Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Основы Защиты Информации**

**Проект информационной безопасности информационной системы банковской компании»**

Студент: Круглик А.В.

ФИТ 2 курс 5 группа

Преподаватель: Берников В.О.

# Практическое занятие №6

**Тема «Криптографическая защита информации»**

**Цель:** Овладение основными криптографическими алгоритмами асимметричного шифрования.

**Теоретическое введение**

Реализация элементов криптосистемы RSA

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Весь алгоритм расписан в таблице:

Таблица 6.1 – Алгоритм RSA

|  |  |  |
| --- | --- | --- |
| Этап | Описание операции | Результат операции |
| Генерация ключей | Выбрать два простых различных числа | p=3557,  q=2579 |
| Вычислить модуль (произведение) | n = p \cdot q = 3557 \cdot 2579 = 9173503 |
| Вычислить функцию Эйлера | \varphi(n) = (p-1) (q-1) = 9167368 |
| Выбрать открытую экспоненту | e = 3 |
| Вычислить секретную экспоненту | d = e^{-1} \mod \varphi(n)  d = 6111579 |
| Опубликовать открытый ключ | \{e, n\} = \{3,9173503 \} |
| Сохранить закрытый ключ | \{d, n\} = \{6111579, 9173503 \} |
| Шифрование | Выбрать текст для зашифровки | m = 111111 |
| Вычислить шифротекст | \begin{align} c &= E(m) \\  &= m^e \mod n \\  &= 111111^3   \mod 9173503 \\  &= 4051753 \end{align} |
| Расшифрование | Вычислить исходное сообщение | \begin{align} m &= D(c) = \\   &= c^d \mod n \\   &= 4051753^{6111579} \mod 9173503 \\   &= 111111 \end{align} |

**Реализация элементов схемы шифрования Эль-Гамаля**

**Генерация ключей**

1. Генерируется случайное простое число ~p длины ~n [битов](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82).
2. Выбирается случайный примитивный элемент ~g.
3. Выбирается случайное целое число ~x такое, что ~1 < x < p-1.
4. Вычисляется ~y = g^x\,\bmod\,p.
5. Открытым ключом является тройка \left( p,g,y \right), закрытым ключом — число ~x.

**Шифрование**

Сообщение ~M шифруется следующим образом:

1. Выбирается сессионный [ключ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) — случайное целое число ~k такое, что ~1 < k < p - 1
2. Вычисляются числа a = g^k\,\bmod\,p и b = y^k M\,\bmod\,p.
3. Пара чисел \left( a, b \right) является [шифротекстом](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82" \o "Шифротекст).

Нетрудно видеть, что длина шифротекста в схеме Эль-Гамаля длиннее исходного сообщения M вдвое.

**Расшифрование**

Зная закрытый ключ ~x, исходное сообщение можно вычислить из шифротекста \left( a, b \right) по формуле:

M = b(a^x)^{-1}\,\bmod\,p.

При этом нетрудно проверить, что

~(a^x)^{-1}\equiv g^{-kx}\pmod{p}

и поэтому

~b(a^x)^{-1}\equiv (y^kM)g^{-xk}\equiv (g^{xk}M) g^{-xk}\equiv M \pmod{p}.

Для практических вычислений больше подходит следующая формула:

M = b(a^x)^{-1}\,\bmod\,p = b \cdot a^{(p-1-x)}\,\bmod\,p **.**

**Пример**

**Шифрование**

Допустим, что нужно зашифровать сообщение ~M=5.

Произведем генерацию ключей :

пусть ~p=11, g=2. Выберем ~x=8 - случайное целое число ~x такое, что ~1 < x < p.

Вычислим ~y= g^x\bmod{p}=2^8\bmod{11}=3.

Итак, открытым является тройка ~(p,g,y)=(11,2,3), а закрытым ключом является число ~x=8.

Выбираем случайное целое число ~k такое, что 1 < k < (p − 1). Пусть ~k=9.

Вычисляем число ~a=g^k\bmod{p}=2^9 \bmod{11}=512 \bmod{11}=6.

Вычисляем число ~b=y^k M\bmod{p}=3^9 5 \bmod{11}=19683 \cdot 5 \bmod{11}=9.

Полученная пара ~(a,b)=(6,9) является шифротекстом.

Расшифрование

Необходимо получить сообщение ~M=5 по известному шифротексту ~(a,b)=(6,9) и закрытому ключу ~x=8.

Вычисляем M по формуле : ~M=b(a^x)^{-1}\bmod{p}=9(6^8)^{-1}\mod{11}=5

Получили исходное сообщение ~M=5.

