МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

Лабораторна робота № 3

з курсу «Технології захисту інформації»

Тема: «Дослідження криптосистеми шифрування даних RSA»

Виконав:

студент IV курсу

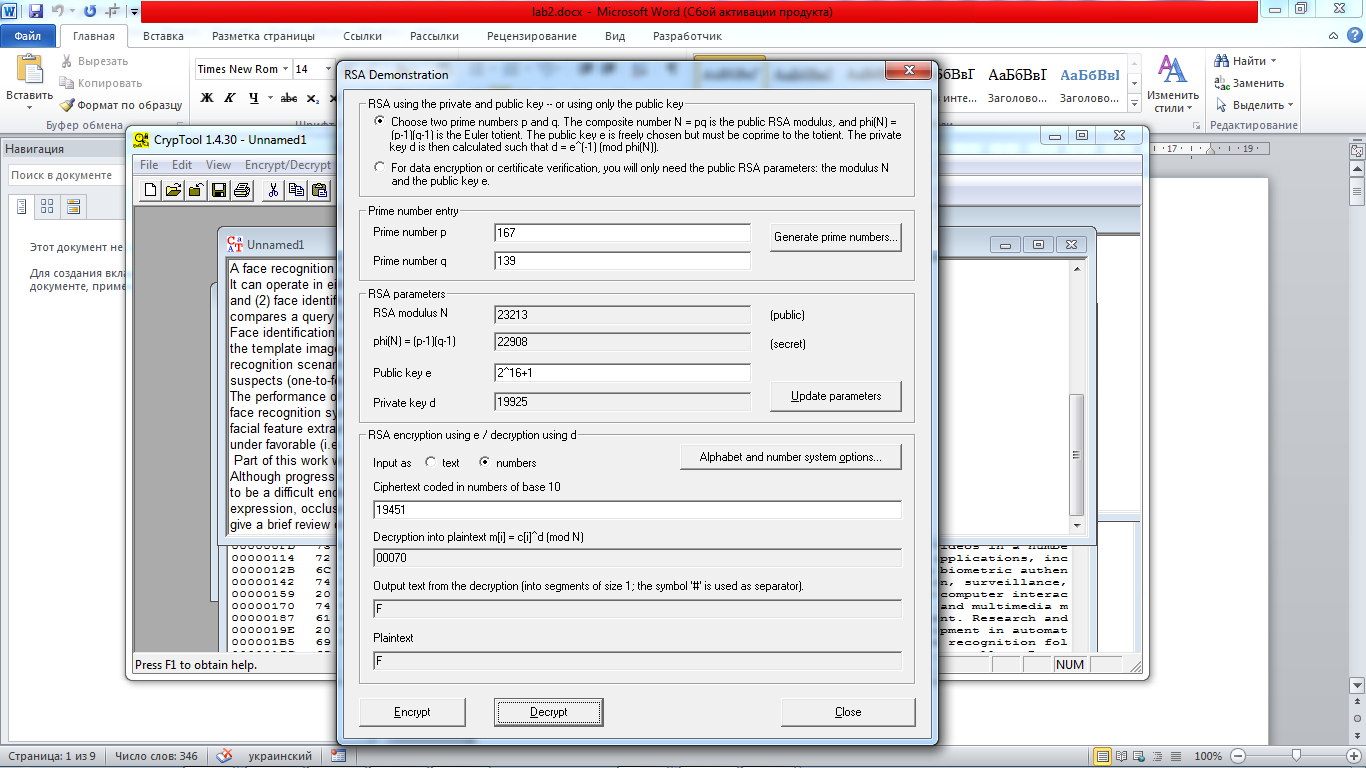
групи ДА-32

Колінько Анжела

Київ – 2017

Варіант 11

Були згенеровані наступні параметри RSA:



1) input= Kolinko Angela Mikhailivna

1.Результати шифрування і дешифрування:

a. ручного:

