МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем і мереж

Лабораторна робота №3

з дисципліни: «Екстремальне програмування»

Ключовий обмін Діффі-Хеллмана

|  |
| --- |
| Виконав:  Студент групи ЕКП-ІСМ-301  Гасій В. Ю.  Прийняв:  Щербак С. С. |

Львів – 2020

**Алгоритм генерації простого числа**

1. Згенерувати випадкове n-бітове число *p*.

2. Встановити його старший і молодший біти рівними 1. Старший біт гарантуватиме необхідну довжину шуканого числа, а молодший біт – забезпечувати його непарність.

3. Переконатися, що *p* не ділиться на невеликі прості числа: 3, 5, 7, 11 і т.д. Найбільш ефективною є перевірка на можливість ділення на всі прості числа, менші 2000.

4. Виконати тест Рабіна-Міллера мінімум 5 разів.

Якщо *p* не пройшло хоча б одну перевірку в пунктах 3 або 4, то воно не є простим. Перевірка того, що випадкове непарне число *р* не ділиться на 3, 5 і 7, відсікає 54% непарних чисел. Перевірка можливості ділення на всі прості числа, які менші 256, відсікає 80% складених непарних чисел.

Навіть якщо складене число «пройшло» через цей алгоритм, це буде

відразу ж помічено, тому що шифрування і дешифрування не будуть

працювати.

**Завдання**

Виконати алгоритм генерації простого числа.

**Розв’язання**

Main.cpp

#include "Include.h"

using boost::multiprecision::cpp\_int;

int main()

{

cpp\_int num = create\_prime\_number(80);

std::cout << num << std::endl;

std::cout << is\_prime(num, 80, 5) << std::endl;

system("pause");

return 0;

}

Include.h

#pragma once

#include <iostream>

#include <boost/multiprecision/cpp\_int.hpp>

#include "Prime\_Number.h"

Prime\_Number.h

#pragma once

#include <iostream>

#include <cmath>

#include <boost/multiprecision/cpp\_int.hpp>

#include <ctime>

#include <vector>

using boost::multiprecision::cpp\_int;

cpp\_int modular\_pow(cpp\_int base, cpp\_int exponent, cpp\_int modulus);

cpp\_int create\_number(int n);

cpp\_int convert\_binary\_to\_decimal(cpp\_int number);

bool Rabin\_Miller(const cpp\_int &p, int n , int t );

cpp\_int create\_prime\_number(int n);

bool is\_prime(const cpp\_int &p, int n , int t );

std::string convert\_decimal\_to\_binary(cpp\_int number);

Prime\_Number.cpp

#include "Prime\_Number.h"

using boost::multiprecision::cpp\_int;

std::vector<int> prime\_numbers = { 2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,521,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719,727,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,941,947,953,967,971,977,983,991,997,1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091,1093,1097,1103,1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,1289,1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,1481,1483,1487,1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,1597,1601,1607,1609,1613,1619,1621,1627,1637,1657,1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,1823,1831,1847,1861,1867,1871,1873,1877,1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,1999 };

cpp\_int modular\_pow(cpp\_int base, cpp\_int exponent, cpp\_int modulus)

{

cpp\_int result = 1;

while (exponent > 0)

{

if (exponent % 2 == 1)

{

result = (result \* base) % modulus;

}

exponent = exponent >> 1;

base = (base \* base) % modulus;

}

return result;

}

cpp\_int create\_number(int n)

{

srand(time(NULL));

cpp\_int number;

std::string temp = "\0";

temp += "1";

for (int i = 0; i < n - 1; i++)

{

temp += std::to\_string(rand() % 2);

}

number = convert\_binary\_to\_decimal(cpp\_int(temp));

return number;

}

cpp\_int convert\_binary\_to\_decimal(cpp\_int number)

{

cpp\_int decimalNumber = 0, remainder;

int i = 0;

while (number != 0)

{

remainder = number % 10;

number /= 10;

decimalNumber += remainder \* cpp\_int(pow(2, i));

++i;

}

return decimalNumber;

}

bool Rabin\_Miller(const cpp\_int &p, int n = 0, int t = 0)

{

int b = 1;

while ((p - 1) % cpp\_int(pow(2, b)) == 0)

{

b++;

}

b -= 1;

cpp\_int m = (p - 1) / cpp\_int(pow(2, b));

cpp\_int a;

cpp\_int z;

srand(time(NULL));

for (int i = 0; i < t; i++)

{

a = create\_number(rand() % (n - 2) + 2);

z = modular\_pow(a, m, p);

if (z == 1 || z == (p - 1))

{

continue;

}

for (int j = 0; j < b - 1; j++)

{

z = modular\_pow(z, 2, p);

if (z == 1)

{

return false;

}

if (z == (p - 1))

{

break;

}

}

if (b == i && z != (p - 1))

{

return false;

}

}

return true;

}

cpp\_int create\_prime\_number(int n)

{

int t = 5;

cpp\_int number;

std::string temp="\0";

do

{

number = create\_number(n);

temp = convert\_decimal\_to\_binary(number);

temp = temp.substr(0, temp.size() - 1);

temp += "1";

number = convert\_binary\_to\_decimal(cpp\_int(temp));

} while (!is\_prime(number, n, t));

return number;

}

bool is\_prime(const cpp\_int &p, int n = 0, int t = 0)

{

for (int i = 0; i < prime\_numbers.size(); i++)

{

if (p == prime\_numbers[i])

{

return true;

}

if (p%prime\_numbers[i] == 0)

{

return false;

}

}

if (Rabin\_Miller(p, n, t))

{

return true;

}

return false;

}

std::string convert\_decimal\_to\_binary(cpp\_int number)

{

std::string binaryNumber = "\0";

int remainder;

while (number != 0)

{

remainder = int(number % 2);

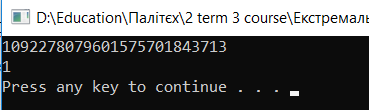
number /= 2;

binaryNumber = std::to\_string(remainder) + binaryNumber;

}

return binaryNumber;

}



**Висновок**: на цій лабораторній роботі було згенеровано велике n-бітне просте число.