## **Реализация элементов схемы шифрования Дифи-Хеллмана**

**Генерация ключей**

В 1976 году после публичной критики алгоритма DES и указания на сложность обработки секретных ключей Уитфилд Диффи (WhitfieldDiffie) и Мартин Хеллман (MartinHellman) опубликовали свой алгоритм обмена ключами. Это была первая публикация на тему криптографии с открытым ключом и, возможно, самый большой шаг вперед в области криптографии, сделанный когда‑либо.

Из‑за невысокого быстродействия, свойственного асимметричным алгоритмам, алгоритм Диффи‑Хеллмана не предназначен для шифрования данных. Он был ориентирован на передачу секретных ключей DES, ARS или других подобных алгоритмов через небезопасную среду. В большинстве случаев алгоритм Диффи‑Хеллмана не используется для шифрования сообщений, потому что он, в зависимости от реализации, от 10 до 1000 раз медленнее алгоритма DES.

До алгоритма Диффи‑Хеллмана было сложно совместно использовать зашифрованные данные из‑за проблем хранения ключей и передачи информации. В большинстве случаев передача информации по каналам связи небезопасна, потому что сообщение может пройти десятки систем, прежде чем оно достигнет потенциального адресата, и нет никаких гарантий, что по пути никто не сможет взломать секретный ключ. УитфилдДиффи и Мартин Хеллман предложили зашифровывать секретный ключ DES по алгоритму Диффи‑Хеллмана на передающей стороне и пересылать его вместе с сообщением, зашифрованным с использованием DES. Тогда на другом конце его сможет расшифровать только получатель сообщения.

На практике обмен ключами по алгоритму Диффи‑Хеллмана происходит по следующей схеме.

1. Два участника обмена договариваются о двух числах. Один выбирает большое простое число, а другой – целое число, меньшее числа первого участника. Переговоры они могут вести открыто, и это никак не отразится на безопасности.
2. Каждый из двух участников, независимо друг от друга, генерирует другое число, которое они будут хранить в тайне. Эти числа выполняют роль секретного ключа. Далее в вычислениях используются секретный ключ и два предыдущих целых числа. Результат вычислений посылается участнику обмена, и он играет роль открытого ключа.
3. Участники обмена обмениваются открытыми ключами. Далее они, используя собственный секретный ключ и открытый ключ партнера, конфиденциально вычисляют ключ сессии. Каждый партер вычисляет один и тот же ключ сессии.
4. Ключ сессии может использоваться как секретный ключ для другого алгоритма шифрования, например DES. Никакое третье лицо, контролирующее обмен, не сможет вычислить ключ сессии, не зная один из секретных ключей.

**Пример**

Ева — криптоаналитик. Она читает пересылку Боба и Алисы, но не изменяет содержимого их сообщений.

* s = секретный ключ. s = 2
* g = простое число меньшее p. g = 5
* p = открытое простое число. p = 23
* a = секретный ключ Алисы. a = 6
* A = открытый ключ Алисы. A = gamod p = 8
* b = секретный ключ Боба. b = 15
* B = открытый ключ Боба. B = gbmod p = 19

Таблица 6.2 – Алгоритм Дифи-Хеллмана



## **Задание для выполнения**

1. **Написать программу, которая получит общий секретный ключ из доступных публичных ключей с помощью алгоритма Диффи-Хеллмана.**

Генерируйте секретный ключ Алисы и Боба, и высчитайте секретный ключ, удостоверьтесь что его одинаково смогут получить и Алиса и Боб.

Реализация алгоритма на языке C#:

Public static classPower

{

Public static uint power(uint a, uint b, uint n)

{// a^b mod n

Uint tmp = a;

uint sum = tmp;

for (inti = 1; i< b; i++)

{

for (int j = 1; j < a; j++)

{

sum += tmp;

if (sum >= n)

{

sum -= n;

}

}

tmp = sum;

}

returntmp;

}

}

Public class PrimeNumberGenerator

{

Private readonly Random \_random = new Random();

Public uint Generate()

{

uint number = (uint)\_random.Next(500);

while (!IsPrime(number))

{

unchecked

{

number++;

}

}

return number;

}

privatestaticboolIsPrime(uint number)

{

if ((number & 1) == 0) return (number == 2);

var limit = (int)Math.Sqrt(number);

for (inti = 3; i<= limit; i += 2)

{

if ((number % i) == 0) returnfalse;

}

returntrue;

}

}

classAlice

{

publicstaticuintprimeNumber;

publicstaticuintaliceNumber;

privatestaticuintsecretKey;

publicstaticuintopenKey;

privatestaticuintgeneralSecretKey;

publicstaticvoidGenerateNumber()