Шифрування: c[i]=(m[i]^e)mod n

c[0]=K=(ASCII)75; (75^(2^16+1))mod(23213)=(75^65537)mod(23213)=2562;

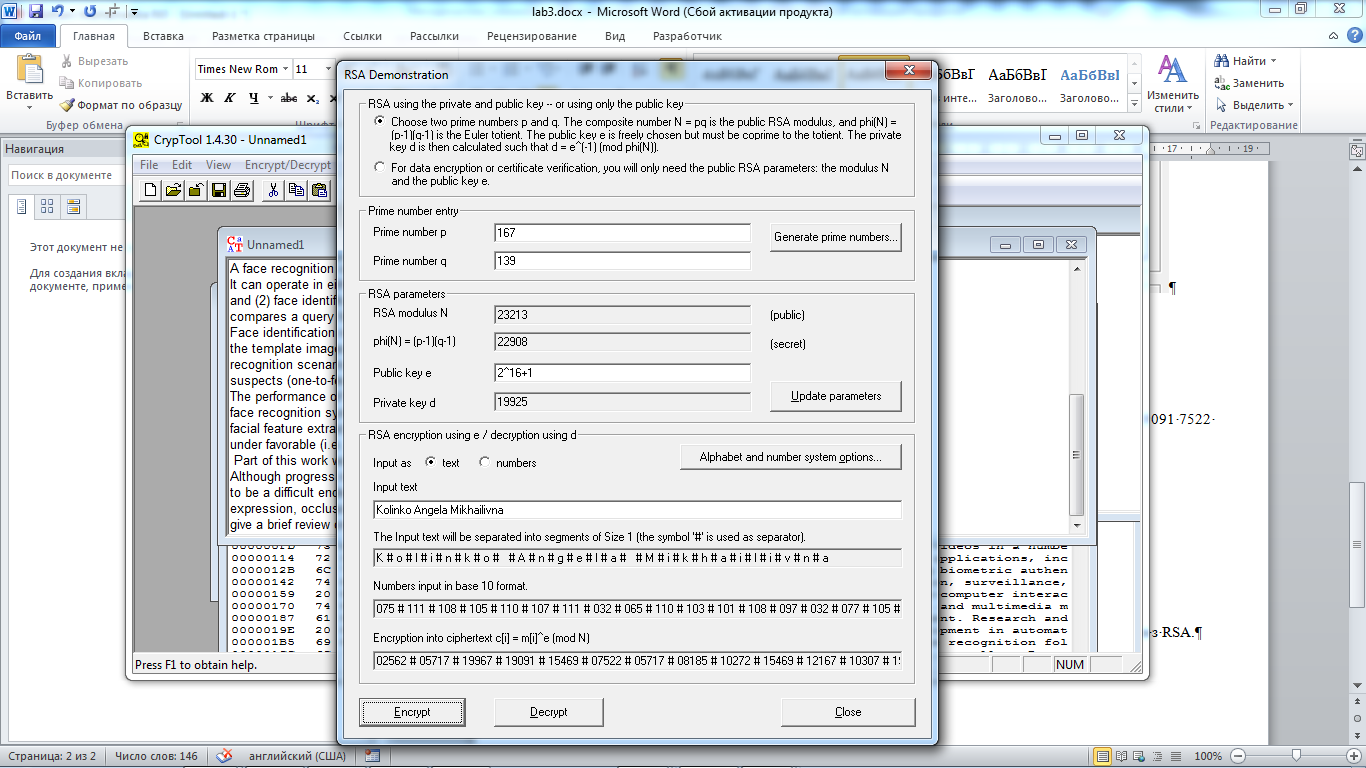
Аналогічно шифруються всі інші літери. В результаті маємо

2562 5717 19967 19091 15469 7522 5717 8185 10272 15469 12167 10307 19967 8939 8185 6245 19091 7522 11178 8939 19091 19967 19091 14069 15469 8939

