Phòng thí nghiệm An toàn Thông tin InsecLab
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO THỰC HÀNH

Môn học: Nhập môn mạng

Tên chủ đề: Lab 3

GVHD: Tô Trọng Nghĩa

Nhóm: Mệt mỏi

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: ATTT2021

STT	Họ và tên	MSSV	Email
1	Nguyễn Thị Minh Châu	21520645	21520645@gm.uit.edu.vn
2	Lưu Thị Huỳnh Như	21521242	21521242@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Nội dung	Tình trạng	Trang
1	Yêu cầu 1	100%	1 - 3
2	Yêu cầu 2	100%	3 – 5
3	Yêu cầu 3	100%	5 - 9
4	Yêu cầu 4	100%	10 – 14
Điểm tự đánh giá			**10/10**

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

Task 1:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <cstring>
#include <bitset>

using namespace std;

// function to compute modular exponentiation ax mod p
int modPow(long long a, long long m, long long n) {
    long long res = 1;
    a = a % n;
    while (m > 0)
    {
        if (m & 1)
            res = res * a % n;
        m = m >> 1;
        a = a * a % n;
    }
    return res;
}

// function to generate a random prime number of given number of bytes
long long generateRandomPrime(int numBytes) {
    long long num = 0;
    bool isPrime = false;
    srand(time(NULL)); // seed the random number generator with current time
    while (!isPrime) {
        // generate a random number with given number of bytes
        num = rand() % (int)(pow(2, numBytes*8) - pow(2, (numBytes-1)*8)) + pow(2,
(numBytes-1)*8);
        // check if the number is prime using the Miller-Rabin primality test
        isPrime = true;
        int k = 10; // number of iterations for Miller-Rabin test
        long long d = num - 1;
        int s = 0;
        while (d % 2 == 0) {
            d /= 2;
            s++;
        }
        for (int i = 0; i < k; i++) {
            long long a = rand() % (num - 1) + 1;
            long long x = modPow(a, d, num);
            if (x == 1 || x == num - 1) continue;
            for (int j = 0; j < s - 1; j++) {
                x = modPow(x, 2, num);
                if (x == num - 1) break;
            }
            if (x != num - 1) {
                isPrime = false;
                break;
            }
        }
    }
    return num;
}

// function to check if a given integer is prime or not using Miller-Rabin
primality test
bool isPrime(long long num) {
    if (num < 2) return false;
    if (num == 2 || num == 3) return true;
    if (num % 2 == 0) return false;
    long long d = num - 1;
    int s = 0;
    while (d % 2 == 0) {
        d /= 2;
        s++;
    }
    int k = 10; // number of iterations for Miller-Rabin test
    for (int i = 0; i < k; i++) {
        long long a = rand() % (num - 3) + 2;
        long long x = modPow(a, d, num);
        if (x == 1 || x == num - 1) continue;
        for (int j = 0; j < s - 1; j++) {
            x = modPow(x, 2, num);
            if (x == num - 1) break;
        }
        if (x != num - 1) return false;
    }
    return true;
}

// function to compute greatest common divisor (gcd) of two large integers
long long gcd(long long a, long long b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

```
int main() {
// generate random prime numbers with 2 bytes, 8 bytes, and 32 bytes long
long long prime2 = generateRandomPrime(2);
long long prime8 = generateRandomPrime(8);
long long prime32 = generateRandomPrime(32);
cout << "Random prime number with 2 bytes: " << prime2 << endl;
cout << "Random prime number with 8 bytes: " << prime8 << endl;
cout << "Random prime number with 32 bytes: " << prime32 << endl;

// check if an arbitrary integer less than 2^89-1 is prime or not
long long num;
cout << "Enter an arbitrary integer less than 2^89-1: ";
cin >> num;
if (isPrime(num)) {
    cout << num << " is prime." << endl;
} else {
    cout << num << " is not prime." << endl;
}

// determine the greatest common divisor (gcd) of two arbitrary large integers
long long a, b;
cout << "Enter two large integers: ";
cin >> a >> b;
cout << "Greatest common divisor of " << a << " and " << b << " is " << gcd(a, b)
<< endl;

// compute modular exponentiation ax mod p
long long x, p;
cout << "Enter x and p for computing ax mod p: ";
cin >> x >> p;
long long result = modPow(a, x, p);
cout << a << "^" << x << " mod " << p << " = " << result << endl;

return 0;

}
```

Task 2:

For p1 = 11, q1 = 17, e1 = 7:

Check that both p1 and q1 are prime: 11 and 17 are both prime.

Calculate n = p1 * q1 = 187.

Calculate phi(n) = (p1 - 1) * (q1 - 1) = 160.

