Федеральное агентство связи

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Информационная безопасность»

Отчёт по лабораторной работе №4

по дисциплине

«Методы и средства защиты компьютерной информации»

Изучение алгоритмов электронной цифровой подписи

Выполнил:

студент группы БВТ1301

Редько Е.Ю.

Проверил:

ассистент кафедры ИБ

Барков В.В.

Москва 2016

**Цель работы**

Изучить принципы работы алгоритмов электронной цифровой подписи на примере реализации одного из алгоритмов.

**Задание**

Используя один из объектно-ориентированных языков программирования реализовать алгоритм электронной цифровой подписи, указанный в индивидуальном варианте.

В качестве алгоритма хеширования используйте алгоритм, разработанный в предыдущей работе.

Проект реализовать в виде библиотеки классов. При реализации использовать интерфейсы. Алгоритм должен быть универсальным с точки зрения данных для хеширования (строка, файл, сокет).

Для созданного проекта необходимо написать модульные тесты, показывающие корректность работы алгоритма.

**Описание алгоритма цифровой подписи RSA**

Система RSA может использоваться не только для шифрования, но и для цифровой подписи.

Поскольку цифровая подпись обеспечивает как аутентификацию автора сообщения, так и подтверждение целостности содержимого подписанного сообщения, она служит аналогом подписи, сделанной от руки в конце рукописного документа.

Важное свойство цифровой подписи заключается в том, что её может проверить каждый, кто имеет доступ к открытому ключу её автора. Один из участников обмена сообщениями после проверки подлинности цифровой подписи может передать подписанное сообщение ещё кому-то, кто тоже в состоянии проверить эту подпись. Например, сторона А может переслать стороне В электронный чек. После того как сторона В проверит подпись стороны А на чеке, она может передать его в свой банк, служащие которого также имеют возможность проверить подпись и осуществить соответствующую денежную операцию. Заметим, что подписанное сообщение не зашифровано. Оно пересылается в исходном виде и его содержимое не защищено от нарушения конфиденциальности. Путём совместного применения представленных выше схем шифрования и цифровой подписи в системе RSA можно создавать сообщения, которые будут и зашифрованы, и содержать цифровую подпись. Для этого автор сначала должен добавить к сообщению свою цифровую подпись, а затем — зашифровать получившуюся в результате пару (состоящую из самого сообщения и подписи к нему) с помощью открытого ключа, принадлежащего получателю. Получатель расшифровывает полученное сообщение с помощью своего секретного ключа. Если проводить аналогию с пересылкой обычных бумажных документов, то этот процесс похож на то, как если бы автор документа поставил под ним свою печать, а затем положил его в бумажный конверт и запечатал, с тем чтобы конверт был распечатан только тем человеком, кому адресовано сообщение.

|  |  |
| --- | --- |
| **Алгоритм подписи**:   * Взять открытый текст m * Создать цифровую подпись s с помощью своего секретного ключа {d,N}: s = Sa(m) = md mod N * Передать пару {m, s}, состоящую из сообщения и подписи. |  |

**Алгоритм проверки**:

* Принять пару {m, s}
* Взять открытый ключ {e,N} автора
* Вычислить прообраз сообщения из подписи: m’ = Pa(s) = se mod N
* Проверить подлинность подписи (и неизменность сообщения), сравнив m и m’

## Схема алгоритма цифровой подписи RSA

Схемы алгоритма генерации цифровой подписи RSA и ее проверки представлены на рисунках 1 и 2 соответственно.