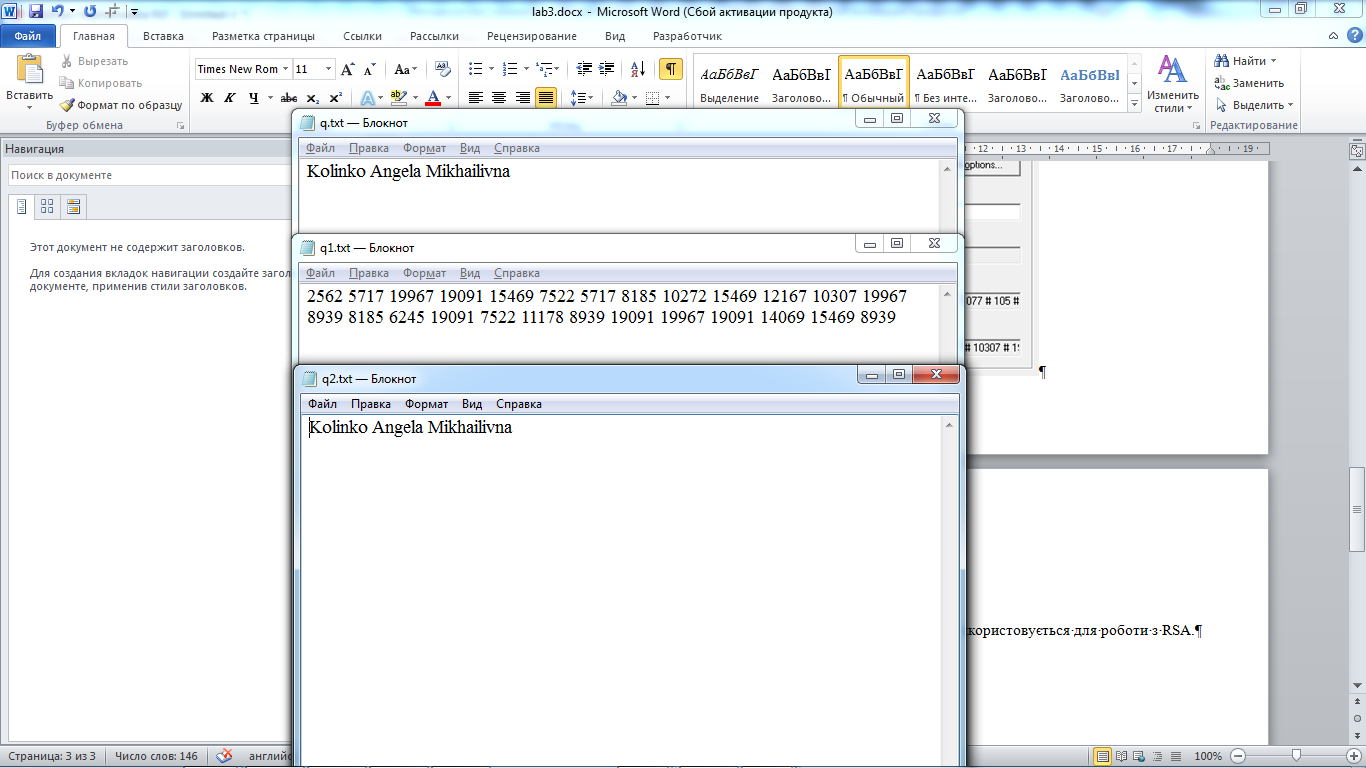
Дешифрування: m[i]=(c[i]^d)mod n

m[0]=(2562^19925)mod(23213)=75=(ASCII)75=K;

b. з використанням пакету CrypTool;



