Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информационных технологий автоматизированных систем

Отчет по лабораторной работе №5

по курсу «ИТОКБ»

на тему: «Асимметричное шифрование и электронная цифровая подпись»

Вариант 4

|  |  |
| --- | --- |
| Выполнил магистрант группы 025941: | Колесников В.Г. |
| Проверил: | Боброва Т.С. |

МИНСК

2021

**Задание 1:** Разработать программное обеспечение, реализующее функции генерации секретного и открытого ключей, шифрования и цифровой подписи для алгоритма RSA. Обмен входными и выходными данными должен осуществляться через файлы:

- файл открытого ключа;

- файл секретного ключа;

- файл исходного сообщения;

- файл зашифрованного сообщения.

Код программы приведен в конце лабораторной работы.

Результат работы программы поиска элементов для шифрования и самого шифрования и расшифрования приведен на рисунке 1.1.

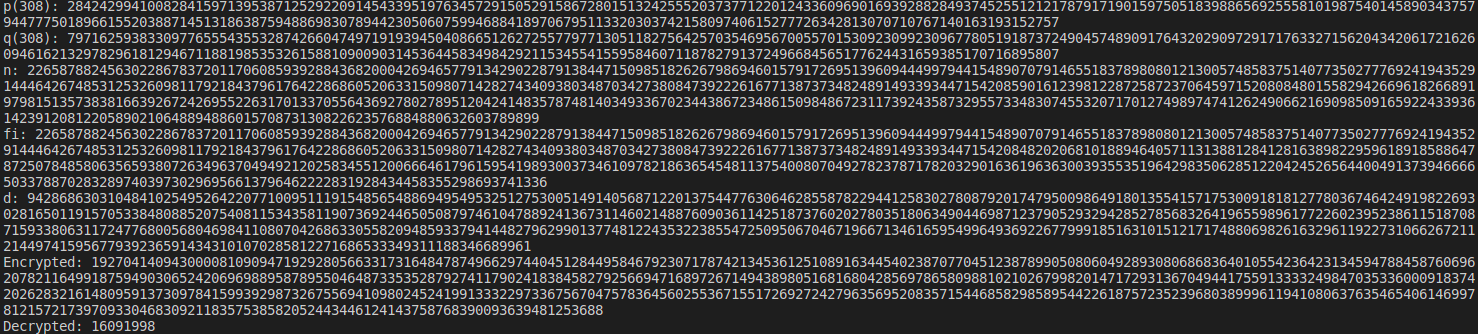


Рисунок 1.1 — Результат работы программы

В результате работы программы используются файлы:

1) prime1.txt ... prime10.txt — для выбора простых чисел p и q;

2) private\_key.txt — сгенерированный приватный ключ (пара {d, n});

3) public\_key.txt — сгенерированный публичный ключ (пара {e, n});

4) to\_encrypt.txt — данные для шифрования;

5) encrypted.txt — зашифрованные данные;

6) decryped.txt — расшифрованные данные.

**Код программы:**

#include <iostream>

#include <string>

#include <bits/stdc++.h>

#include <cmath>

using namespace std;

string removeLeadingZeros(string str)

{

while (str.size() > 1 && str[0] == '0')

{

str.erase(0, 1);

}

return str;

}

string multiplyLargeNumbers(string num1, string num2)

{

int len1 = num1.size();

int len2 = num2.size();

if (len1 == 0 || len2 == 0)

return "0";

// will keep the result number in vector

// in reverse order

vector<int> result(len1 + len2, 0);

// Below two indexes are used to find positions

// in result.

int i\_n1 = 0;

int i\_n2 = 0;

// Go from right to left in num1

for (int i = len1 - 1; i >= 0; i--)

{

int carry = 0;

int n1 = num1[i] - '0';

// To shift position to left after every

// multiplication of a digit in num2

i\_n2 = 0;

// Go from right to left in num2

for (int j = len2 - 1; j >= 0; j--)

{

// Take current digit of second number

int n2 = num2[j] - '0';

// Multiply with current digit of first number

// and add result to previously stored result

// at current position.

int sum = n1 \* n2 + result[i\_n1 + i\_n2] + carry;

// Carry for next iteration

carry = sum / 10;

// Store result

result[i\_n1 + i\_n2] = sum % 10;

i\_n2++;

}

// store carry in next cell

if (carry > 0)

result[i\_n1 + i\_n2] += carry;

// To shift position to left after every

// multiplication of a digit in num1.

i\_n1++;

}

// ignore '0's from the right

int i = result.size() - 1;

while (i >= 0 && result[i] == 0)

i--;