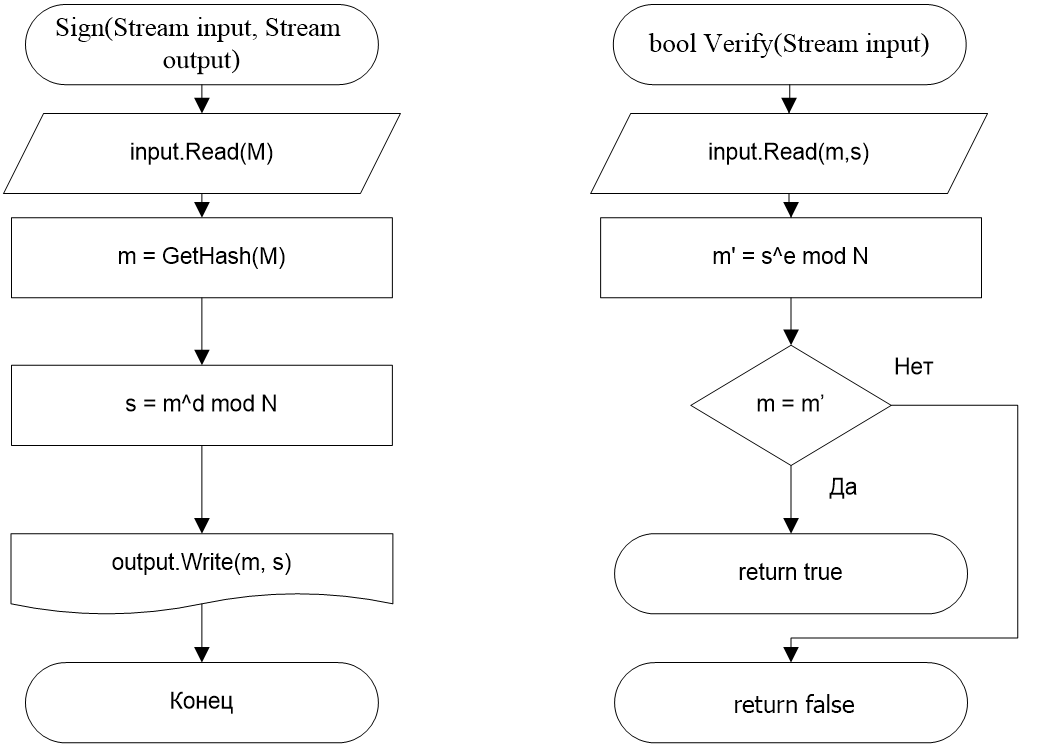


Рисунок 1. Алгоритм формирования подписи. Рисунок 2. Алгоритм проверки подписи.

**Код программы**

ISign.cs (интерфейс):

namespace RedkoLib

{

public interface ISign

{

void Sign(Stream input, Stream output);

void SetKey(Stream key);

void SetHashFunction(IMD5 hash);

bool Verify(Stream input);

}

}

SignRSA.cs:

namespace RedkoLib

{

public class SignatureRSA : ISign

{

IMD5 hashFunction;

static BigInteger N;

static BigInteger d, c;

public void RSA\_Params()

{

BigInteger P = GeneratePrime();

BigInteger Q = GeneratePrime();

P = 19; Q = 31;

N = P \* Q;

BigInteger f = (P - 1) \* (Q - 1);

Random rand = new Random();

while (true)

{

d = rand.Next(1, Convert.ToInt32(f.ToString())); d = 7;

if (!CheckMutualPrime(Convert.ToInt32(d.ToString()), Convert.ToInt32(f.ToString())))

continue;

break;

}

c = Inverse(d, f);

}

private BigInteger Power(BigInteger a, BigInteger b, BigInteger m) // a^b mod m

{

BigInteger tmp = a;

BigInteger sum = tmp;

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

{

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

{

sum += tmp;

if (sum >= m)

{

sum -= m;

}

}

tmp = sum;

}

return tmp;

}

public void EuclideanAlgorithm(BigInteger a, BigInteger b, out BigInteger x, out BigInteger y, out BigInteger NOD)

{

if (a < b)

{

BigInteger temp = a;

a = b;

b = temp;

}

BigInteger[] U = { a, 1, 0 };

BigInteger[] V = { b, 0, 1 };

BigInteger[] T = new BigInteger[3];

while (V[0] != 0)

{

BigInteger q = U[0] / V[0];

T[0] = U[0] % V[0];

T[1] = U[1] - q \* V[1];

T[2] = U[2] - q \* V[2];

V.CopyTo(U, 0);

T.CopyTo(V, 0);

}

NOD = U[0];

x = U[1];

y = U[2];

return;

}

public BigInteger Inverse(BigInteger c, BigInteger m)

{

BigInteger x, y, NOD;

EuclideanAlgorithm(m, c, out x, out y, out NOD);

if (y < 0)

y += m;

return y;

}

public int GeneratePrime()

{

Random rand = new Random();

int a = rand.Next(10000, 11000);

if (a % 2 == 0)

a++;

while (true)

{

if (CheckPrime(a))

return a;

a += 2;

}

}

public bool CheckPrime(int n)

{

bool isPrime = true;

for (int i = 2; i < n; i++)