{

Random ranGenerator;

ranGenerator = new Random();

aliceNumber = (uint)ranGenerator.Next((int)primeNumber);

Bob.aliceNumber = aliceNumber;

}

publicstaticvoidGenerateSecretKey()

{

Random ranGenerator;

ranGenerator = new Random();

secretKey = (uint)ranGenerator.Next((int)primeNumber);

}

publicstaticvoidGenerateOpenKey()

{

openKey = Power.power(aliceNumber, secretKey, primeNumber);

}

publicstaticvoidGenerateGeneralSecretKey()

{

generalSecretKey = Power.power(Bob.openKey, secretKey, primeNumber);

}

publicstaticvoid Output()

{

Console.WriteLine($"Случайноечисломеньшее{primeNumber}: {aliceNumber}");

Console.WriteLine($"СекретныйключАлисы{secretKey}");

Console.WriteLine($"ОткрытыйключАлисы{openKey}");

Console.WriteLine($"Общийсекретныйключ, которыйполучилаАлиса{generalSecretKey}");

}

}

publicstaticclassBob

{

publicstaticuintprimeNumber;

publicstaticuintaliceNumber;

privatestaticuintsecretKey;

publicstaticuintopenKey;

privatestaticuintgeneralSecretKey;

publicstaticvoidGeneratePrimeNumber()

{

PrimeNumberGenerator generator;

generator = newPrimeNumberGenerator();

primeNumber = generator.Generate();

Alice.primeNumber = primeNumber;

}

publicstaticvoidGenerateSecretKey()

{

Random ranGenerator;

ranGenerator = new Random();

secretKey = (uint)ranGenerator.Next((int)primeNumber);

}

publicstaticvoidGenerateOpenKey()

{

openKey = Power.power(aliceNumber, secretKey, primeNumber);

}

publicstaticvoidGenerateGeneralSecretKey()

{

generalSecretKey = Power.power(Alice.openKey, secretKey, primeNumber);

}

publicstaticvoid Output()

{

Console.WriteLine($"Простоечисло: {primeNumber}");

Console.WriteLine($"СекретныйключБоба: {secretKey}");

Console.WriteLine($"ОткрытыйключБоба: {openKey}");

Console.WriteLine($"Общийсекретныйключ, которыйполучилБоб: {generalSecretKey}");

}

}

publicclassDiffieHellman

{

publicDiffieHellman()

{

Bob.GeneratePrimeNumber();

Alice.GenerateNumber();

Bob.GenerateSecretKey();

Thread.Sleep(1000);

Alice.GenerateSecretKey();

Bob.GenerateOpenKey();

Alice.GenerateOpenKey();

Bob.GenerateGeneralSecretKey();

Alice.GenerateGeneralSecretKey();

Bob.Output();

Alice.Output();

}

}

classProgram

{

staticvoid Main(string[] args)

{

DiffieHellmandiffieHellman = newDiffieHellman();

}

}

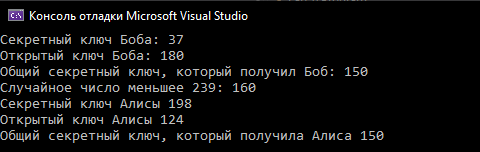
Результатработыпрограммыпредставленнарисунке 6.1:

Рисунок 6.1 – Результат работы программы

**2. Написать программу, которая зашифрует и расшифрует текст, введенный с клавиатуры, зашифрует и расшифрует его алгоритмом RSA.**

Таблица 6.3 – Ключи для алгоритма RSA

|  |  |  |  |
| --- | --- | --- | --- |
| # | e | n | d |
| 3 | 17 | 391 | 145 |

Реализация алгоритма на языке C#:

publicclassRSA

{

publiclonge;

publiclongn;

publiclong d;

publicstring input;

public List<string> encoded;

publicstring output;

publicRSA()

{

Console.WriteLine("Введитее ");

e = Convert.ToInt64(Console.ReadLine());

Console.WriteLine("Введите n ");

n = Convert.ToInt64(Console.ReadLine());

Console.WriteLine("Введите d ");

d = Convert.ToInt64(Console.ReadLine());

Console.WriteLine("Введите текст для шифрования ");

input = Console.ReadLine();

encoded = RSA\_Encode(input, e, n);

foreach(string item in encoded)

{

Console.WriteLine(item);

}

output = RSA\_Decode(encoded, d, n);

Console.WriteLine(output);

}

public List<string>RSA\_Encode(string input, long e, long n)

{

List<string> result = new List<string>();

BigInteger bi;

for(inti = 0; i<input.Length; i++)

