Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Лабораторна робота

із Криптографії №5

Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем

Виконали:

Топчій Микита ФБ - 74

Височанська Вікторія ФБ - 71

Перевірено_____

Київ 2019

КРИПТОГРАФІЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №5

Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем

Мета та основні завдання роботи:

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

Порядок і рекомендації щодо виконання роботи

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.
2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і $p1, q1$ довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \le p1q1$; p і q – прості числа для побудови ключів абонента *A*, $p1$ і $q1$ – абонента *B*.
3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ (d, p, q) та відкритий ключ (n, e) . За допомогою цієї функції побудувати схеми RSA для абонентів *A* і *B* – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , $(e1, n1)$ та секретні d і $d1$
4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів *A* і *B*. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання.
   За допомогою датчика випадкових чисел вибрати відкрите повідомлення *M* і знайти криптограму для абонентів *A* и *B*, перевірити правильність розшифрування. Скласти для *A* і *B* повідомлення з цифровим підписом і перевірити його.
5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання.

Труднощі:

Основним питання, що виникло при виконанні даного комп’ютерного практикуму, було як працювати з числами великої розрядності у обраній мові програмування(C++), адже за замовчуванням стандартні типи не підтримують таку розрядність. Для вирішення цього питання було обрано бібліотеку Boost.

Також виникло питання з генерацією великих псевдовипадкових чисел, вбудований в мову генератор не давав числа достатньої розрядності, тому було використано Boost.Random.

Обрані числа:

P = 228478183506015368011317436896217482289154169757319

Q = 19745967561219605259533383841115519622675442104028 3

P1 = 351602323736395062876373706881784750259331497912 8479

Q1 = 78340591635928942878772424149794241730218403991091

E – фіксоване; E = 2^16 + 1 = 65537

Числа, що не підійшли:

P:

11145277244195871610308167653474023526300203402797

55726386220979358051540838267370117631501017013981

72444302087273165467003089747581152920951322118175

…

Q:

21698865451889676109377344880346724860082903411 03

43397730903779352218754689760693449720165806822 05

86795461807558704437509379521386899440331613644 09

…

P1:

64044139114097461361816704350051867078202458636 23

12808827822819492272363340870010373415640491727245

19213241734229238408545011305015560123460737590867

…

Q1:

52227061090619295252514949433196161153478935994 07

104454122181238590505029898866392322306957871988 13

156681183271857885757544848299588483460436807982 19

…

Відкритий текст:

625079447713544335536190327587578229365097294888529234576709741784323118586 37

Шифрований текст:

387655732719369507819781957817689435875349126614649216302177519649856781542354279830934604209911886 77

Цифровий підпис:

258965982815796165193872296674007363261239487977508733603569433202238460317706307598078829384648934 80

SendKey:

k1=490768756536391396788407679462667649512298130613638437817028576195805822960292627776708914889273545 79

S1=267461900599724371820286409840742175495790130888373305760702555708294738677130054674295722457344363654

Висновок: у ході комп’ютерного практикуму було набуто навичок роботи з числами великої розрядності, написання тестів перевірки чисел на простоту та методів генерації ключів для асиметричної криптосистеми RSA. Набуто навичок побудови цифрового підпису на основі криптосистеми RSA.

На основі RSA було побудовано модель протоколу для розсилання ключів по відкритих каналах.

Програмна реалізація:

