Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №6

**Отечественный стандарт цифровой подписи ГОСТ 3410.**

Выполнил:

Студент гр. 853504

Пресный В.И,

Проверил:

Олисейчик В.В.

Минск 2021

**Теоретические сведения**

**Отечественный стандарт цифровой подписи ГОСТ 3410**

В стандарте *ГОСТ 3410* используется хэш-функция ГОСТ 3411, которая создает хэш-код длиной 256 бит. Это во многом обуславливает требования к выбираемым простым числам p и q:

р должно быть простым числом в диапазоне

2509 < p < 2512либо 21020 < p < 21024

q должно быть простым числом в диапазоне 2254 < q < 2256

q также должно быть делителем (р-1).

Аналогично выбирается и параметр g.

При этом требуется, чтобы gq (mod p) = 1.

В соответствии с теоремой Ферма это эквивалентно условию в *DSS*,

что g = h(p-1)/q mod p.

Закрытым ключом является произвольное число х: 0 < x < q

Открытым ключом является число y: y = gx mod p

Для создания подписи выбирается случайное число k: 0 < k < q

Подпись состоит из двух чисел (r, s), вычисляемых по следующим формулам:

r = (gk mod p) mod q

s = (k H(M) + xr) mod q

Обратим внимание на отличия *DSS* и *ГОСТ 3410*.

1)                Используются разные хэш-функции: в *ГОСТ 3410* применяется отечественный стандарт на хэш-функции ГОСТ 3411, в *DSS* используется SHA-1, которые имеют разную длину хэш-кода. Отсюда и разные требования на длину простого числа q: в *ГОСТ 3410* длина q должна быть от 254 бит до 256 бит, а в *DSS* длина q должна быть от 159 бит до 160 бит.

2)                По-разному вычисляется компонента s подписи.

В *ГОСТ 3410* компонента s вычисляется по формуле:

s = (k H(M) + xr) mod q

В *DSS* компонента s вычисляется по формуле:

s = [k-1 (H(M) + xr)] mod q

Последнее отличие приводит к соответствующим отличиям в формулах для проверки подписи.

Получатель вычисляет w = H(M)-1 mod q

u1 = w s mod q

u2 = (q-r) w mod q

v = [(gu1 yu2) mod p] mod q

Подпись корректна, если v = r.

Структура обоих алгоритмов довольно интересна. Заметим, что значение r совсем не зависит от сообщения. Вместо этого r есть функция от k и трех общих компонент открытого ключа. Мультипликативная инверсия k (mod p) (в случае *DSS*) или само значение k (в случае ГОСТ 3410) подается в функцию, которая, кроме того, в качестве входа имеет хэш-код сообщения и закрытый ключ пользователя. Эта функция такова, что получатель может вычислить r, используя входное сообщение, подпись, открытый ключ пользователя и общий открытый ключ.

В силу сложности вычисления дискретных логарифмов нарушитель не может восстановить k из r или х из s.

Другое важное замечание заключается в том, что экспоненциальные вычисления при создании подписи необходимы только для gk mod p. Так как это значение от подписываемого сообщения не зависит, оно может быть вычислено заранее. Пользователь может заранее просчитать некоторое количество значений r и использовать их по мере необходимости для подписи документов. Еще одна задача состоит в определении мультипликативной инверсии k-1 (в случае *DSS*). Эти значения также могут быть вычислены заранее.

**Хэш-функция ГОСТ 3411-94**

Алгоритм *ГОСТ 3411* является отечественным стандартом для хэш-функций. Длина хэш-кода, создаваемого алгоритмом *ГОСТ 3411*, равна 256 битам. Алгоритм разбивает сообщение на блоки, длина которых также равна 256 битам. Кроме того, параметром алгоритма является стартовый вектор хэширования *Н* - произвольное фиксированное значение длиной также 256 бит.

***Алгоритм обработки одного блока сообщения***

Сообщение обрабатывается блоками по 256 бит справа налево.

Каждый блок сообщения обрабатывается по следующему алгоритму.

1.           Генерация четырех ключей длиной 256 бит каждый.

2.           Шифрование 64-битных значений промежуточного хэш-кода H на ключах Ki(i = 1, 2, 3, 4) с использованием алгоритма ГОСТ 28147 в режиме простой замены.

