Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Фізико-технічний інститут

ЛАБОРАТОРНА РОБОТА №5

з дисципліни

«Криптографія»

на тему: «Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем»

Виконали:

студенти 3 курсу ФТІ

групи ФБ-71

Безлюдний В.

Мельник Д.

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

Мета 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 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 якостідатчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабінаіз 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 1 1 p , q довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq ≤ p1q1 ; p і q – прості числа для побудови ключів абонента A, 1 p 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) , (,) 1 n1 e 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 доякої 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 роботупротоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналуза 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. Перевірити роботу програм для випадково обраного ключа $0 < k < n$.

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey().

Труднощі та етапи розробки програмного коду

Перш за все було розроблено функцію перевірки числа на простоту. Для цього був використаний тест Мілера-Рабіна. Першою перешкодою була необхідність швидко піднести число в степінь по модулю, так як звичайна математична операція піднесення числа в степінь довго працює з великими числами. Тому ми створили функцію за схемою Горнера, яка вирішує це питання.

Далі ми отримали пари простих чисел p1, q1, p, q та написали функції за шифрування, розшифрування, підписання ЦП, його перевірки, та функції відправки та отримання ключів. На цих етапах проблем не було.

Хід роботи:

Первый набор:

P1: 13242934204990234010039845000504705036195097277775946398410647976842697988349 9
Q1:205072871744498393625691140029148606985394066532198480890968940705369434233167

P2:12763526428067118108366887875762997318423667087217983460780023042196536659990 3
Q2:12352437731938471177965877020608970194277502987250540240046129787775942303584 7

n1:27157665477407930579066622398684530707088190211332691840079829541368000714762 7
85160403627839952628786442114671433027096613990184492372959977889188061811333

n2:15766066544265012880417124363258369486940618003723222305441407929997506222957 1
96417966785469458581986247801365457680947525040259320817026841521632475722841

e1: 65537
e2: 65537

d1:90671968248647775912928081039049119795199153152291168370963997466266875755636 8
33728863351310488549965640839966804652126921521894057677527503670626461877 57
d2:12263164443908834652061329196354099866736117664186566065614556824992796088033 5
26732813290990462274354040226861641690486787300169133409152396460014437737089

M1:78229198338062314354986704307940498777107181242510095522767953296440420516207
84838222107489880022325156103375285955899111325808654130941510752264219638687
C1 deciphered:
78229198338062314354986704307940498777107181242510095522767953296440420516207848
38222107489880022325156103375285955899111325808654130941510752264219638687

M2:18419319197679411979536271757612642884668552176724223957565578986866417927754
95290410747695311711582425342903404310759422864227774901544379136269674152761 7
C2 deciphered:
18419319197679411979536271757612642884668552176724223957565578986866417927754952
9041074769531171158242534290340431075942286422777490154437913626967415276 17

k::122782399877127271642506539380202622071380622016478247705116671629103632329894312831808184862666613939866879208253526625315825580039865663014820411822214583

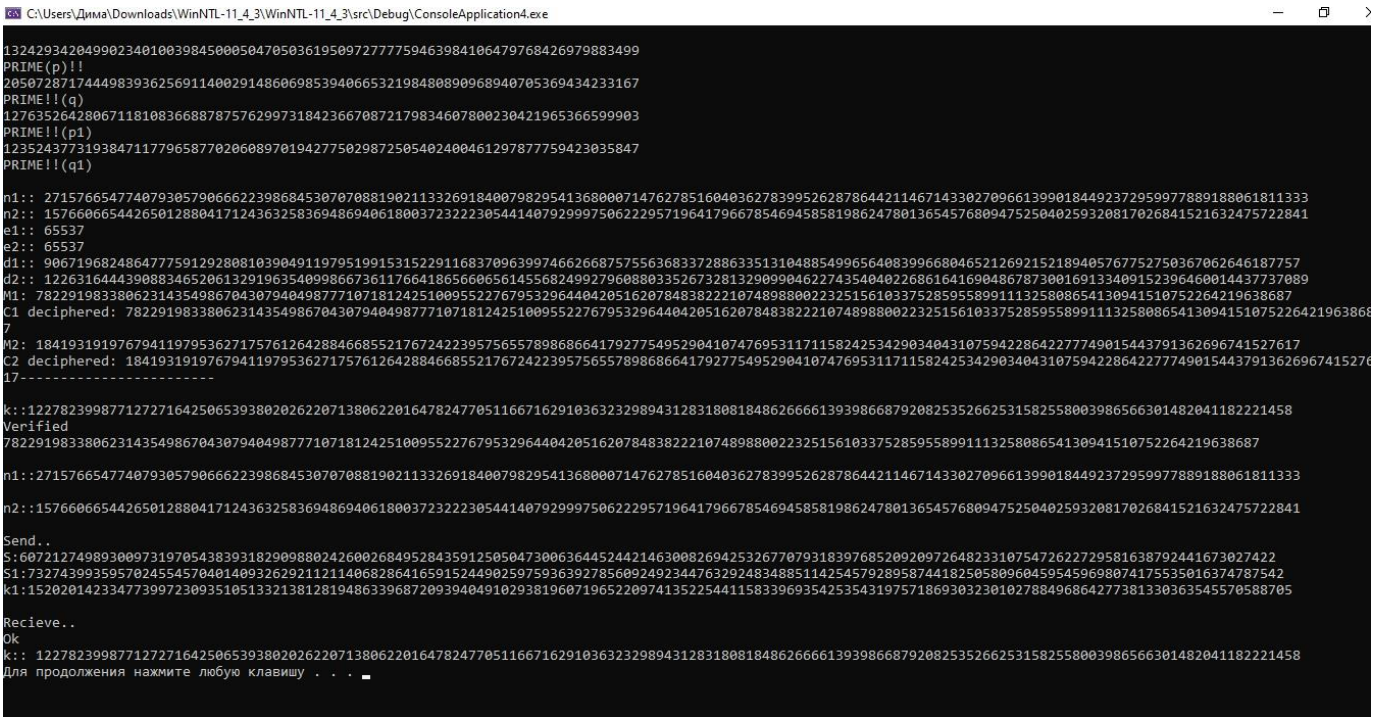
Verified S:
782291983380623143549867043079404987771071812425100955227679532964404205162078483822210748988002232515610337528595589911132580865413094151075226421963868 7