Main.cpp

```
#include "Header.h"
int main()
{
	srand(time(0));
	cpp_int a("357425491988726172913535086566266425 67");
	cpp_int b("3574254919887261729135350865662664256778680092345");

	std::pair <cpp_int, cpp_int> SimpleNumbersPairA;
	std::pair <cpp_int, cpp_int> SimpleNumbersPairB;
	SimpleNumbersPairA = GenerateSimpleNumberPair(a, b);
	SimpleNumbersPairB = GenerateSimpleNumberPair(a, b);
	cpp_int FirstComposition = SimpleNumbersPairA.first * SimpleNumbersPairA.second;
	cpp_int SecondComposition = SimpleNumbersPairB.first * SimpleNumbersPairB.second;
	std::cout << SimpleNumbersPairA.first << "  " << SimpleNumbersPairA.second << std::endl;
	while (FirstComposition > SecondComposition) {
		SimpleNumbersPairB = GenerateSimpleNumberPair(a, b);
		SecondComposition = SimpleNumbersPairB.first * SimpleNumbersPairB.second;
	}
	std::cout << SimpleNumbersPairB.first << "  " << SimpleNumbersPairB.second << std::endl;
	// Values in report:
	/*cpp_int P("22847818350601536801131743689621748228915416975 7319");
	cpp_int Q("19745967561219605259533383841115519622675442104 0283");

	cpp_int P1("35160232373639506287637370688178475025933149791284 79");
	cpp_int Q1("78340591635928942878772424149794241730218403991091");
	std::pair <cpp_int, cpp_int> SimpleNumbersPairA(P, Q);
	std::pair <cpp_int, cpp_int> SimpleNumbersPairB(P1, Q1);*/
	std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrivateKeyA;
	std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrivateKeyB;
	std::pair<cpp_int, cpp_int> PublicKeyA;
	std::pair<cpp_int, cpp_int> PublicKeyB;
	GenerateKeyPair(PrivateKeyA, PublicKeyA, SimpleNumbersPairA);
	GenerateKeyPair(PrivateKeyB, PublicKeyB, SimpleNumbersPairB);
	std::cout << "A: d = " << PrivateKeyA.first << "\n" << " n = " << PublicKeyA.first << "\n" << " e = "
<< PublicKeyA.second << std::endl;
	std::cout << "B: d = " << PrivateKeyB.first << "\n" << " n = " << PublicKeyB.first << "\n" << " e = "
<< PublicKeyB.second << std::endl;
	//-----------------------------------------------------------------------------------
	cpp_int Message("0x8A323E34209BA7068FD7C190C132122F859ECA9F9C6057AADBCE86D58C2395CD");
	cpp_int EncMessage = Encrypt(Message, PublicKeyA);
	cpp_int DecMessage = Decrypt(EncMessage, PrivateKeyA);
	std::cout << "Message: " << Message<< std::endl;
	std::cout << "EncMessage: " << EncMessage<< std::endl;
	std::cout << "DecMessage: " << DecMessage<< std::endl;
	//-------------------------------------------------------------
	std::pair<cpp_int, cpp_int> Signature;
	Signature = Sign(Message, PrivateKeyA);
	std::cout << "Signature: " << Signature.second << std::endl;
	std::cout << Verify(Signature, PublicKeyA) << std::endl;
	//-------------------------------------------------------------
	std::pair<cpp_int, cpp_int> Notif;
	Notif = SendKey(PublicKeyA, PrivateKeyA, PublicKeyB, Message);
	std::cout << "k1: " << Notif.first << "\nS1: " << Notif.second << std::endl;
	std::cout << ReceiveKey(PublicKeyB, PrivateKeyB, PublicKeyA, Notif) << std::endl;

	system("pause");
	return 0;
}
```

Header.h

```
#pragma once
#include <algorithm>
#include <iostream>
#include <numeric>
#include <boost/multiprecision/cpp_int.hpp>
#include <boost/multiprecision/integer.hpp>
#include <boost/random/mersenne_twister.hpp>
#include <boost/random/uniform_int_distribution.hpp>
#include <boost/multiprecision/random.hpp>
#include <utility>      // std::pair, std::make_pair
#include <boost/integer/mod_inverse.hpp>
using namespace boost::multiprecision;
using namespace boost::random;
bool MLprimTest(cpp_int NumberToCheck, int RoundsAmount); // Miller-Rabin primality test
cpp_int GetPrimeInRange(cpp_int MinBound, cpp_int MaxBound); // Get prime number in range [MinBound; MaxBound]
std::pair<cpp_int, cpp_int> GenerateSimpleNumberPair(cpp_int MinBound, cpp_int MaxBound); // Generate simple
numbers pair;

// Generate private and public keys; PrivateKey (d, p, q); PublicKey (e, n);
void GenerateKeyPair(std::pair<cpp_int, std::pair<cpp_int, cpp_int>>& PrivateKey,
                                                std::pair<cpp_int, cpp_int>& PublicKey, std::pair<cpp_int,
cpp_int > SimpleNumbers);

cpp_int Encrypt(cpp_int Message, std::pair<cpp_int, cpp_int> PublicKey); // Encryption function



cpp_int Decrypt(cpp_int EncMessage, std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrivateKey);  //
Decryption function

std::pair<cpp_int, cpp_int> Sign(cpp_int Message, std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrivateKey);
//Create Digital sign
bool Verify(std::pair<cpp_int, cpp_int> Sign, std::pair<cpp_int, cpp_int> PublicKey); // Digital sign
verification function

std::pair<cpp_int, cpp_int> SendKey(std::pair<cpp_int, cpp_int> PubKeyA, std::pair<cpp_int, std::pair<cpp_int,
cpp_int>> PrvtKeyA,
                                                                                    std::pair<cpp_int,
cpp_int> PubKeyB, cpp_int SecValue); //SendKey function for

                                                                                       //confidential
key distribution protocol
bool ReceiveKey(std::pair<cpp_int, cpp_int> PubKeyB, std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrvtKeyB,
        std::pair<cpp_int, cpp_int> PubKeyA, std::pair<cpp_int, cpp_int> Notification);
```