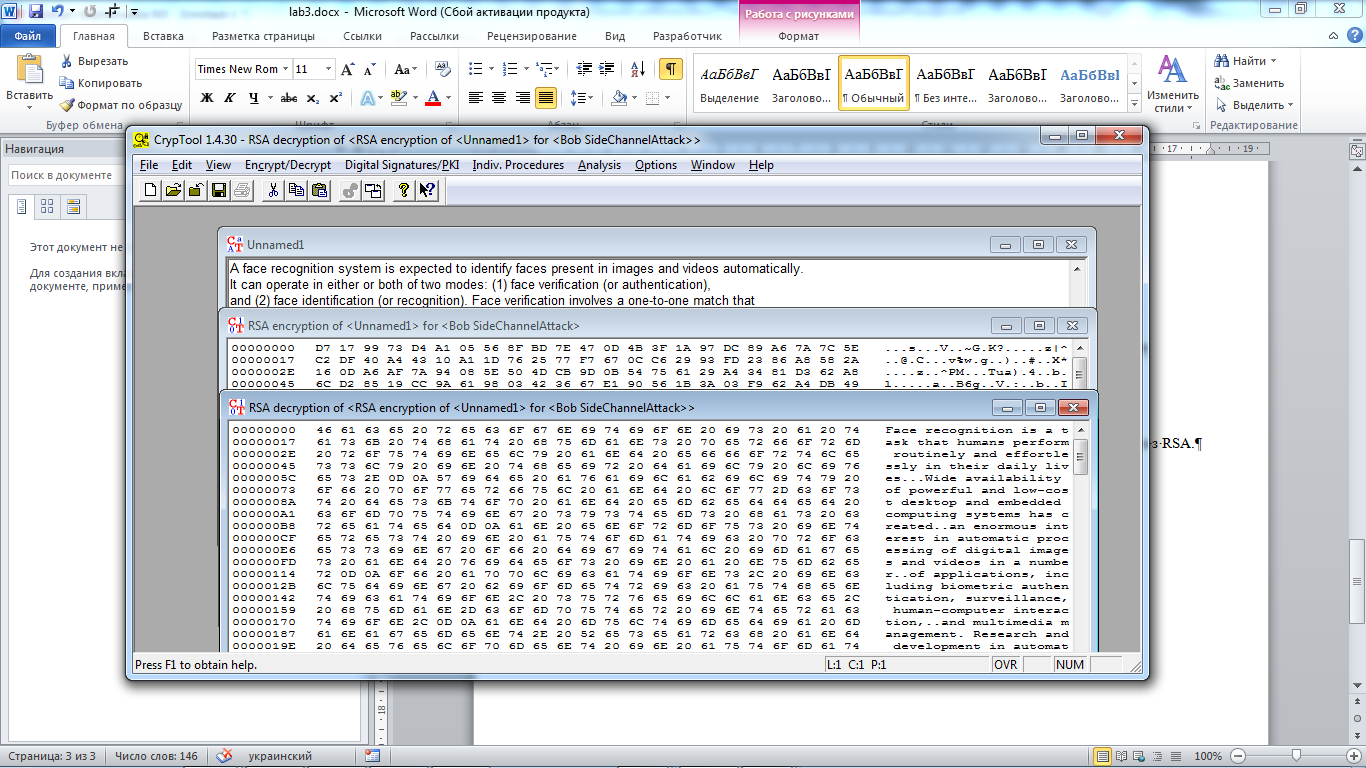
c. використовуючи готові бібліотеки;