Key exchange:

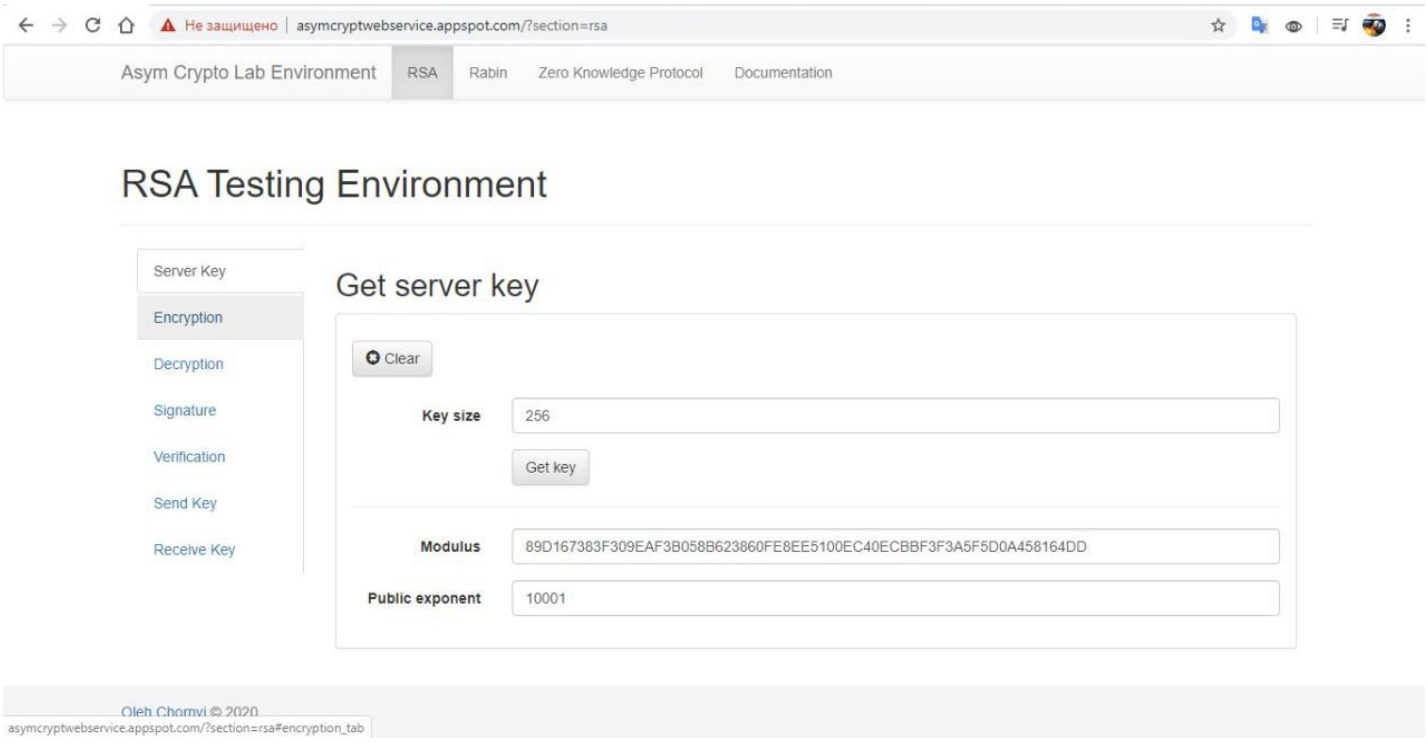
S:607212749893009731970543839318290988024260026849528435912505047300636445244214630
082694253267707931839768520920972648233107547262272958163879244167302742 2