Header.cpp

```
#include "Header.h"
bool MLprimTest(cpp_int NumberToCheck, int RoundsAmount)
{
         // p - 1 = d * 2^s;
        // NumberToCheck - 1 = d * 2^s; d is odd
        cpp_int NumberToCheckCopy = NumberToCheck - 1;
        int s = 0;
        while (NumberToCheckCopy % 2 == 0) {
                s++;
                NumberToCheckCopy /= 2;
        }
        cpp_int d = NumberToCheckCopy; // d is odd
        //std::time_t now = std::time(0);
        //boost::random::mt19937 gen{ static_cast<std::uint32_t>(now) }; //Pseudo-random number generator
        std::time_t now = std::time(0);
        typedef independent_bits_engine<mt19937, 256, cpp_int> generator_type;
        generator_type gen{ static_cast<std::uint32_t>(now) };

        cpp_int TempRandomNumber;
        cpp_int X;
        for (int i = 0; i < RoundsAmount; i++) {
                TempRandomNumber = (1 + gen()) % NumberToCheck - 2;
                //std::cout << TempRandomNumber << std::endl;
```

```
                X = powm(TempRandomNumber, d, NumberToCheck);
                if (X == 1 || X == NumberToCheck - 1) continue;
                for (int j = 0; j < s - 1; j++) {
                        X = powm(X, 2, NumberToCheck);
                        if (X == 1) return 0;
                        else if (X == NumberToCheck - 1) break;
                }
                return 0;
        }
        return 1;
}

cpp_int GetPrimeInRange(cpp_int MinBound, cpp_int MaxBound)
{
        cpp_int Number;
        std::time_t now = std::time(0);
        typedef independent_bits_engine<mt19937, 256, cpp_int> generator_type;
        generator_type gen{ static_cast<std::uint32_t>(now) };
        do {
                Number = (gen() % MaxBound + MinBound) % MaxBound;
                //std::cout << Number << std::endl;
        } while (!MLprimTest(Number, 15));
        return Number;
}

std::pair<cpp_int, cpp_int> GenerateSimpleNumberPair(cpp_int MinBound, cpp_int MaxBound)
{
        cpp_int CoreP, CoreQ, p, q; // p = 2*i*CoreP + 1, q = 2*i*CoreQ + 1; i = 1, 2...
        CoreP = GetPrimeInRange(MinBound, MaxBound);
        //std::cout << "CoreP " << std::endl;
        CoreQ = GetPrimeInRange(MinBound, MaxBound);
        //std::cout << "CoreQ " << std::endl;
        while (CoreP == CoreQ) {
                CoreQ = GetPrimeInRange(MinBound, MaxBound);
        }
        int i = 1;
        do {
                p = 2 * i * CoreP + 1;
                i++;
                std::cout << " Pi = " << p << std::endl;
                std::cout << i << "\t";
        } while (!MLprimTest(p, 10));
        std::cout << std::endl;
        i = 1; // Zeroing counter
        do {
                q = 2 * i * CoreQ + 1;
                i++;
                std::cout << " Qi = " << q << std::endl;
                std::cout << i << "\t";
        } while (!MLprimTest(q, 10));
        std::cout << std::endl;
        return std::pair<cpp_int, cpp_int>(p, q);
}

void GenerateKeyPair(std::pair<cpp_int, std::pair<cpp_int, cpp_int>>& PrivateKey,
                                                std::pair<cpp_int, cpp_int>& PublicKey, std::pair<cpp_int,
cpp_int > SimpleNumbers)
{
        cpp_int n; // n = p * q;
        n = SimpleNumbers.first * SimpleNumbers.second;
        cpp_int fi; // Euler function; fi(n) = (p-1)*(q-1)