// If all were '0's - means either both or

// one of num1 or num2 were '0'

if (i == -1)

return "0";

// generate the result string

string s = "";

while (i >= 0)

s += to\_string(result[i--]);

return s;

}

string minusOne(string number)

{

const int numLength = number.length();

for (int i = numLength - 1; i > 0; i--)

{

int digit = number[i] - '0';

if (digit > 0)

{

number.replace(i, 1, to\_string(digit - 1));

break;

}

else

{

number.replace(i, 1, "9");

}

}

if (number[0] == '0')

{

string newNumber = "";

for (int i = 1; i < numLength; i++)

{

newNumber += number[i];

}

return newNumber;

}

else

{

return number;

}

}

bool gt(string a, string b)

{

a = removeLeadingZeros(a);

b = removeLeadingZeros(b);

return a.size() == b.size() ? a < b : a.size() < b.size();

}

void reduceDividend(string &a, const string &b)

{

for (string::size\_type cur\_pos = 0; cur\_pos < b.size(); ++cur\_pos)

{

string::size\_type a\_cur\_pos = a.size() - 1 - cur\_pos;

string::size\_type b\_cur\_pos = b.size() - 1 - cur\_pos;

char &cur\_a\_dig\_ref = a[a\_cur\_pos];

const char &cur\_b\_dig\_ref = b[b\_cur\_pos];

if (cur\_a\_dig\_ref >= cur\_b\_dig\_ref)

{

cur\_a\_dig\_ref -= cur\_b\_dig\_ref - '0';

}

else

{

(cur\_a\_dig\_ref -= cur\_b\_dig\_ref - '0') += 10;

for (int i = 1;; ++i)

{

if (a[a\_cur\_pos - i] == '0')

{

a[a\_cur\_pos - i] = '9';

}

else

{

--a[a\_cur\_pos - i];

break;

}

}

}

a = removeLeadingZeros(a);

}

a = removeLeadingZeros(a);

}

string incrementNumber(string number)

{

for (string::size\_type i = number.size() - 1;; --i)

{

if (number[i] < '9')

{

++number[i];

return number;

}

else

{

number[i] = '0';

if (i == 0)

{

number.insert(0, "1");

return number;

}

}

}

}

string divideLargeNumbers(string a, string b)

{

string res = "0";

string divisor = b;

while (divisor.size() < a.size())

{

divisor += '0';

}

while (true)

{

while (!gt(a, divisor))

{

reduceDividend(a, divisor);

res = incrementNumber(res);

}

if (divisor.size() <= b.size())

{

break;

}

divisor.erase(divisor.size() - 1);

res += '0';

res = removeLeadingZeros(res);

}

return res;

}

string sumLargeNumbers(string a, string b)

{

while (a.length() != b.length())

{

if (a.length() > b.length())

{

b = "0" + b;

}

else

{

a = "0" + a;

}

}

a = "0" + a;

b = "0" + b;

int finalLength = a.length();

string res = "";

for (int n = 0, i = finalLength - 1; i >= 0; i--, n /= 10)

{

n += a[i] - '0' + b[i] - '0';

res = to\_string(n % 10) + res;

}

return removeLeadingZeros(res);

}

string subtractLargeNumbers(string a, string b)

{

while (a.length() != b.length())

{

if (a.length() > b.length())

{

b = "0" + b;

}

else

{

a = "0" + a;

}

}

const int length = a.length();

string res = "";

int aGTb = a.compare(b);

string first;

string second;

if (aGTb >= 0)

{

first = a;

second = b;

}

else

{

first = b;

second = a;

}

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

{

res += '0';

}

for (int i = length - 1; i >= 0; i--)

{

if (first[i] >= second[i])

{

res.replace(i, 1, to\_string(first[i] - '0' - (second[i] - '0')));

}

else

{

res.replace(i, 1, to\_string((first[i] - '0' + 10 - (second[i] - '0')) % 10));

int j = i - 1;

while (j >= 0)

{

if (first[j] == '0')

{

first[j] = '9';

j--;

}

else

{

first.replace(j, 1, to\_string(first[j] - '0' - 1));

break;

}

}

}

}

return removeLeadingZeros(res);

}

string getRemainder(string a, string b)

{

return subtractLargeNumbers(a, multiplyLargeNumbers(divideLargeNumbers(a, b), b));

}

string gcdExt(string a, string b, string &x, string &y, bool &prevXWasMinus, bool &prevYWasMinus)

{

if (a.length() == 1 && a[0] == '0')

{

x = '0';

y = '1';

return b;

}

string x1;

string y1;

string d = gcdExt(getRemainder(b, a), a, x1, y1, prevXWasMinus, prevYWasMinus);

const string multiplied = multiplyLargeNumbers(divideLargeNumbers(b, a), x1);

if (prevXWasMinus)