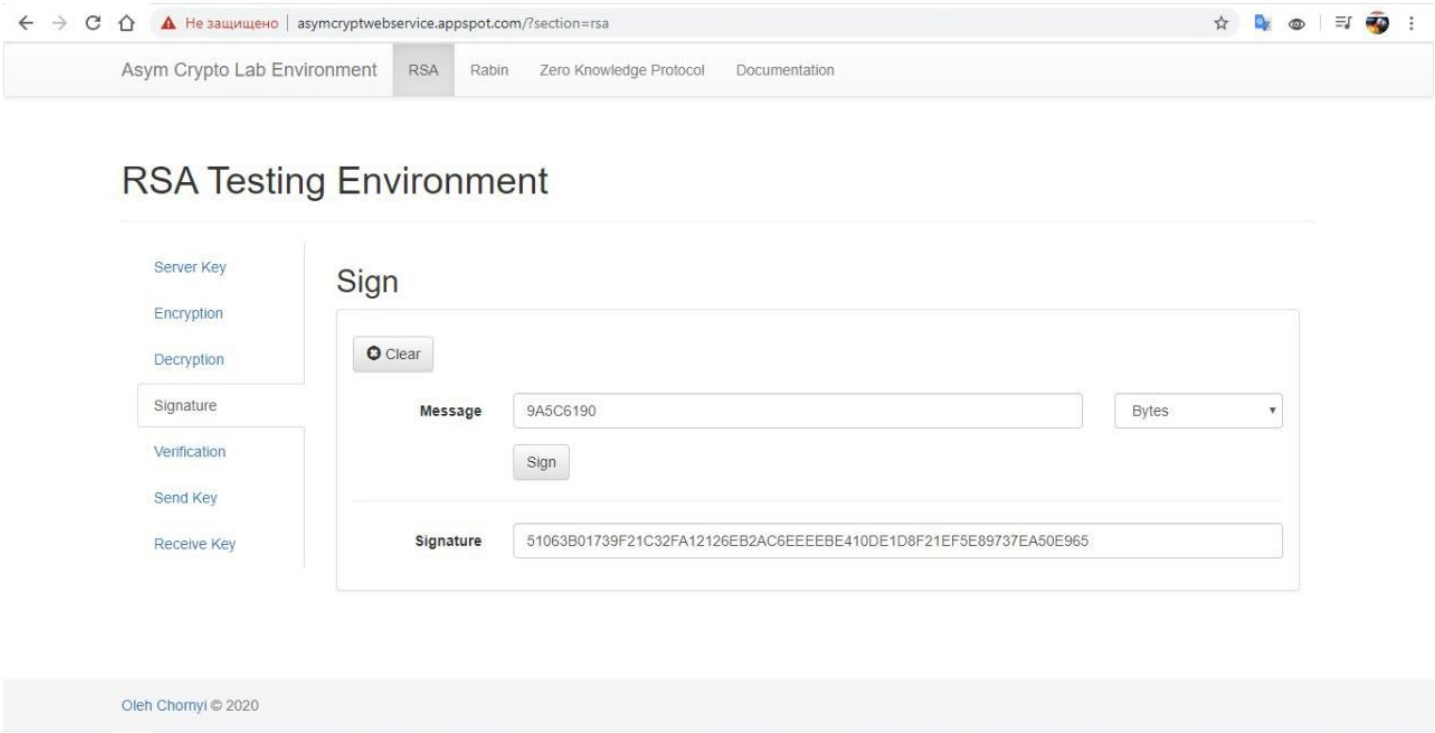
S1:732743993595702455457040140932629211211406828641659152449025975936392785609249234476329248348851142545792895874418250580960459545969807417553501637478754 2

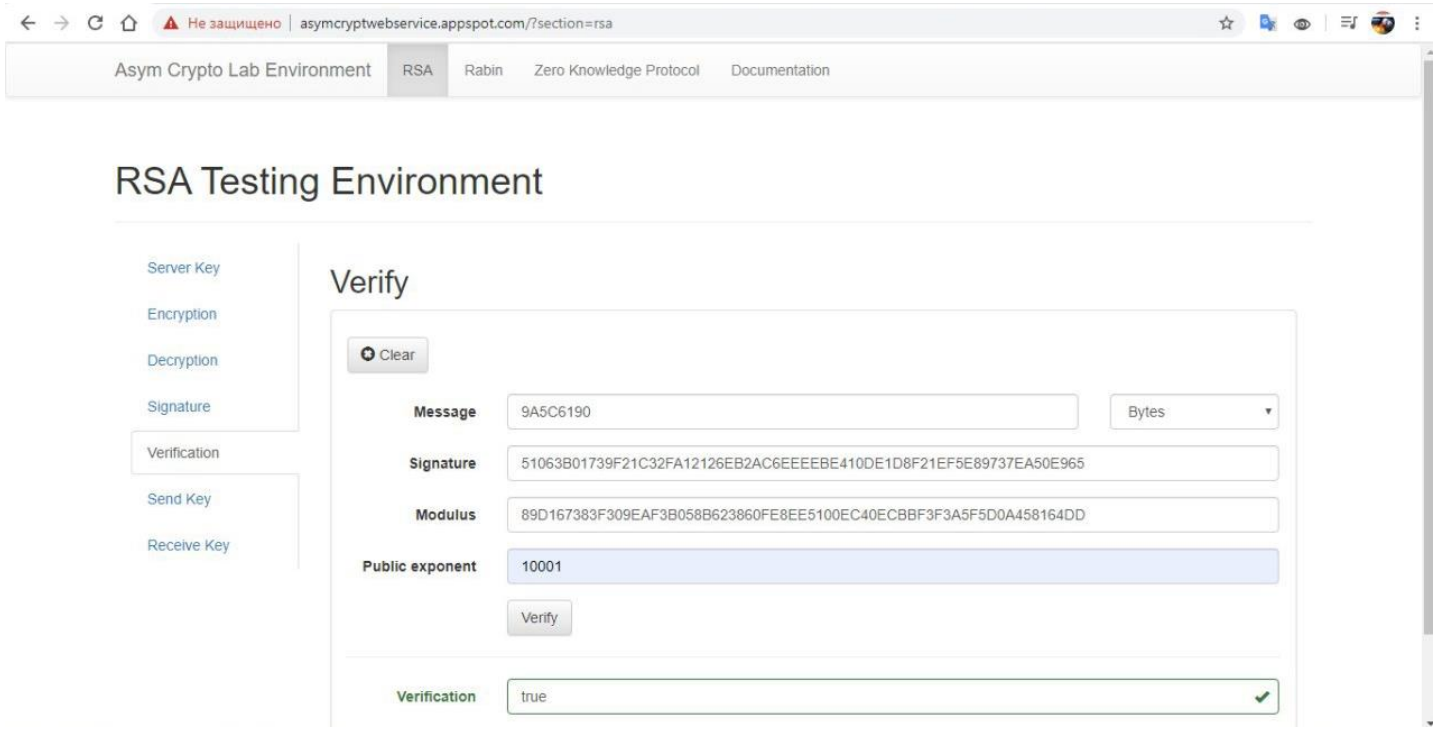
k1:152020142334773997230935105133213812819486339687209394049102938196071965220974135225441158339693542535431975718693032301027884968642773813303635455705887 05