{

long index = Convert.ToInt64(input[i]);

bi = newBigInteger(index);

bi = BigInteger.Pow(bi, (int)e);

BigInteger \_n = newBigInteger((int)n);

bi = bi % \_n;

result.Add(bi.ToString());

}

return result;

}

publicstringRSA\_Decode(List<string> input, long d, long n)

{

string result = "";

BigInteger bi;

foreach(string item in input)

{

bi = newBigInteger(Convert.ToDouble(item));

bi = BigInteger.Pow(bi, (int)d);

BigInteger \_n = newBigInteger((int)n);

bi = bi % \_n;

result += (char)bi;

}

return result;

}

}

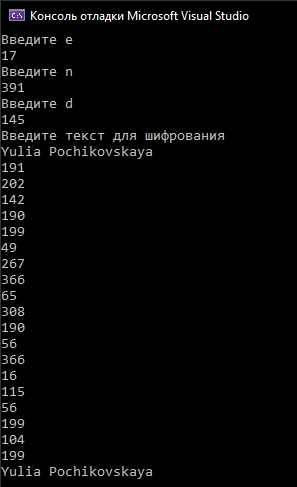
staticvoid Main(string[] args)

{

RSArsa = newRSA();

}

Результат работы программы представлен на рисунке 6.2:

Рисунок 6.2 – Результат работы программы

**3. Написать программу, которая сгенерирует ключи, зашифрует и расшифрует текст, введенный с клавиатуры алгоритмом Эль-Гамаля.**

Реализация алгоритма на языке C#:

publicclassElGamal

{

publicint p;

publicint x;

publicint g;

publicint y;

publicstring input;

public List<string> encoded;

publicstring decoded;

privatereadonly Random \_random = new Random();

publicintGetPRoot(int p)

{

for (inti = 0; i< p; i++)

if (IsPRoot(p, i))

returni;

return 0;

}

publicboolIsPRoot(int p, int a)

{

if (a == 0 || a == 1)

returnfalse;

int last = 1;

HashSet<int> set = newHashSet<int>();

for (inti = 0; i< p - 1; i++)

{

last = (last \* a) % p;

if (set.Contains(last)) // Еслиповтор

returnfalse;

set.Add(last);

}

returntrue;

}

publicint Power(int a, int b, int n)

{// a^b mod n

inttmp=a;

int sum=tmp;

for(inti=1;i<b;i++)

{

for(int j=1;j<a;j++)

{ sum+=tmp;

if(sum>=n)

{

sum-=n;

}

} tmp=sum;

}

returntmp;

}

publicintMul(int a, int b, int n)

{// a\*b mod n

int sum=0;

for(inti=0;i<b;i++)

{

sum+=a;

if(sum>=n)

{

sum-=n;

}

}

return sum;

}

publicElGamal()

{

bool ok = false;

while(ok == false){

p = \_random.Next(1000);

int l = GetPRoot(p);

g = l;

if (g > 0)

ok = true;

}

x = \_random.Next(1, p - 1);

y = Power(g, x, p);

Console.WriteLine("Введитешифруемыйтекст");

input = Console.ReadLine();

encoded = Encode(p, g, x, input);

foreach (stringiin encoded)

Console.WriteLine(i);

decoded = Decode(p, x, encoded);

Console.WriteLine(decoded);

}

public List<string> Encode(int p, int g, int x, string input)

{

List<string> result = new List<string>();

for (inti = 0; i<input.Length; i++)

{

int k = \_random.Next(1, p - 1);

int m = Convert.ToInt32(input[i]);

int a = Power(g, k, p);

int b = Mul(Power(y, k, p), m, p);

result.Add($"{a},{b}");

}

return result;

}

publicstring Decode(int p, int x, List<string> input)

{

string result = "";

foreach(string item in input)

{

string[] nums = item.Split(',');

int a = int.Parse(nums[0]);

int b = int.Parse(nums[1]);

intdM = Mul(b, Power(a, p - 1 - x, p), p);

result += (char)dM;

}

return result;

}

}

staticvoid Main(string[] args)

{

ElGamalelg = newElGamal();

}

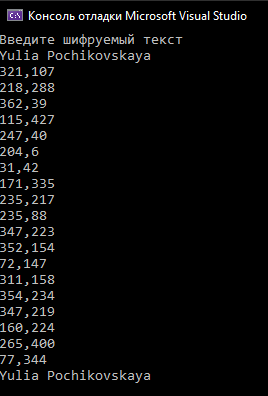
Результат работыпрограммыпредставленнарисунке 6.3:

Рисунок 6.3 – Результат работы программы

**Вывод:** Были изучены основные криптографические алгоритмы асимметричного шифрования.