{

if (n % i == 0)

{

isPrime = false;

break;

}

}

return isPrime;

}

public static bool CheckMutualPrime(int a, int b)

{

if (NOD(a, b) == 1)

return true;

return false;

}

public static int NOD(int a, int b)

{

if (a < b)

{

int temp = a;

a = b;

b = temp;

}

while (b != 0)

{

int t = a % b;

a = b;

b = t;

}

return a;

}

public void Sign(Stream input, Stream output)

{

byte[] buf = new byte[input.Length];

MemoryStream hash = new MemoryStream();

hashFunction.GetHash(input, hash);

buf = new byte[hash.Length];

hash.Position = 0;

hash.Read(buf, 0, buf.Length);

string hash\_msg = Encoding.UTF8.GetString(buf);

RSA\_Params();

List<string> result = SignRSA(hash\_msg, (int)c, (int)N);

output.Position = 0;

output.Write(Encoding.Default.GetBytes(hash\_msg), 0, Encoding.Default.GetBytes(hash\_msg).Length);

output.Write(Encoding.Default.GetBytes(" "), 0, Encoding.Default.GetBytes(" ").Length);

for (int i = 0; i < result.Count; ++i)

{

output.Write(Encoding.Default.GetBytes(result[i]), 0, Encoding.Default.GetBytes(result[i]).Length);

output.Write(Encoding.Default.GetBytes(" "), 0, Encoding.Default.GetBytes(" ").Length);

}

}

char[] characters = new char[] { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', 'A', 'B', 'C', 'D', 'E', 'F' };

public void SetKey(Stream key)

{

try

{

byte[] buf = new byte[key.Length];

key.Read(buf, 0, buf.Length);

string[] keys = Encoding.UTF8.GetString(buf).Split(' ');

N = new BigInteger(Encoding.Default.GetBytes(keys[0]));

d = new BigInteger(Encoding.Default.GetBytes(keys[1]));

}

catch (Exception ex) { }

}

public Stream GetPublicKey()

{

MemoryStream ms = new MemoryStream();

ms.Write(N.ToByteArray(), 0, N.ToByteArray().Length);

ms.Write(Encoding.Default.GetBytes(" "), 0, Encoding.Default.GetBytes(" ").Length);

ms.Write(d.ToByteArray(), 0, d.ToByteArray().Length);

return ms;

}

public void SetHashFunction(IMD5 hash)

{

hashFunction = hash;

}

private List<string> SignRSA(string s, long e, long n)

{

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

BigInteger bi;

for (int i = 0; i < s.Length; i++)

{

int index = Array.IndexOf(characters, s[i]);

bi = new BigInteger(index);

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

BigInteger n\_ = new BigInteger((int)n);

bi = bi % n\_;

result.Add(bi.ToString());

}

return result;

}

private string CheckRSA(List<string> input, long d, long n)

{

string result = "";

BigInteger bi;

foreach (string item in input)

{

bi = new BigInteger(Convert.ToInt32(item));

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

BigInteger n\_ = new BigInteger((int)n);

bi = bi % n\_;

int index = Convert.ToInt32(bi.ToString());

result += characters[index].ToString();

}

return result;

}

public bool Verify(Stream input)

{

byte[] buf = new byte[input.Length];

input.Position = 0;

input.Read(buf, 0, buf.Length);

List<string> inp = new List<string>(Encoding.UTF8.GetString(buf).Split(' '));

List<string> sign = new List<string>(inp.Count - 1);

for (int i = 0; i < inp.Count - 2; ++i)

sign.Add(inp[i + 1]);

string res = CheckRSA(sign, (int)d, (int)N);

if (res == inp[0])

return true;

return false;

}

}

struct Message

{

public string message;

public BigInteger[] signature;

}

}

TSign.cs (тест):

namespace RedkoTest

{

[TestClass]

public class TSign

{

[TestMethod]

public void SignAndVerify()

{

string sourceDataString = "test";

MemoryStream input = new MemoryStream(Encoding.Default.GetBytes(sourceDataString));

MemoryStream output = new MemoryStream();

RedkoLib.SignatureRSA rsa = new RedkoLib.SignatureRSA();

rsa.RSA\_Params();

rsa.SetHashFunction(new RedkoLib.MD5());

rsa.Sign(input, output);

Assert.IsTrue(rsa.Verify(output));

}

}

}