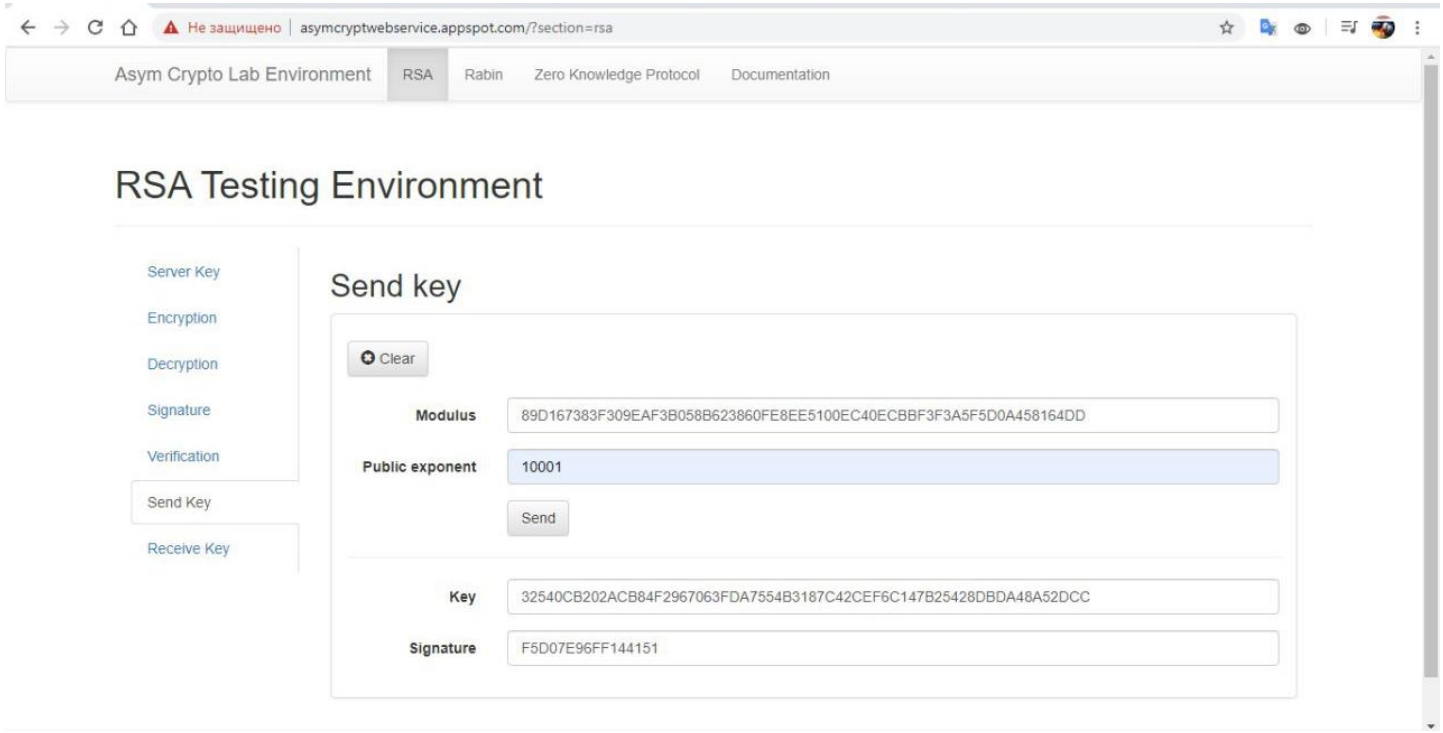


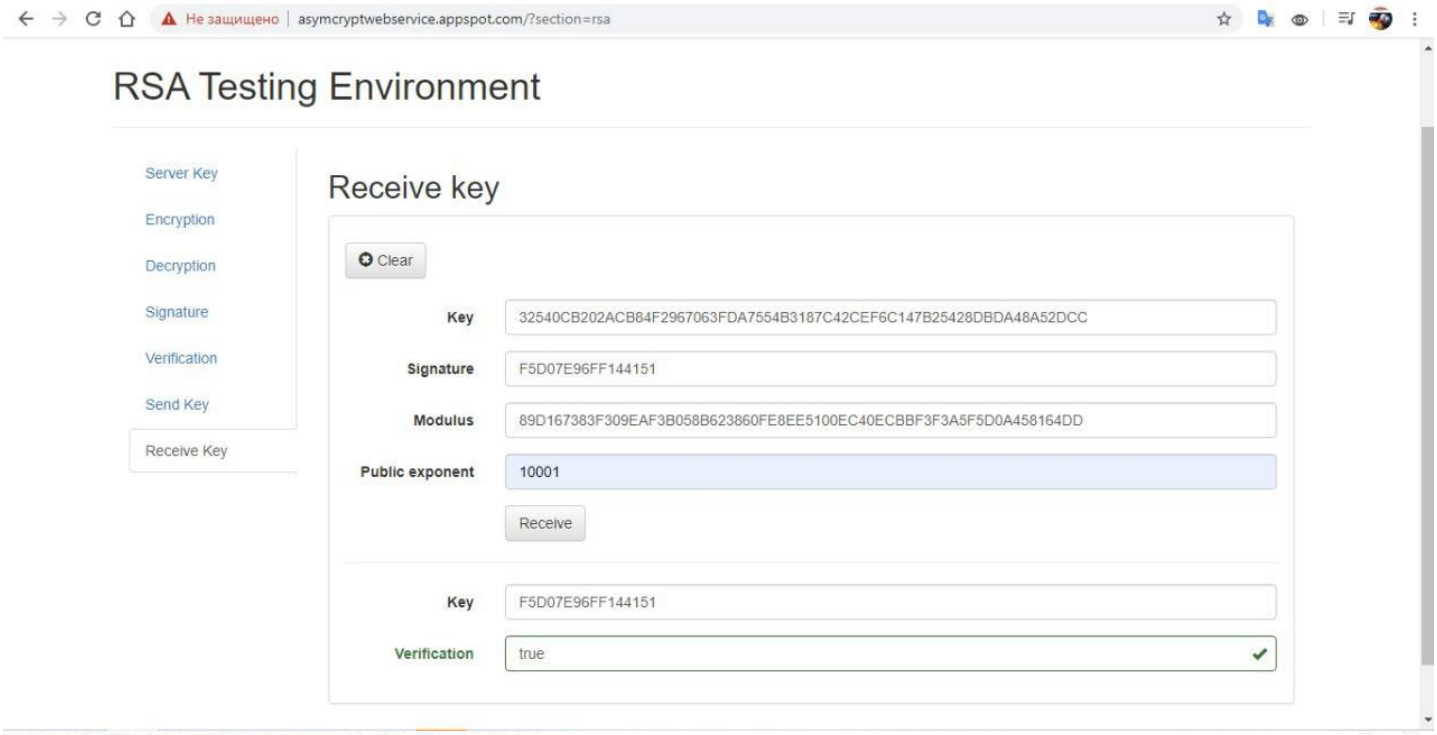
Робота з сайтом:

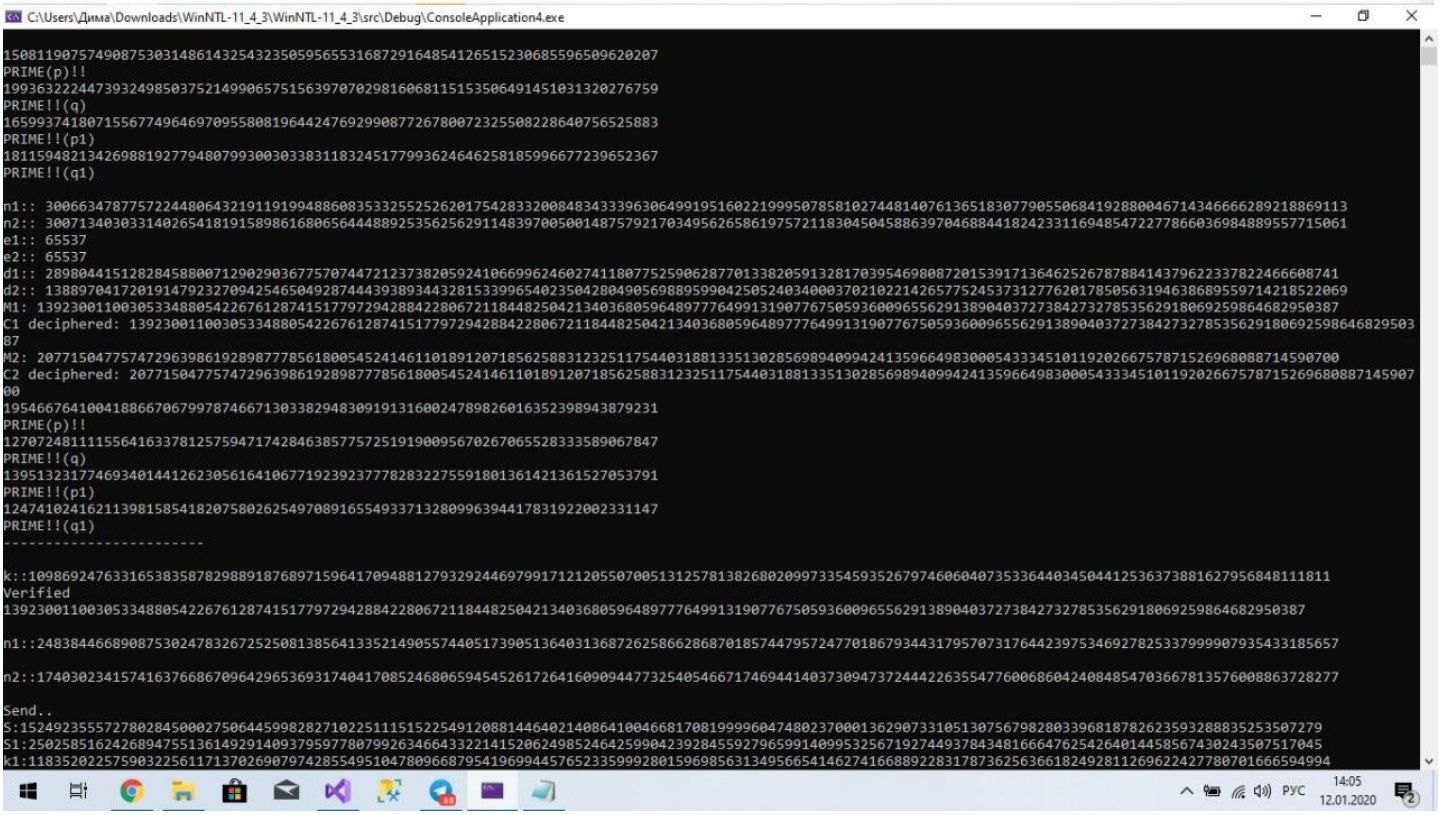


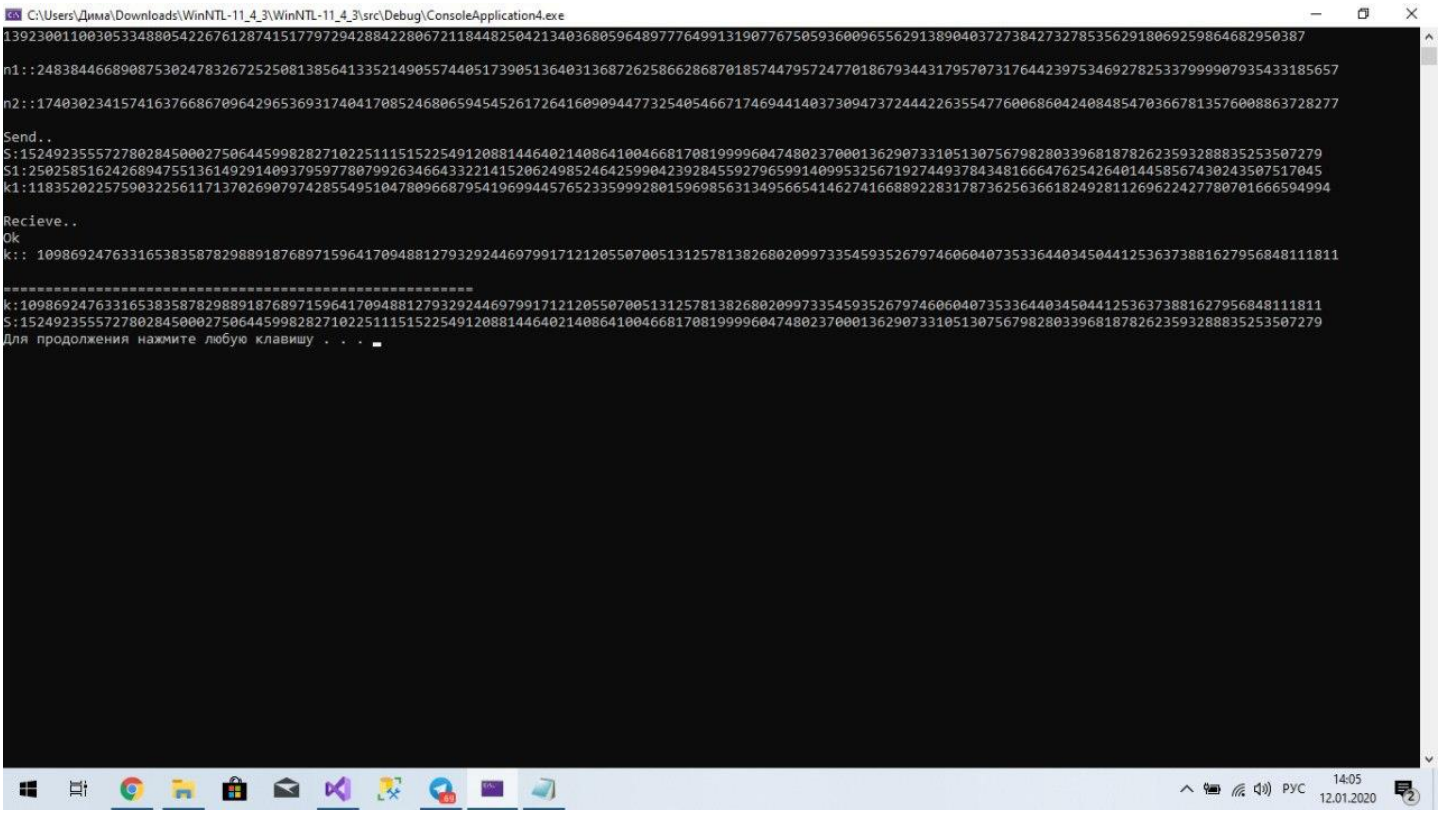



Не защищено | asymcryptwebservice.appspot.com/?section=rsa
Asym Crypto Lab Environment
RSA
Rabin
Zero Knowledge Protocol
Documentation
RSA Testing Environment
Server Key
Encryption
Decryption
Signature
Verification
Send Key
Receive Key
Verify
Clear
Message
9A5C6190
Bytes
Signature
51063B01739F21C32FA12126EB2AC6EEEEBE410DE1D8F21EF5E89737EA50E965
Modulus
89D167383F309EAF3B058B623860FE8EE5100EC40ECBBF3F3A5F5D0A458164DD
Public exponent
10001
Verify
Verification
true


Не защищено | asymcryptwebservice.appspot.com/?section=rsa
Asym Crypto Lab Environment
RSA
Rabin
Zero Knowledge Protocol
Documentation
RSA Testing Environment
Server Key
Encryption
Decryption
Signature
Verification
Send Key
Receive Key
Send key
Clear
Modulus
89D167383F309EAF3B058B623860FE8EE5100EC40ECBBF3F3A5F5D0A458164DD
Public exponent
10001
Send
Key
32540CB202ACB84F2967063FDA7554B3187C42CEF6C147B25428DBDA48A52DCC
Signature
F5D07E96FF144151