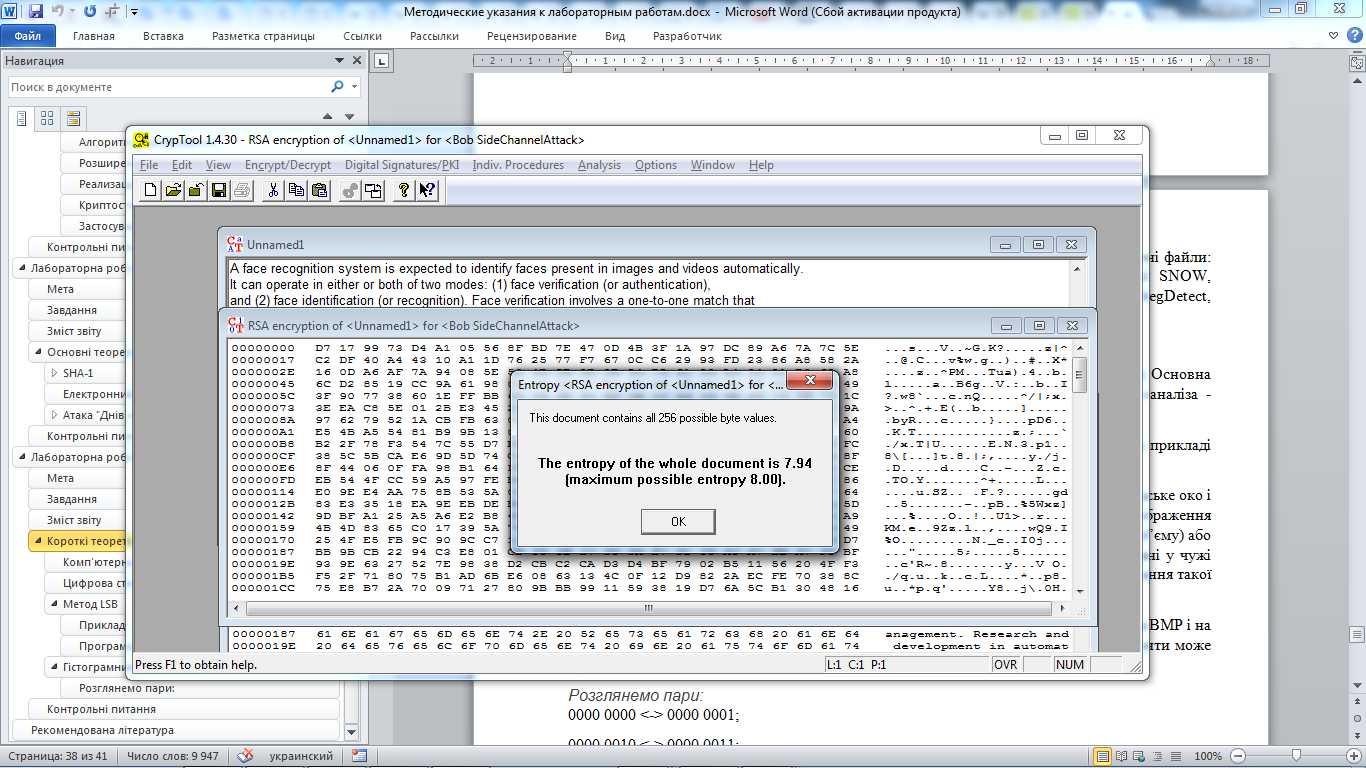


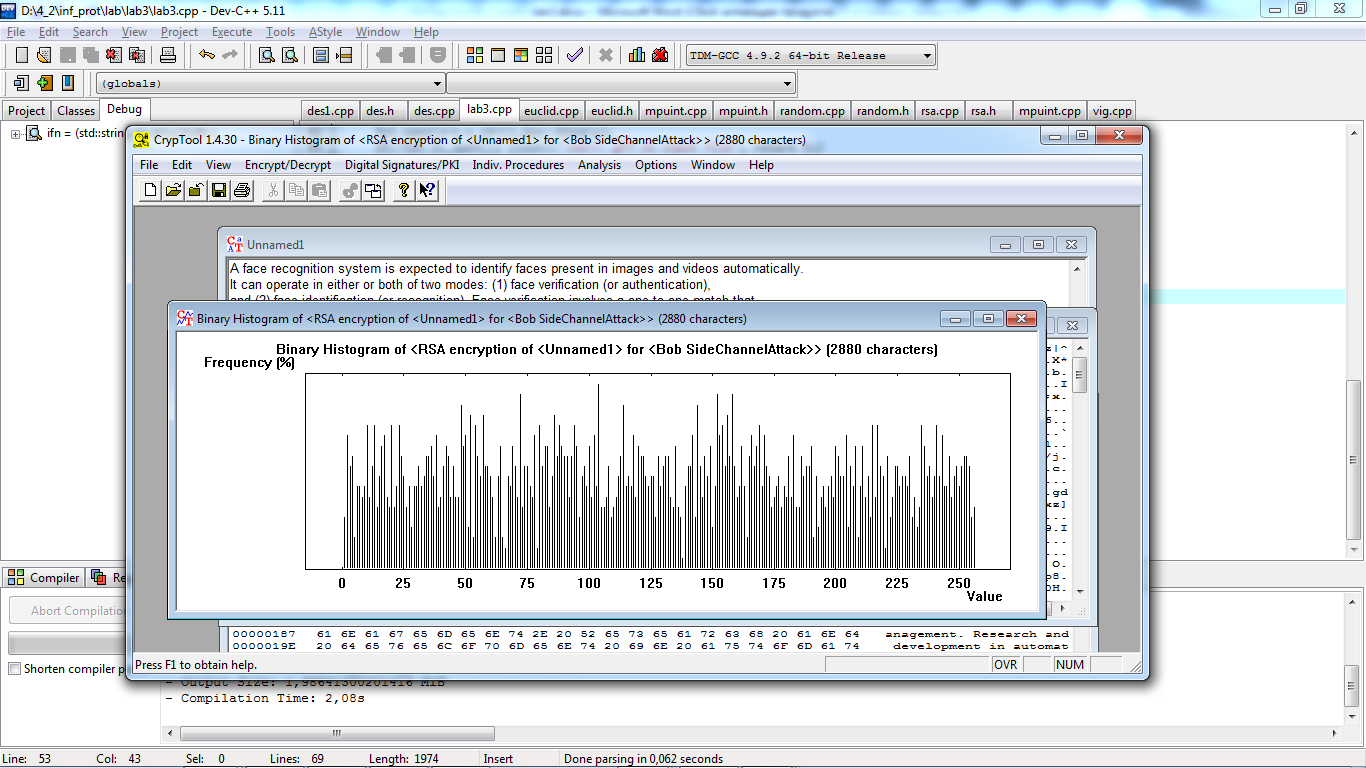
2) input – текст за варіантом