        fi = (SimpleNumbers.first - 1) * (SimpleNumbers.second - 1);
        cpp_int e = pow((cpp_int)2, 16) + 1; // e = 2^16 + 1
        std::cout  << "gcd(e, fi) = " << gcd(e, fi) << std::endl; // For test
        cpp_int d = boost::integer::mod_inverse(e, fi);
        std::cout << "e*d mod fi = " << (e * d) % fi << std::endl; //For test
        //PrivateKey: <d, <p, q>>; SimpleNumbers : <p, q>;
        PrivateKey.first = d;
        PrivateKey.second.first = SimpleNumbers.first;
        PrivateKey.second.second = SimpleNumbers.second;
        // PublicKey : <n, e>;
        PublicKey.first = n;
```

```
        PublicKey.second = e;
}

cpp_int Encrypt(cpp_int Message, std::pair<cpp_int, cpp_int> PublicKey)
{
        //PublicKey: (n, e);
        cpp_int EncMessage; //Encrypted message
        EncMessage = powm(Message, PublicKey.second, PublicKey.first); // C = M^e mod n
        return EncMessage;
}

cpp_int Decrypt(cpp_int EncMessage, std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrivateKey)
{
        cpp_int DecMessage; //Decrypted message
        cpp_int n = PrivateKey.second.first * PrivateKey.second.second; // n = p * q;
        DecMessage = powm(EncMessage, PrivateKey.first, n);
        return DecMessage;
}

std::pair<cpp_int, cpp_int> Sign(cpp_int Message, std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrivateKey)
{
        cpp_int Sign; // S = M^d mod n;
        //PrivateKey: <d, <p, q>>
        cpp_int n = PrivateKey.second.first * PrivateKey.second.second;
        Sign = powm(Message, PrivateKey.first, n);
        return std::pair<cpp_int, cpp_int>(Message, Sign);
}

bool Verify(std::pair<cpp_int, cpp_int> Sign, std::pair<cpp_int, cpp_int> PublicKey)
{
        cpp_int DecSign;
        // PublicKey: <n, e>;
        DecSign = powm(Sign.second, PublicKey.second, PublicKey.first);
        if (Sign.first = DecSign) return 1;
        else return 0;
}

std::pair<cpp_int, cpp_int> SendKey(std::pair<cpp_int, cpp_int> PubKeyA,
        std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrvtKeyA, std::pair<cpp_int, cpp_int> PubKeyB,
cpp_int SecValue)
{
        //S = k(SecValue)^d mod n;
        cpp_int d = PrvtKeyA.first;
        cpp_int n = PubKeyA.first;
        cpp_int S = powm(SecValue, d, n);
        //PublicKey: <n, e>
        //PrivateKey: <d, <p, q>>
        cpp_int e1 = PubKeyB.second;
        cpp_int n1 = PubKeyB.first;
        // k1 = k(SecValue)^e1 mod n1;
        cpp_int k1 = powm(SecValue, e1, n1);
        // S1 = S^e1 mod n1;
        cpp_int S1 = powm(S, e1, n1);
        return std::pair<cpp_int, cpp_int>(k1, S1);
}

bool ReceiveKey(std::pair<cpp_int, cpp_int> PubKeyB, std::pair<cpp_int, std::pair<cpp_int, cpp_int>> PrvtKeyB,
        std::pair<cpp_int, cpp_int> PubKeyA, std::pair<cpp_int, cpp_int> Notification)
{
        //PrvtKey: <d, <p, q>>
        cpp_int d1 = PrvtKeyB.first;
        //Notification: <k1, S1>
        cpp_int k1 = Notification.first;
        cpp_int S1 = Notification.second;
        cpp_int n1 = PubKeyB.first;
        cpp_int SecValue = powm(k1, d1, n1);
        cpp_int S = powm(S1, d1, n1);
        cpp_int e = PubKeyA.second;
        cpp_int n = PubKeyA.first;
        if (SecValue == powm(S, e, n)) return 1;
        else return 0;
}
```