Не защищено | asymcryptwebservice.appspot.com/?section=rsa
RSA Testing Environment
Server Key
Encryption
Decryption
Signature
Verification
Send Key
Receive Key
Receive key
Clear
Key
32540CB202ACB84F2967063FDA7554B3187C42CEF6C147B25428DBDA48A52DCC
Signature
F5D07E96FF144151
Modulus
89D167383F309EAF3B058B623860FE8EE5100EC40ECBBF3F3A5F5D0A458164DD
Public exponent
10001
Receive
Key
F5D07E96FF144151
Verification
true


C:\Users\Дима\Downloads\WinNTL-11_4_3\WinNTL-11_4_3\src\Debug\ConsoleApplication4.exe
PRIME(p)!!
PRIME!!(q)
PRIME!!(p1)
PRIME!!(q1)
n1:: 3006634787757224480643219119199488608353325525262017542833200848343339630649919516022199950785810274481407613651830779055068419288004671434666628921886911 3
n2:: 3007134030331402654181915898616806564448892535625629114839700500148757921703495626586197572118304504588639704688441824233116948547227786603698488955771506 1
e1:: 65537
e2:: 65537
d1:: 2898044151282845880071290290367757074472123738205924106699624602741180775259062877013382059132817039546980872015391713646252678788414379622337822466608741
d2:: 1388970417201914792327094254650492874443938934432815339965402350428049056988959904250524034000370210221426577524537312776201785056319463868955971421852206 9
M1: 139230011003053348805422676128741517797294288422806721184482504213403680596489777649913190776750593600965562913890403727384273278535629180692598646829503 87
C1 deciphered: 139230011003053348805422676128741517797294288422806721184482504213403680596489777649913190776750593600965562913890403727384273278535629180692598646829503 87
M2: 207715047757472963986192898777856180054524146110189120718562588312325117544031881335130285698940994241359664983000543334510119202667578715269680887145907 00
C2 deciphered: 207715047757472963986192898777856180054524146110189120718562588312325117544031881335130285698940994241359664983000543334510119202667578715269680887145907 00
PRIME(p)!!
PRIME!!(q)
PRIME!!(p1)
PRIME!!(q1)
---
k::109869247633165383587829889187689715964170948812793292446979917121205507005131257813826802099733545935267974606040735336440345044125363738816279568481118 11
Verified
1392300110030533488054226761287415177972942884228067211844825042134036805964897776499131907767505936009655629138904037273842732785356291806925986468295038 7
n1::2483844668908753024783267252508138564133521490557440517390513640313687262586628687018574479572477018679344317957073176442397534692782533799990793543318565 7
n2::1740302341574163766867096429653693174041708524680659454526172641609094477325405466717469441403730947372444226355477600686042408485470366781357600886372827 7
Send..
S:152492355572780284500027506445998282710225111515225491208814464021408641004668170819999604748023700013629073310513075679828033968187826235932888352535072 79
S1:250258516242689475513614929140937959778079926346643322141520624985246425990423928455927965991409953256719274493784348166647625426401445856743024350751704 5
k1:118352022575903225611713702690797428554951047809668795419699445765233599928015969856313495665414627416688922831787362563661824928112696224277807016665949 94