1.Результати шифрування і дешифрування:

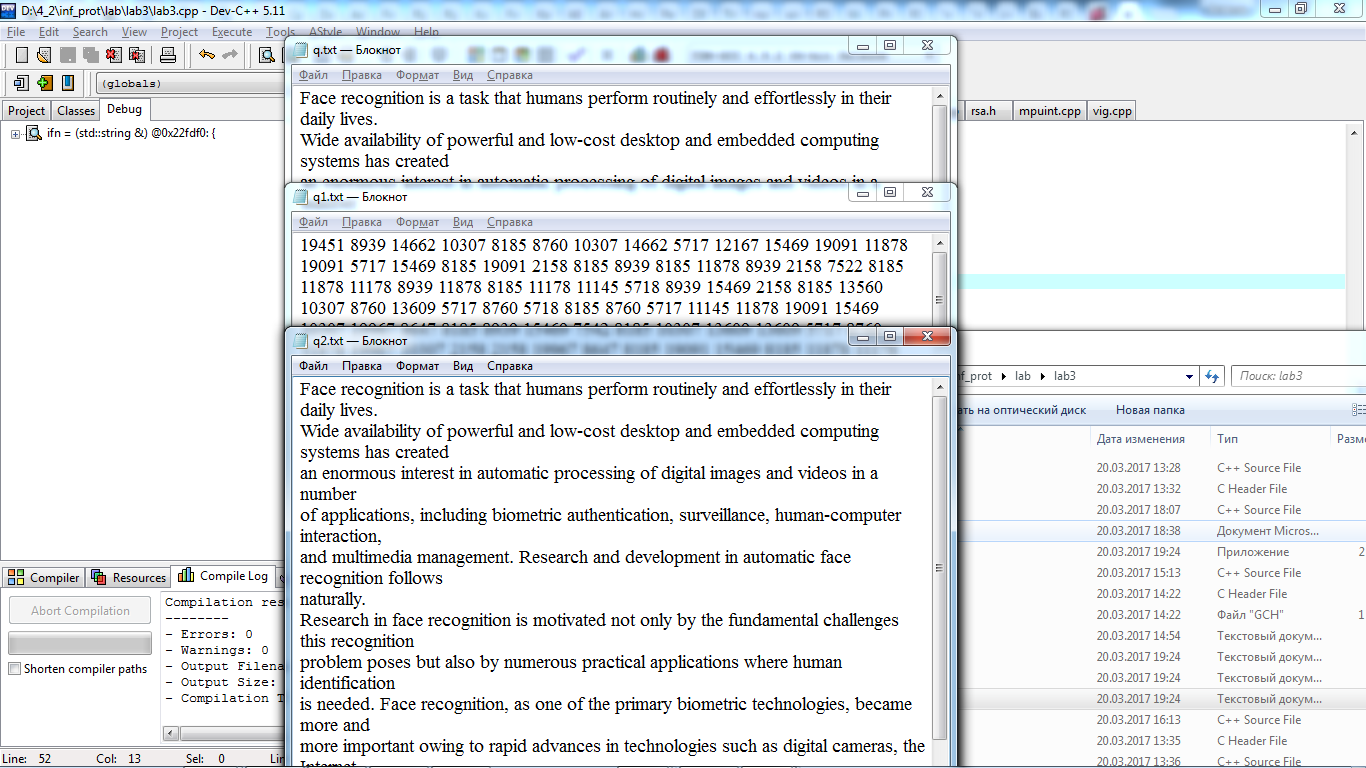
b. з використанням пакету CrypTool;







