Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Фізико-технічний інститут

Лабораторна робота

із Криптографії №5

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

Виконав:

студент 3 курсу ФТІ

групи ФБ-74,ФБ-72

Скуратов Илья, Демиденко Дарья

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

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 $pq \leq p1q1$; p і q – прості числа для побудови ключів абонента A, p1 iq1 – 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.

Обрані числа:

Абонент А

```
P("426607533187898514262889419997972951719857068972468470327252186232811817124 83");
```

```
Q("438889191245969860594842406195064855456504135352497259524298309675395018617 81");
```

Абонент Б

```
P1("9860087683395917844212339922802874333727414325977032278663301669771888885 4293");
```

```
Q1("4834770550286548651000208679240440837674146262415995501746558519797584654 1939");
```

E = 2^16 + 1 = 65537 *(ФІКСОВАНЕ)*

Відкритий текст:

132349847982573289743895632897198278972978 97

Шифрований текст:

236992765679517894096416183377232743833973292490253447169218736821235768368772220599553653216173059797774839082140230361088912917086055554723195642814876 6

Цифровий підпис:

247781165786098641744246416814660601059451087251423292826223193193180054661830926468576836400364645716563248394527151839522439069535049390522964337635373 5

For site :

k1=2202bcea38069eea3186e0e34708153f7669b7ed4bffa1ed204eefc446bfd2f56ac50113f2c15dbe846a04003f8fb8d86d5641f742620e93b4b356f64d49fa19

S1=2822f40ed031bd71329180be63910d8b7df9677f4e239e3df7ba4f60cefbe706b833180624985deccd0510db136cedf7d4d7107892d7a679ead47697fdde34d3

N=23bfc798cfe6decce94a42b15f72a1a29ee8cd7aa8318d48efec96785906455047809d472669f06e8c573e75d0f28bf437e2546290f3ffd7113372b61419da9f

Public Exponent = 10001

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

Код :

```
#include "stdafx.h"
#include <iostream>

#include <algorithm>
#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>
#include <boost/integer/mod_inverse.hpp>

using namespace boost::multiprecision;
using namespace boost::random;
using namespace std;

using namespace boost::multiprecision;
using namespace boost::random;
using namespace std;

cpp_int e("65537");

cpp_int
serverkey("0x882BD1C544CA94EEFEEB483F72A6676C9D754118274682C669AE28DDEE253D73F7F0E5E0D
6F0177D714AA9D96A89A9C4441FE02E3E3B2CD73C6603C26F08F6A1");

cpp_int p, q, p1, q1;

std::time_t now = std::time(NULL);
typedef independent_bits_engine <mt19937, 256, cpp_int> generator_type;
generator_type gen{ static_cast<std::uint32_t>(now) };


cpp_int S1;

//cpp_int P("13"); // наш открытый текст
cpp_int P("132349847982573289743895632897198278972978897"); // наш открытый текст

cpp_int n;

cpp_int sendkey(cpp_int P, cpp_int p, cpp_int q,cpp_int d,cpp_int n);
cpp_int close_key(cpp_int p, cpp_int q);
cpp_int open_key(cpp_int p, cpp_int q);
cpp_int rashifr(cpp_int E, cpp_int d1, cpp_int n1, cpp_int S1, cpp_int n);
cpp_int sign(cpp_int P, cpp_int d, cpp_int n, cpp_int n1);
void generate_key_pair();
bool miller(cpp_int key, int iterac);
```

```
cpp_int gcd(cpp_int a, cpp_int b, cpp_int& x, cpp_int& y);


int  main()
{




	cpp_int
p("426607533187898514262889419997972951719857068972468470327252186232811817124 83");
	cpp_int
q("438889191245969860594842406195064855456504135352497259524298309675395018617 81");
	cpp_int
p1("986008768339591784421233992280287433372741432597703227866330166977188888542 93");
	cpp_int
q1("483477055028654865100020867924044083767414626241599550174655851979758465419 39");

//	generate_key_pair();




	cpp_int n = open_key(p, q);
	cpp_int d = close_key(p, q);

	cpp_int n1 = open_key(p1, q1);
	cpp_int d1 = close_key(p1, q1);

	//if (serverkey < n) cout << "ERROR!!!" << endl;

	cout << "User A : " << endl;
	cout << "open key  (e,n) :"<< "( " << e << "," << n << " )"<<endl;
	cout << "close key (d,n) : " << "( " << d << "," << n << " )" << endl;

	cout << endl<<  "User B : " << endl;
	cout << "open key  (e1,n1) :" << "( " << e << "," << n1 << " )" << endl;
	cout << "close key (d1,n1) : " << "( " << d1 << "," << n1 << " )" << endl;



	cout <<endl<< "OPEN TEXT : " << P << endl ;

	cpp_int E = sendkey(P, e, n1, d,n);

	cout << "SHIFR TEXT : " << E << endl << endl;



	cpp_int O = rashifr(E, d1, n1, S1,n);

	cout << endl << "RASHIFR TEXT : " << O << endl << endl;




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

