\_mod(long long base, long long exp, long long mod) {

        long long result = 1;

        base = base % mod;

        while (exp > 0) {

            if (exp & 1)

                result = (result \* base) % mod;

            base = (base \* base) % mod;

            exp >>= 1;

        }

        return result;

    }

    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;

    }

public:

    ElGamal() {

        cout << "Enter prime number (p): ";

        cin >> p;

        cout << "Enter generator (alpha): ";

        cin >> alpha;

        cout << "Enter private key (a): ";

        cin >> a;

        beta = power\_mod(alpha, a, p);  // Public key beta = alpha^a mod p

        cout << "ElGamal Parameters:\n";

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

        cout << "Generator (alpha): " << alpha << "\n";

        cout << "Private key (a): " << a << "\n";

        cout << "Public key (beta): " << beta << "\n\n";

    }

    pair<long long, long long> encrypt(long long message) {

        long long k = 4;

        long long c1 = power\_mod(alpha, k, p);

        long long c2 = (message \* power\_mod(beta, k, p)) % p;

        cout << "Encrypting message: " << message << "\n";

        cout << "c1: " << c1 << ", c2: " << c2 << "\n";

        return make\_pair(c1, c2);

    }

    long long decrypt(pair<long long, long long> cipher) {

        long long c1 = cipher.first;

        long long c2 = cipher.second;

        long long s = power\_mod(c1, a, p);

        long long s\_inv = power\_mod(s, p-2, p);

        long long m = (c2 \* s\_inv) % p;

        cout << "Decrypting (c1=" << c1 << ", c2=" << c2 << "): " << m << "\n";

        return m;

    }

    // Sign a message

    pair<long long, long long> sign(long long m) {

        long long k = 4;

        while (gcd(k, p-1) != 1) {

            k = (k + 1) % (p-1);

            if (k == 0) k = 1;

        }

        // gamma = alpha^k mod p

        long long gamma = power\_mod(alpha, k, p);

        // k\_inv = k^(-1) mod (p-1)

        long long k\_inv = mod\_inverse(k, p-1);

        // delta = (m - a\*gamma)k^(-1) mod (p-1)

        long long delta = (k\_inv \* (m - a \* gamma)) % (p-1);

        if (delta < 0) delta += (p-1);

        cout << "Signing message: " << m << "\n";

        cout << "Signature components (gamma,delta): (" << gamma << "," << delta << ")\n";

        return make\_pair(gamma, delta);

    }

    // Verify signature

    bool verify(long long m, pair<long long, long long> signature) {

        long long gamma = signature.first;

        long long delta = signature.second;

        cout << "Verifying signature for message: " << m << "\n";

        cout << "With signature (gamma,delta): (" << gamma << "," << delta << ")\n";

        if (gamma <= 0 || gamma >= p) {

            cout << "Verification failed: gamma out of range\n";

            return false;

        }

        // Verify: beta^gamma \* gamma^delta ≡ alpha^m mod p

        long long right = (power\_mod(beta, gamma, p) \* power\_mod(gamma, delta, p)) % p;

        long long left = power\_mod(alpha, m, p);

        cout << "Verification calculation:\n";

        cout << "Left side (alpha^m mod p): " << left << "\n";

        cout << "Right side (beta^gamma \* gamma^delta mod p): " << right << "\n";

        return left == right;

    }

};

int main() {

    ElGamal crypto;

    long long m = 12;

    cout << "Original message: " << m << "\n\n";

    cout << "=== Encryption/Decryption Demo ===\n";

    auto encrypted = crypto.encrypt(m);

    long long decrypted = crypto.decrypt(encrypted);

    cout << "Decrypted message: " << decrypted << "\n\n";

    cout << "=== Digital Signature Demo ===\n";

    auto signature = crypto.sign(m);

    bool is\_valid = crypto.verify(m, signature);

    cout << "Signature is " << (is\_valid ? "valid" : "invalid") << "\n\n";

    // Try verifying with a tampered message

    long long tampered\_m = m + 1;

    cout << "Trying to verify tampered message: " << tampered\_m << "\n";

    bool is\_invalid = crypto.verify(tampered\_m, signature);

    cout << "Tampered signature is " << (is\_invalid ? "valid" : "invalid") << "\n";

    return 0;

}

## **Output:**

## 