c. використовуючи готові бібліотеки;



2. Короткий опис синтаксису основних функцій бібліотеки, яка використовується для роботи з RSA.

RSA Public-Key Cryptography needs large integers for reasonable security. The 32-bit or 64-bit integers available on most machines just aren't big enough. Therefore, the RSA Public-Key Cryptography package uses another package, called the Multiple-Precision Unsigned Integer Arithmetic, to do its arithmetic. In this package, the number of bits can be any multiple of 16.

A 512-bit key is considered at least moderately secure; 1024 bits are preferred. The package will, in theory at least, handle any key size which is an even multiple of 16, up to the point where the computer runs out of memory. However, the computations for keys more than 1024 bits long are very slow, even on today's fastest computers.

The package resides in the following files:

EUCLID.CPP

EUCLID.H

MPUINT.CPP

MPUINT.H

RANDOM.CPP

RANDOM.H

RSA.CPP

RSA.H

for your convenience, these files have been combined into a single file RSA.TXT which you can download. Use a text editor to separate the files before compiling them.

When a random number is needed, the package asks the operator to type in a specified number of "random characters". It uses both the characters AND THE TIMES BETWEEN KEYSTROKES to generate a random number. The C library function rand() must NOT be used for this purpose because it produces pseudo-random values that can be reproduced by anyone with the same C compiler. (This function is used to generate test values for the prime test, because these values are not used in the generated keys.)

The random-number generation code is specific to DOS and DOS sessions under Windows.

The following function call generates a key pair. The variables d, e and n must be the same length, which must be an even multiple of 16 bits. One key is (d,n) and the other is (e,n).

GenerateKeys(d, e, n);

mpuint &d;

mpuint &e;

mpuint &n;

It takes several minutes to generate a pair of 512-bit keys, even on a fairly fast computer.

The value of the modulus n is always at least 0x4000..., with enough zeros to fill out all the bits.

The following function call (actually an inline call on a function defined in the Multiple-Precision Unsigned Integer Arithmetic Package) is used to encrypt (or decrypt) a message:

EncryptDecrypt(result, source, e, n);

mpuint &result;

const mpuint &source;

const mpuint &e;

const mpuint &n;

Here e and n are the exponent and modulus, respectively, of the key (either public or private).

The source must be less than the modulus n. If the function GenerateKeys() was used to generate the keys, it is sufficient if the two most significant bits of the source are zero. The result will also be less than the modulus n and can be decrypted without further restriction.