Код:

```
#include <NTL/ZZ.h>
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <iomanip>
#include <cmath>
NTL_CLIENT

using namespace std;


ZZ e = pow(2, 16) + (ZZ)1;
long o = 0;

ZZ pow(int a, int b) {
	ZZ temp;
	temp = a;
	for (int i = 1; i < b; i++)
		temp = temp * a;
	return temp;
}

ZZ pow(ZZ a, ZZ b) {
	ZZ temp;
	temp = a;
	for (ZZ i = (ZZ)1; i < b; i++)
		temp = temp * a;
	return temp;
}
```

```
ZZ rando(ZZ min) {
	return RandomBnd(min) + min;

}

ZZ mod(ZZ k, ZZ m)
{
	if (k < m)
	{
		if (k < 0) { for (;;) { k += m; if (k > 0) return k; } }
		return k;
	}
	else {
		for (;;)
		{
			k = k - m;
			if (k < m)return k;
		}
		return k;
	}
}


ZZ gcdExtended(ZZ a, ZZ b, ZZ* x, ZZ* y)
{

	if (a == 0)
	{
		*x = 0, * y = 1;
		return b;
	}
	/*Change all data types*/
	ZZ x1, y1;//
	ZZ gcd = gcdExtended(b % a, a, &x1, &y1);//


	*x = y1 - (b / a) * x1;
	*y = x1;

	return gcd;
}
ZZ modInverse(ZZ a, ZZ m)
{
	ZZ x, y;
	ZZ g = gcdExtended(a, m, &x, &y);
	if (g != 1)
	{

		return (ZZ)0;
	}
	else
	{

		ZZ res = (x % m + m) % m;
		return res;
	}
}


bool test(/*Change data type*/ZZ p)
{
```

```
int count = 0;
/*Change data type*/ZZ temp = p - 1, x , x1, x2, x0;
x = 2;
```

```
	while (1)
	{
		if (temp % 2 == 0) { temp = temp / 2; count++; }
		else break;
	}



	x = RandomBnd(p);


	if (gcdExtended(x, p, &x1, &x2) != 1)return 0;
	ZZ power;

	x0 = PowerMod(x, temp,p);


	if (x0 == 1 || x0 == -1)return 1;
	for (int i = 1; i < count; i++)
	{


		x0 = PowerMod(x0 , (ZZ)2 , p) ;
		if (x0 == -1) return 1;
		if (x0 == 1) return 0;
	}
	return 0;

}


bool find(ZZ &p) {
	ZZ min;
	bool chk = 0;

	min = pow(2, 256);
	p = rando(min);




	//cout << p << "\n";

	int i = 0, k = 4;//your value

	ZZ arr[5] = { (ZZ) 2,(ZZ)3, (ZZ)5 , (ZZ)7 , (ZZ)11 };
	for (int i = 0; i < 5; i++)
		if (p % arr[i] == 0) { return 0; }



	while (i < k) {
		if (test(p) == 1)i++;
		else break;

	}
	if (i >= k)chk = 1;


	if(chk)return 1;
	else return 0;
```

```
}

ZZ GenerateKeyPair(ZZ p, ZZ q, ZZ &n) {
	ZZ d, e;

	n = p * q;
	e = pow(2, 16) + 1;
	d = modInverse(e, (p - 1) * (q - 1));
	return d;
}


ZZ Encrypt(ZZ e, ZZ n, ZZ M)
{
	ZZ C = PowerMod(M, e, n);
	return C;
}

ZZ Decrypt(ZZ C, ZZ d, ZZ n)
{
	ZZ M = PowerMod(C, d, n);
	return M;
}

ZZ Sign(ZZ M, ZZ d, ZZ n)
{
	ZZ S = PowerMod(M, d, n);
	return S;
}
ZZ Verify(ZZ S, ZZ e, ZZ n)
{
	ZZ M;
	M = PowerMod(S, e, n);
	return M;
}


void Send(ZZ &S,ZZ &S1, ZZ &k1,ZZ k,ZZ d, ZZ n, ZZ n1, ZZ e1)
{
	S = PowerMod(k, d, n);
	S1 = PowerMod(S, e1, n1);
	k1 = PowerMod(k, e1, n1);
	cout << "S:" << S << endl;
	cout << "S1:" << S1 << endl;
	cout << "k1:" << k1 << endl;
}

void Recieve(ZZ &S, ZZ &S1,ZZ k1,ZZ d1, ZZ n1 ,ZZ k, ZZ e, ZZ n)
{
	S = PowerMod(S1, d1, n1);
	k = PowerMod(k1, d1, n1);
	if(k == PowerMod(S , e, n))cout<<"\nOk";
	else cout<< "\nNot ok";
	cout << "\nk:: "<<k<<endl;
}


int main()
{

	ZZ p, q,p1, q1, e1 ,e2, n1 ,n2, d1, d2, C1 , M1, C2, M2, k1, S1, k ,S;
	while (1) {
		if (find(p)) {
			cout << endl << p << "\nPRIME(p)!!"; break;
```

```
		}
	}
	while (1) {
		if (find(q)) {
			cout << endl << q << "\nPRIME!!(q)"; break;
		}
	}
	while (1) {
		if (find(p1)) {
			cout << endl << p1 << "\nPRIME!!(p1)"; break;
		}
	}
	while (1) {
		if (find(q1)) {
			cout <<endl<< q1 << "\nPRIME!!(q1)\n"; break;
		}
	}
	e1 = ::e;
	e2 = ::e;

	d1 = GenerateKeyPair(p, q, n1);
	d2 = GenerateKeyPair(p1, q1, n2);

	cout << "\nn1:: " << n1;
	cout << "\nn2:: " << n2;
	cout << "\ne1:: " << e1;
	cout << "\ne2:: " << e2;
	cout << "\nd1:: " << d1;
	cout << "\nd2:: " << d2;

	M1 = (ZZ)RandomBnd(n2 - 1);
	C1 = Encrypt(e2, n2, M1);
	M2 = (ZZ)RandomBnd(n1 - 1);
	C2 = Encrypt(e1, n1, M2);


	//User2
	cout << "\nM1: " << M1 << endl;
	cout << "C1 deciphered: " << Decrypt(C1, d2, n2);

	//User1
	cout << "\nM2: " << M2 << endl;
	cout << "C2 deciphered: " << Decrypt(C2, d1, n1);

//----------------------------

	while (n2 > n1)
	{

		while (1) {
			if (find(p)) {
				cout << endl << p << "\nPRIME(p)!!"; break;
			}
		}
		while (1) {
			if (find(q)) {
				cout << endl << q << "\nPRIME!!(q)"; break;
			}
		}
		while (1) {
			if (find(p1)) {
				cout << endl << p1 << "\nPRIME!!(p1)"; break;
			}
		}
```

```
		while (1) {
			if (find(q1)) {
				cout << endl << q1 << "\nPRIME!!(q1)\n"; break;
			}
		}
		d1 = GenerateKeyPair(p, q, n1);
		d2 = GenerateKeyPair(p1, q1, n2);
	}
	cout << " ----------------------" << endl;
	k = RandomBnd(n1 - 2) + (ZZ)1;// Нехай
	cout << endl << "k::" << k << endl;

	S = Sign(M1, d1, n1);

	if (M1 == Verify(S, e1, n1)) {cout << "Verified\n"; cout << Verify(S, e1, n1) << endl;}
	else{ cout << "Not verified\n" << Verify(S, e1, n1) << endl;}

	///////
	cout << "\nn1::"<<n1<<endl;
	cout << "\nn2::" << n2 << endl;


	//k1 = (ZZ)44;
	cout << "\nSend..\n";
	Send(S, S1, k1, k, d1, n1, n2, e2);
	cout << "\nRecieve..";
	Recieve(S, S1, k1, d2, n2, k, e1, n1);



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

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.