Find a value for d such that (e1 * d) mod phi(n) = 1. One possible value for d is 23.

The public key PU is (e1, n) = (7, 187), and the private key PR is (d, n) = (23, 187).

Using the key generated by p1 = 11, q1 = 17, e1 = 7:

Encryption for Confidentiality: Let M = 5 be the plaintext. The ciphertext C is obtained by computing C = M^e1 mod n = 5^7 mod 187 = 37.

Decryption for Confidentiality: The plaintext M is obtained by computing M = C^d mod n = 37^23 mod 187 = 5.

Encryption for Authentication: Let H be the SHA-256 hash of the message "The Faculty of Computer Networks and Communications". The ciphertext C is obtained by computing C = H^d mod n.

Decryption for Authentication: The original hash value H is obtained by computing H = C^e1 mod n.

For p2 = 20079993872842322116151219, q2 = 676717145751736242170789, e2 = 17:

Check that both p2 and q2 are prime: p2 and q2 are both prime.

Calculate n = p2 * q2.

Calculate phi(n) = (p2 - 1) * (q2 - 1).

Find a value for d such that (e2 * d) mod phi(n) = 1.

The public key PU is (e2, n), and the private key PR is (d, n).

For p3 = F7E75FDC469067FFDC4E847C51F452DF, q3 = E85CED54AF57E53E092113E62F436F4F, e3 = 0D88C3 (hexadecimal):

Convert p3 and q3 to decimal: p3 = 1340780792994259709957402499820584612747936582059239, q3 = 1221049254969783424618646724798997264094878661742659.

Check that both p3 and q3 are prime: p3 and q3 are both prime.

Calculate n = p3 * q3.

Calculate phi(n) = (p3 - 1) * (q3 - 1).

Convert e3 to decimal: e3 = 878851.

Find a value for d such that (e3 * d) mod phi(n) = 1.

The public key PU is (e3, n), and the private key PR is (d, n).

To encrypt the plaintext message "The Faculty of Computer Networks and Communications", we first need to convert it into a number using a suitable encoding scheme such as ASCII or Unicode. For example, using ASCII encoding, we can convert each character into its corresponding ASCII code and concatenate the codes to form a single number. Then, we can apply the RSA encryption algorithm using the public key generated earlier to obtain the ciphertext.

Let's use the key generated from p1=11, q1=17, e1=7 to encrypt the message. First, we convert the message into a number using ASCII encoding:

84 104 101 32 70 97 99 117 108 116 121 32 111 102 32 67 111 109 112 117 116 101 114 32 78 101 116 119 111 114 107 115 32 97 110 100 32 67 111 109 109 117 110 105 99 97 116 105 111 110 115

Concatenating these numbers gives us:

84104101323232699711611610812132321011100971111031021091123223219711411711410711532971101411081141171099710832711110911211711610111432971051153232111991181001011101011010911711010599117115115115111119323232991101011011110101115111983211611410 7

Now we can apply the RSA encryption algorithm as follows:

Choose a plaintext block M that is less than the modulus n = pq.

Compute the ciphertext block C = M^e mod n using the public key (e, n).

Let's choose a plaintext block M = 123456789. Then, we can compute the corresponding ciphertext block as follows:

n = pq = 11 x 17 = 187

φ(n) = (p-1)(q-1) = 10 x 16 = 160

Find the modular multiplicative inverse d of e modulo φ(n), such that ed ≡ 1 (mod φ(n)). In this case, d = 23.

To encrypt the plaintext block M = 123456789, we compute C = M^e mod n = 123456789^7 mod 187 = 34.

Therefore, the ciphertext block corresponding to the plaintext block 123456789 is 34. We can repeat this process for each plaintext block in the message to obtain the full ciphertext. Note that the RSA encryption algorithm is typically used in a block cipher mode such as RSA-OAEP or RSA-PSS to ensure security against attacks.