```
cpp_int open_key(cpp_int p, cpp_int q)
{
	n = p * q;




	return n;
}

cpp_int close_key(cpp_int p, cpp_int q)
{
	cpp_int x, y;

	cpp_int fi = (q - 1)*(p - 1);

	cpp_int key = gcd(e, fi, x, y);
	if (key < 0) key += fi;

	return key;
}

cpp_int sendkey(cpp_int P, cpp_int e, cpp_int n1, cpp_int d, cpp_int n)
{

	cpp_int E = powm(P, e, n1);


	///cpp_int S = powm(P, d, n);


	 //S1 = powm(S, e, n1);

	S1 = sign(P, d, n, n1);

	cout << "S1 : " << S1 << endl;

	return E;
}

cpp_int sign(cpp_int P, cpp_int d, cpp_int n, cpp_int n1)
{
	cpp_int S = powm(P, d, n);

	cpp_int sign;

	sign = powm(S, e, n1);


	return sign;
}

cpp_int rashifr(cpp_int E, cpp_int d1, cpp_int n1,cpp_int S1, cpp_int n)
{

	cpp_int k;

	k = powm(E, d1, n1);

	cpp_int S = powm(S1, d1, n1);
```

```
	cpp_int a = powm(S, e, n);

	cout << "PROVERKA k  : " << a ;

	return k;

}

cpp_int gcd(cpp_int a, cpp_int b, cpp_int& x, cpp_int& y) {
	if (a == 0) {
		x = 0; y = 1;
		return b;
	}
	cpp_int x1, y1;
	cpp_int d = gcd(b % a, a, x1, y1);
	x = y1 - (b / a) * x1;
	y = x1;

	return x;
}

bool miller(cpp_int key, int iterac)
{
	if (key % 2 == 0)
		return false;

	cpp_int t = key - 1;

	int s = 0;

	while (t % 2 == 0)
	{
		t /= 2;
		s += 1;
	}

	for (int i = 0; i < iterac; i++)
	{
		cpp_int rand1;

		do
		{
			rand1 = gen();

		} while (rand1 < 2 || rand1 >= key - 2);

		cpp_int x = powm(rand1, t, key);

		if (x == 1 || x == key - 1)
			continue;

		for (int r = 1; r < s; r++)
		{

			x = powm(x, 2, key);


			if (x == 1)
				return false;


			if (x == key - 1)
```

```
                        break;
                }

                if (x != key - 1)
                        return false;

        }


        return true;



}

void generate_key_pair()
{


        int i;

        do { p = gen(); } while (miller(p, 20) == 0);
        do { q = gen(); } while (miller(q, 20) == 0);

        do
        {
                do { p1 = gen(); } while (miller(p1, 20) == 0);
                do { q1 = gen(); } while (miller(q1, 20) == 0);
        } while (q*p > q1*p1);


}
```