3.           Перемешивание результата шифрования.

Для генерации ключей используются следующие данные:

                промежуточное значение хэш-кода Н длиной 256 бит;

                текущий обрабатываемый блок сообщения М длиной 256 бит;

                параметры - три значения С2, С3 и С4 длиной 256 бит следующего вида: С2 и С4 состоят из одних нулей, а С3 равно

18 08 116 024 116 08 (08 18)2 18 08 (08 18)4 (18 08)4

где степень обозначает количество повторений 0 или 1.

Используются две формулы, определяющие перестановку и сдвиг.

Перестановка Р битов определяется следующим образом: каждое 256-битное значение рассматривается как последовательность тридцати двух 8-битных значений.

Перестановка Р элементов 256-битной последовательности выполняется по формуле y = φ(x), где x - порядковый номер 8-битного значения в исходной последовательности; y - порядковый номер 8-битного значения в результирующей последовательности.

φ(i + 1 + 4(k - 1)) = 8i + k ; i = 0 ÷ 3, k = 1 ÷ 8

Сдвиг А определяется по формуле

A (x) = (x1  x2) || x4 || x3 || x2

Где

|  |
| --- |
| xi - соответствующие 64 бита 256-битного значения х, |
| || обозначает конкатенацию. |

Присваиваются следующие начальные значения:

i = 1, U = H, V = M.

W = U V, K1 = Р (W)

Ключи K2, K3, K4 вычисляются последовательно по следующему алгоритму:

U = A(U)  Сi, V = A(A(V)),  W = U  V,  Ki = Р(W)

Далее выполняется шифрование 64-битных элементов текущего значения хэш-кода Н с ключами K1, K2, K3 и K4. При этом хэш-код Н рассматривается как последовательность 64-битных значений:

H = h4 || h3 || h2 || h1

Выполняется шифрование алгоритмом ГОСТ 28147:

si = EKi [hi]    i = 1, 2, 3, 4

S = s1 || s2 || s3 || s4

Наконец на заключительном этапе обработки очередного блока выполняется перемешивание полученной последовательности. 256-битное значение рассматривается как последовательность шестнадцати 16-битных значений. Сдвиг обозначается Ψ и определяется следующим образом:

|  |
| --- |
| η16 || η15 || ... || η1 - исходное значение |
| η1η2η3η4η13η16 || η16 || ... || η2 - результирующее значение |

Результирующее значение хэш-кода определяется следующим образом:

Χ(M, H) = ψ61 (H   ψ (M  ψ12(S)))

где

|  |
| --- |
| H - предыдущее значение хэш-кода, |
| М - текущий обрабатываемый блок, |
| Ψi - i-ая степень преобразования Ψ. |

Входными параметрами алгоритма являются:

                исходное сообщение М произвольной длины;

                стартовый вектор хэширования Н, длина которого равна 256 битам;

                контрольная сумма Σ, начальное значение которой равно нулю и длина равна 256 битам;

                переменная L, начальное значение которой равно длине сообщения.

Сообщение М делится на блоки длиной 256 бит и обрабатывается справа налево. Очередной блок i обрабатывается следующим образом:

1.           H = Χ(Mi, H)

2.           Σ = Σ  ' Mi

3.           L рассматривается как неотрицательное целое число, к этому числу прибавляется 256 и вычисляется остаток от деления получившегося числа на 2256. Результат присваивается L.

Где ' обозначает следующую операцию: Σ и Mi рассматриваются как неотрицательные целые числа длиной 256 бит. Выполняется обычное сложение этих чисел и находится остаток от деления результата сложения на 2256. Этот остаток и является результатом операции.

Самый левый, т.е. самый последний блок М' обрабатывается так:

1.           Блок добавляется слева нулями так, чтобы его длина стала равна 256 битам.

2.           Вычисляется Σ = Σ  ' Mi.

3.           L рассматривается как неотрицательное целое число, к этому числу прибавляется длина исходного сообщения М и находится остаток от деления результата сложения на 2256.