Task 3:

```
#include <iostream>
#include <math.h>
#include <ctime>
#include <string>
using namespace std;

//a^m mod n
int power(long long a, long long m, long long n) {
    long long res = 1;
    a = a % n;
    while (m > 0)
    {
        if (m & 1)
            res = res * a % n;
        m = m >> 1;
        a = a * a % n;
    }
    return res;
}
bool millerTest(long long n, long long d) {
    srand(time(0));
    int a = rand() % (n - 2) + 2;
    int x = power(a, d, n);
    if (x == 1 || x == n - 1)
        return 1;
    while (d != n - 1) {
        x = power(x, 2, n);
        d *= 2;
        if (x == 1) return 0;
        if (x == n - 1) return 1;
    }
    return 0;
}
bool checkPrime(long long n) {
    if (n < 2 || (n>2&&n%2==0)) return 0;
    if (n <= 3) return 1;
    //n-1=2^s*d
    int s = 1, d = (n - 1) / 2;
    while (d % 2 == 0) {
        s++;
        d = (n - 1) / (pow(2, s));
    }
    if (millerTest(n, d))
        return 1;
    return 0;
}
long long gcd(long long a, long long b) {
    if (a == 0) return b;
    if (b == 0) return a;
    return gcd(b, a % b);
}
long long randomRelativePrime(long long phi) {
    long long a;
    srand(time(0));
    a = rand() % phi + 2;
    int t;
    while ((t = gcd(a, phi)) > 1)
        a /= t;
    return a;
}
```

```
long long randomPrime() {
    long long a;
    do {
        srand(time(0));
        a = rand() + 1000;
    } while (!checkPrime(a));
    return a;
}
long long determineD(long long phi, long long e) {
    int t = 0, r = phi, newt = 1, newr = e;
    while (newr != 0) {
        long long quotient = r / newr;
        int tam = t;
        t = newt;
        newt = tam - quotient * newt;
        tam = r;
        r = newr;
        newr = tam - quotient * newr;
    }
    if (r > 1)
        return -1;
    if (t < 0)
        t += phi;
    return t;
}
void encrypt(string s, long long e, long long n, long long d) {
    string a = "", conf = "", auth = "";
    for (int i = 0; i < s.length(); i++)
        a += to_string((int)((int)s[i]-32));
    for (int i = 0; i < a.length();) {
        string t = "";
        for (int j = 0; j < 4 && i<a.length(); j++)
            t += a[i++];
        int m = stoi(t);
        conf += to_string(power(m, e, n))+" ";
        auth += to_string(power(m, d, n))+" ";
    }
    cout << "\nEncryption for Confidentiality: " << conf;
    cout << "\nEncryption for Authentication: " << auth;
}
void decrypt(string m, long long e, long long n, bool flag, long long d) {
    string de = "";
    for (int i = 0; i < m.length();) {
        string s = "";
        while (m[i] != ' ' && i<m.length())
            s += m[i++];
        i++;
```

```cpp
        int c = stoi(s);
        if (flag) c = power(c, e, n);
        else c = power(c, d, n);
        s = to_string(c);
        if (s.length() < 4 && i<m.length()) {
            reverse(s.begin(), s.end());
            while (s.length() < 4)
                s += "0";
            reverse(s.begin(), s.end());
        }
        string t="";
        t += s[0];
        t += s[1];
        int so1 = stoi(t);
        de += (char)((int)(so1 + 32));
        t = "";
        if (s[2] != '\0') {
            t += s[2];
            t += s[3];
            int so2 = stoi(t);
            de += (char)((int)(so2 + 32));
        }
    }
    cout << "\nDecrypt message: " << de;
}
void entervale(long long& q, long long& p, long long& e,long long& phi) {
    do {
        cout << "Enter p: ";
        cin >> p;
    } while (!checkPrime(p));

    do {
        cout << "Enter q: ";
        cin >> q;
    } while (!checkPrime(q) || q == p);

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

    do {
        cout << "Enter e: ";
        cin >> e;
    } while (gcd(e, phi) > 1);
}
int main() {
    long long p, q, e, phi;
    cout << "You want to encrypt (0) or decrypt (other): ";
    bool f;
    cin >> f;
    if (f) {
        entervale(q, p, e, phi);
        cout << "Enter the message: ";
        string s;
        cin.ignore();
        getline(cin, s);
        long long n = q * p;
        cout << "Decryption for Confidentiality (0) or  Decryption for
Authentication (other): ";
        bool flag;
        cin >> flag;
        if (!flag) {
            long long d;
            cout << "Enter d: ";
            cin >> d;
            decrypt(s, e, n, flag, d);
        }
        else decrypt(s, e, n, flag, 0);
    }
    else {
        cout << "You want to enter p, q, e (0) or generate a randomized keypair
(other): ";
        cin >> f;
        if (!f) {
            entervale(q, p, e, phi);
        }
        else {
            p = randomPrime();
            do
                q = randomPrime();
            while (q == p);
            phi = (q - 1) * (p - 1);
            e = randomRelativePrime(phi);
        }
        cout << "\nEnter the message: ";
        string s;
        cin.ignore();
        getline(cin,s);
        long long n = q * p;
        long long d = determineD(phi, e);
        cout << "\nq: " << q;
        cout << "\np: " << p;
        cout << "\ne " << e;
        cout << "\nd: " << d;
        encrypt(s, e, n, d);
    }
}
```