In the communication protocol described above, Alice uses her private key to encrypt the session key, and then uses Bob's public key to encrypt it again (or vice-versa). The order of encryption is important, because in every case the number to be encrypted must be less than the modulus of the key used to encrypt it. The first encryption poses no problem, because session keys are generally much smaller than public keys. However, the encrypted session key may be nearly as large as the modulus, and may be larger than the modulus of the second key. The easiest way to prevent this from happening is to use the key with the smaller modulus first.

The following function, which is used internally by the RSA package to generate random numbers, can also be used to generate a random session key:

Random(x);

mpuint &x;

It will ask the operator to type a specified number of "random characters". If this is not acceptable, you may have to use another method. But make sure the method produces truly random numbers that nobody else will be able to reproduce.

This package is designed to merely illustrate RSA public-key cryptography. Some additional steps are required to make it as secure as it can be. For example, it is desirable to check for "weak keys", special cases in which the product pq can be factored easily.

3. Висновки: на лабораторній роботі було розглянуто криптосистеми шифрування даних RSA за допомогою програмних засобів CryptTool та методики її практичної реалізації.

Додаток 1. Лістинг lab3.cpp

#include<iostream>

#include<string>

#include<fstream>

#include"mpuint.cpp"

#include"rsa.cpp"

#include"random.cpp"

#include"euclid.cpp"

#define pnum 167

#define qnum 139

#define nnum (pnum\*qnum)

#define phinnum ((pnum-1)\*(qnum-1))

#define dnum 19925

#define estrnum "65537"

#define nl 2

using namespace std;

typedef unsigned short int usi;

int main(){

ifstream fpi; string srs="q.txt",dst="q1.txt",ddst="q2.txt";

char sout[3]; //cout<<sizeof(unsigned short int)<<" sizeof(s)="<<sizeof(s)<<'\n';

mpuint x(nl),d(nl),e(nl),n(nl),p(nl),q(nl),phin(nl); const char\* s={estrnum};

p=pnum; q=qnum; n=nnum; phin=phinnum; d=dnum; bool fnd=e.scan(s);

if(fnd==0){

cout<<"\nmpuinterr\n"; system("pause"); return 0;

}

mpuint res(nl\*2),strt(nl\*2);

x.edit(sout); cout<<"\nx="<<sout;

p.edit(sout); cout<<"\tp="<<sout;

q.edit(sout); cout<<"\tq="<<sout;

n.edit(sout); cout<<"\tn="<<sout;

phin.edit(sout); cout<<"\tphin="<<sout;

d.edit(sout); cout<<"\td="<<sout;

e.edit(sout); cout<<"\te="<<sout;

ofstream fpo,fpo1;

fpo.open(dst.c\_str(),ios::binary);

if(!fpo.is\_open()) {cout<<"\nerr: gff no input file"; return 0;}

fpo1.open(ddst.c\_str(),ios::binary);

if(!fpo1.is\_open()) {cout<<"\nerr: gff no input file"; return 0;}

fpi.open(srs.c\_str(),ios::binary);

if(!fpi.is\_open()) {cout<<"\nerr: gff no input file"; return 0;}

else{

unsigned char ch='\0';

while(1){

fpi>>noskipws>>ch;

if(fpi.eof()) break;

x=(usi)ch;

x.edit(sout); cout<<"\nx="<<sout;

EncryptDecrypt(res, x, e, n);

res.edit(sout); cout<<"\nres="<<sout<<'\n';

for(usi i=0;i<res.length;i++){

cout<<res.value[res.length-1-i]<<' ';

}

fpo<<sout<<' ';

EncryptDecrypt(strt, res, d, n);

strt.edit(sout); cout<<"\nptxt="<<sout;

fpo1<<(unsigned char)strt.value[0];

cout<<'\n'; //system("pause");

}

fpi.close();

}

fpo.close(); fpo1.close();

return 0;

}