{

x = sumLargeNumbers(y1, multiplied);

prevXWasMinus = false;

prevYWasMinus = true;

}

else if (prevYWasMinus)

{

x = sumLargeNumbers(y1, multiplied);

prevXWasMinus = true;

prevYWasMinus = false;

}

else

{

x = subtractLargeNumbers(y1, multiplied);

if (y1.compare(multiplied) >= 0)

{

prevXWasMinus = false;

}

else

{

prevXWasMinus = true;

}

}

y = x1;

return d;

}

const string decToBinStr(string sourceDec)

{

string resBin;

for (int i = 0; sourceDec.length() > 1 || sourceDec[0] != '0'; i++)

{

// cout << sourceDec << endl;

resBin = getRemainder(to\_string(sourceDec[sourceDec.length() - 1] - '0'), "2") + resBin;

sourceDec = divideLargeNumbers(sourceDec, "2");

}

return resBin;

}

const string encryptDecrypt(string m, string power, string n)

{

string modulo = "1";

string binPower = decToBinStr(power);

while (binPower.length() > 1 || binPower[0] != '0')

{

// cout << binPower << endl;

if (binPower[binPower.length() - 1] == '1')

{

modulo = getRemainder(multiplyLargeNumbers(modulo, m), n);

binPower.replace(binPower.length() - 1, 1, "0");

}

else

{

m = getRemainder(multiplyLargeNumbers(m, m), n);

binPower.erase(binPower.length() - 1, 1);

}

}

return modulo;

}

int main(int argc, char \*\*argv)

{

setlocale(LC\_ALL, "");

srand(time(NULL));

string fileNames[10] = {"prime1.txt", "prime2.txt", "prime3.txt", "prime4.txt", "prime5.txt", "prime6.txt", "prime7.txt", "prime8.txt", "prime9.txt", "prime10.txt"};

const int idx1 = rand() % 10;

int idx2 = rand() % 10;

while (idx2 == idx1)

{

idx2 = rand() % 10;

}

const string fileName1 = fileNames[idx1];

const string fileName2 = fileNames[idx2];

string p;

string q;

ifstream readFileHandle;

readFileHandle.open(fileName1);

getline(readFileHandle, p);

readFileHandle.close();

readFileHandle.open(fileName2);

getline(readFileHandle, q);

readFileHandle.close();

string n = multiplyLargeNumbers(p, q);

const string pMinus1 = minusOne(p);

const string qMinus1 = minusOne(q);

string fi = multiplyLargeNumbers(pMinus1, qMinus1);

string e = "65537";

string d = "0";

string dummy = "1";

bool dummyXBool = false;

bool dummyYBool = false;

const string commonRem = gcdExt(e, fi, d, dummy, dummyXBool, dummyYBool);

if (dummyXBool) // fi - d if "d" is negative

{

d = subtractLargeNumbers(fi, d);

}

cout << "p(" << p.length() << "): " << p << endl;

cout << "q(" << q.length() << "): " << q << endl;

cout << "n: " << n << endl;

// cout << "p - 1: " << pMinus1 << endl;

// cout << "q - 1: " << qMinus1 << endl;

cout << "fi: " << fi << endl;

// cout << "(q - 1)/(p - 1): " << divideLargeNumbers(qMinus1, pMinus1) << endl;

// cout << "p + q: " << sumLargeNumbers(p, q) << endl;

// cout << "p - q: " << subtractLargeNumbers(p, q) << endl;

// cout << "p % q: " << getRemainder(p, q) << endl;

cout << "d: " << d << endl;

// cout << "common remainder: " << commonRem << endl;

// cout << "calculated remainder: " << getRemainder(multiplyLargeNumbers(d, e), fi) << endl;

ofstream writeFileHandle;

writeFileHandle.open("public\_key.txt");

writeFileHandle << e << endl

<< n;

writeFileHandle.close();

writeFileHandle.open("private\_key.txt");

writeFileHandle << d << endl

<< n;

writeFileHandle.close();

readFileHandle.open("to\_encrypt.txt");

string readStr;

getline(readFileHandle, readStr);

string encrypted = encryptDecrypt(readStr, e, n);

cout << "Encrypted: " << encrypted << endl;

writeFileHandle.open("encrypted.txt");

writeFileHandle << encrypted << endl;

writeFileHandle.close();

string decrypted = encryptDecrypt(encrypted, d, n);

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

writeFileHandle.open("decrypted.txt");

writeFileHandle << decrypted << endl;

writeFileHandle.close();

return 0;

}