Task 4:

```
// g++ -g3 -ggdb -O0 -DDEBUG -I/usr/include/cryptopp Driver.cpp -o Driver.exe -
lcryptopp -lpthread
// g++ -g -O2 -DNDEBUG -I/usr/include/cryptopp Driver.cpp -o Driver.exe -lcryptopp
-lpthread

#include "cryptopp\eccrypto.h"
#include "cryptopp\integer.h"
#include "cryptopp\oids.h"

#include "cryptopp\files.h"
using CryptoPP::FileSource;
using CryptoPP::FileSink;
using CryptoPP::BufferedTransformation;

#include "cryptopp\osrng.h"
using CryptoPP::AutoSeededRandomPool;
using CryptoPP::byte;

#include <iostream>
using std::wcin;
using std::wcout;
using std::cerr;
using std::endl;

#include <string>
using std::string;
using std::wstring;

#include "cryptopp\cryptlib.h"

#include "cryptopp\hex.h"
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;

#include "cryptopp\filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;
using CryptoPP::Redirector; // string to bytes
#include <cryptopp/base64.h>

#include <fstream>
/* Convert string*/
#include <locale>
using std::wstring_convert;
#include <codecvt>
```

```
using  std::codecvt_utf8;
using namespace CryptoPP;
wstring string_to_wstring(const std::string& str);
string wstring_to_string(const std::wstring& str);

void sign() {
    AutoSeededRandomPool prng;
    ECDSA<ECP, SHA256>::PrivateKey privateKey;
    ByteQueue queue;
    FileSource file("privateKey.key", true /*pumpAll*/);
    file.TransferTo(queue);
    queue.MessageEnd();
    privateKey.Load(queue);
    ECDSA<ECP, SHA256>::Signer signer(privateKey);
    std::cout << "Enter the name of file store message: ";
    string filemessage;
    std::cin >> filemessage;
    std::ifstream ifs(filemessage);
    string m;
    getline(ifs, m);
    wstring message = string_to_wstring(m);
    size_t siglen = signer.MaxSignatureLength();
    std::string signature(siglen, 0x00);
    siglen = signer.SignMessage(prng, (const byte*)&message[0], message.size(),
(byte*)&signature[0]);
    signature.resize(siglen);
    ifs.close();
    std::ofstream ofs("output.txt");
    ofs << signature;
}

void verify() {
    ECDSA<ECP, SHA256>::PublicKey publicKey;
    ByteQueue queue;
    FileSource file("publicKey.key", true /*pumpAll*/);
    file.TransferTo(queue);
    queue.MessageEnd();
    publicKey.Load(queue);
    ECDSA<ECP, SHA256>::Verifier verifier(publicKey);
    bool result = false;
    std::cout << "Enter the name of file store message: ";
    string filemessage;
    std::cin >> filemessage;
    std::ifstream ifs(filemessage);
    string m, signature;
    getline(ifs, m);
    wstring message = string_to_wstring(m);
    ifs.close();
    std::ifstream _ifs("output.txt");
    getline(_ifs, signature);
    _ifs.close();

    result = verifier.VerifyMessage((const byte*)&message[0], message.size(),
(const byte*)&signature[0], signature.size());
    if (result)
        std::cout << "Verified signature on message" << std::endl;
    else
        cerr << "Failed to verify signature on message" << std::endl;
}
int main(int argc, char* argv[])
{
    /*AutoSeededRandomPool prng;
    ECDSA<ECP, SHA256>::PrivateKey privateKey;
    privateKey.Initialize(prng, ASN1::secp256k1());
    ECDSA<ECP, SHA256>::PublicKey publicKey;
    privateKey.MakePublicKey(publicKey);
    ByteQueue queue;
    privateKey.Save(queue);
    FileSink file("privateKey.key");
    queue.CopyTo(file);
    file.MessageEnd();
    ByteQueue _queue;
    publicKey.Save(_queue);
    FileSink _file("publicKey.key");
    _queue.CopyTo(_file);
    _file.MessageEnd();*/
    std::cout << "Sign (0) or verify (other): ";
    bool flag(0);
    std::cin >> flag;
    if (flag)
        verify();
    else sign();
    return 0;
}

/* convert string to wstring */
wstring string_to_wstring(const std::string& str)
{
    wstring_convert<codecvt_utf8<wchar_t> > towstring;
    return towstring.from_bytes(str);
}

/* convert wstring to string */
string wstring_to_string(const std::wstring& str)
{
    wstring_convert<codecvt_utf8<wchar_t> > tostring;
    return tostring.to_bytes(str);
}
```