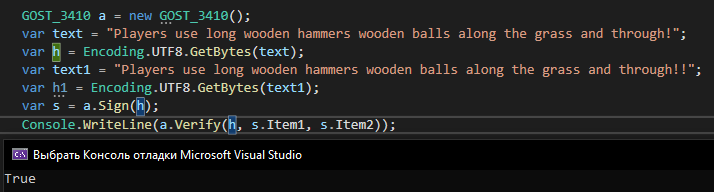
4.           Вычисляется Н = Χ(М', Н).

5.           Вычисляется Н = Χ(L, Н).

6.           Вычисляется Н = Χ(Σ, Н).

Значением функции хэширования является Н.

**Результат работы**

****

**Код приложения**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Text;

using System.Threading.Tasks;

namespace LR6

{

public class GOST\_3410

{

private readonly byte[] \_H = { 11, 25, 34, 41, 78, 24, 76, 78, 21, 38, 45, 46, 15, 51, 61, 71, 18, 25, 21, 11, 5, 6, 7, 8, 9, 10, 86, 24, 144, 125, 122, 59 };

public GOST\_3410()

{

GenerateKeys();

}

private BigInteger P { get; set; }

private BigInteger Q { get; set; }

private BigInteger X { get; set; }

public BigInteger Y { get; set; }

public BigInteger G { get; set; }

static (BigInteger, BigInteger, BigInteger) Gcdex(BigInteger a, BigInteger b)

{

if (a == 0)

return (b, 0, 1);

(BigInteger gcd, BigInteger x, BigInteger y) = Gcdex(b % a, a);

return (gcd, y - (b / a) \* x, x);

}

static BigInteger InvMod(BigInteger a, BigInteger m)

{

(BigInteger g, BigInteger x, BigInteger y) = Gcdex(a, m);

return g > 1 ? 0 : (x % m + m) % m;

}

public void GenerateKeys()

{

P = BigInteger.Parse("00000000" + "EE8172AE" + "8996608F" + "B69359B8" + "9EB82A69" +

"854510E2" + "977A4D63" + "BC97322C" + "E5DC3386" +

"EA0A12B3" + "43E9190F" + "23177539" + "84583978" +

"6BB0C345" + "D165976E" + "F2195EC9" + "B1C379E3",

System.Globalization.NumberStyles.HexNumber);

Q = BigInteger.Parse("00000000" + "98915E7E" + "C8265EDF" + "CDA31E88" + "F24809DD" +

"B064BDC7" + "285DD50D" + "7289F0AC" + "6F49DD2D",

System.Globalization.NumberStyles.HexNumber);

X = RandBigInt(Q);

G = BigInteger.Parse("00000000" + "9E960315" + "00C8774A" + "869582D4" + "AFDE2127" +

"AFAD2538" + "B4B6270A" + "6F7C8837" + "B50D50F2" +

"06755984" + "A49E5093" + "04D648BE" + "2AB5AAB1" +

"8EBE2CD4" + "6AC3D849" + "5B142AA6" + "CE23E21c",

System.Globalization.NumberStyles.HexNumber);

Y = BigInteger.ModPow(G, X, P);

}

public (byte[], byte[]) Sign(byte[] data)

{

var gost3411 = new GOST\_3411\_94();

var h = gost3411.GetHashe(data, \_H);

var value = new BigInteger(h);

var k = RandBigInt(Q);

var r = BigInteger.ModPow(G, k, P) % Q;

var s = (k \* value + X \* r) % Q;

return (r.ToByteArray(), s.ToByteArray());

}

public bool Verify(byte[] data, byte[] r, byte[] s)

{

var gost3411 = new GOST\_3411\_94();

var h = gost3411.GetHashe(data, \_H);

var value = new BigInteger(h);

var w = InvMod(value, Q);

var u1 = new BigInteger(s) \* w % Q;

var u2 = ((Q - new BigInteger(r)) \* w) % Q;

return ((BigInteger.ModPow(G, u1, P) \* BigInteger.ModPow(Y, u2, P) % P) % Q == new BigInteger(r));

}

private BigInteger RandBigInt(BigInteger stop)

{

byte[] bytes = stop.ToByteArray();

Random random = new Random();

BigInteger R;

do

{

random.NextBytes(bytes);

bytes[bytes.Length - 1] &= 0x7F;

R = new BigInteger(bytes);

} while (R >= stop);

return R;

}

}

}