1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. «Аутентификация в сетях GSM»
2. по дисциплине «Технологии передачи данных в сетях мобильной связи»
3. Выполнил
4. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. ассистент Мясников А.В.

<*подпись*>

1. Санкт-Петербург
2. 2021

**Цели**

При помощи программных средств смоделировать механизмы обеспечения безопасности передачи данных в сетях GSM. Программный стенд должен представлять собой клиент-серверное приложение, где в роли клиента выступает мобильная станция, а серверная часть имитирует подсистему аутентификации GSM.

**Задача**

Необходимо реализовать аутентификацию мобильной станции посредством алгоритмов А3/А8, а также шифрование передачи данных от мобильной станции к базовой посредством алгоритма А5.

**Ход работы**

Перед описанием алгоритма введем некоторые определения:

AuC- Центр Аутентификации (Authentication Centre),подсистема аутентификации.

IMSI - уникальный ID каждой SIM-карты, называемый IMSI (International Mobile Subscriber Identity).

(Key for identification). Это число будет служить секретным ключом для данной SIM-карты. Когда SIM-карты доставляются мобильному оператору, с ними передаются данные о Ki каждой новый SIM-карты. Эти данные (в виде пар (IMSI, Ki)) заносят в “центр аутентификации” мобильной сети (AuC).

HLR (Home Location Register) — база данных, которая содержит информацию об абоненте сети GSM-оператора.

VLR (Visitors Location Register) — временная база данных абонентов, которые находятся в зоне действия определённого центра мобильной коммутации.

**A3/A8 (COMP128)**

Общее описание алгоритма:

А3 – алгоритм аутентификации

А8 – алгоритм генерации сеансового ключа

Алгоритмы A3 и А8 реализованы как единое вычисление. В настоящее время наиболее распространённая реализация алгоритма A3/A8 — это **COMP128**.

Данная функция выполняет обе задачи – генерирует данные для аутентификации, а также генерирует общий ключ шифрования.

Общие шаги установления соединения:

1. Центр Аутентификации(ЦА) генерирует 128-битное случайное число RAND и посылает его на Мобильную Станцию(МС).
2. МС зашифровывает полученное число RAND, используя свой секретный ключ Ki и алгоритм аутентификации A3.
3. MC берет первые 32 бита из последовательности, полученной на предыдущем шаге(назовем их SRES(signed response)) и отправляет их обратно на ЦА.
4. ЦА проделывает ту же операцию и получает 32 битную последовательность XRES(expected response).
5. После чего ЦА сравнивает SRES и XRES. В случае, если оба значения равны, телефон считается аутентифицированным.
6. МС и ЦА вычисляют сеансовый ключ шифрования, используя секретный ключ Ki и алгоритм формирования ключа A8:

|  |
| --- |
|  |

Рисунок 1 – Аутентификация.

COMP128 - хеш-функция разрядности 128 бит. 128 бит и 128 бит RAND. В качестве SRES берутся 32 старших бита вычисленного значения, а 64 младших бита в качестве сеансового ключа .

X [0..32] — 32байтный вход алгоритма, где X [0..15] = , а X [16..31] = RAND.

[0..511], [0..255], [0..127], [0..63] и [0..31] — секретные таблицы подстановки байт.

Алгоритм состоит из 8 раундов, в каждом раунде 5 итераций. Каждая итерация заключается в поиске по соответствующей таблице ( для первой итерации,  — для второй, и т. д.) и подстановке байт. В конце каждого раунда (за исключением последнего) происходит перестановка полученных 128 бит результата, и после перестановки эти 128 бит используется в следующем раунде.

|  |
| --- |
|  |

Рисунок 2 - Схема реализации схема алгоритма COMP 128.

**А5 – алгоритм поточного шифрования**

A5 состоит из трех линейных регистров сдвига с обратной связью(РСЛОС) R1, R2, R3, длиной 19, 22 и 23 бита соответственно(всего 64 бита).  
Сдвиг в регистрах R1, R2, R3 происходит только при выполнении определенного условия. Каждый регистр содержит " бит управления тактированием".

В R1 это 8-й бит, а в R2 и R3 — 10-й. На каждом шаге сдвигаются только те регистры, у которых значение бита синхронизации равно большинству значений синхронизирующих битов всех трех регистров.

Перед работой алгоритма выполняется инициализация регистров. После инициализации производится вычисление 228 бит выходной последовательности. Само шифрование представляет собой обыкновенный XOR между данными и произведенным алгоритмом A5/1 ключевым потоком.

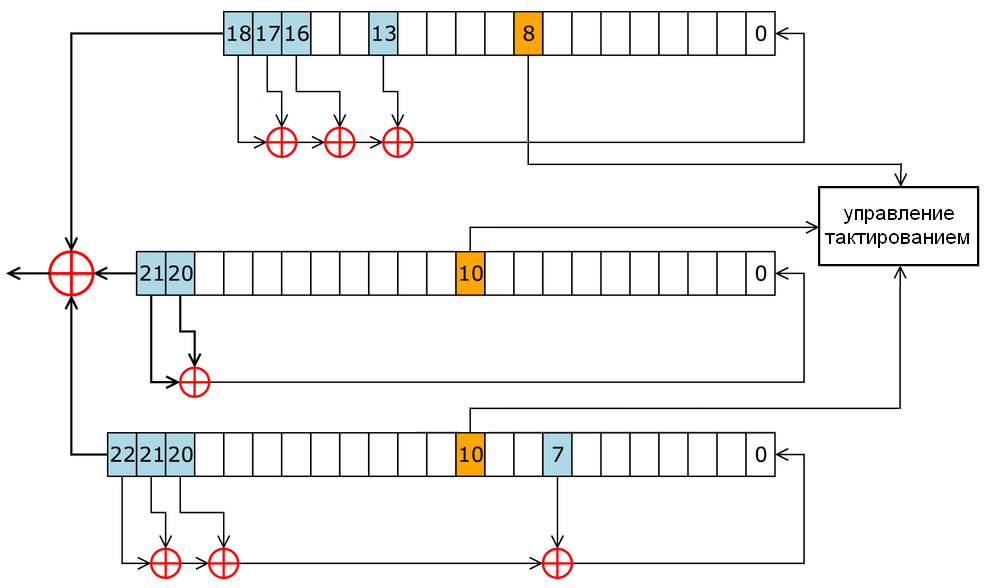


Рисунок 3 – РСЛОС алгоритма А5.

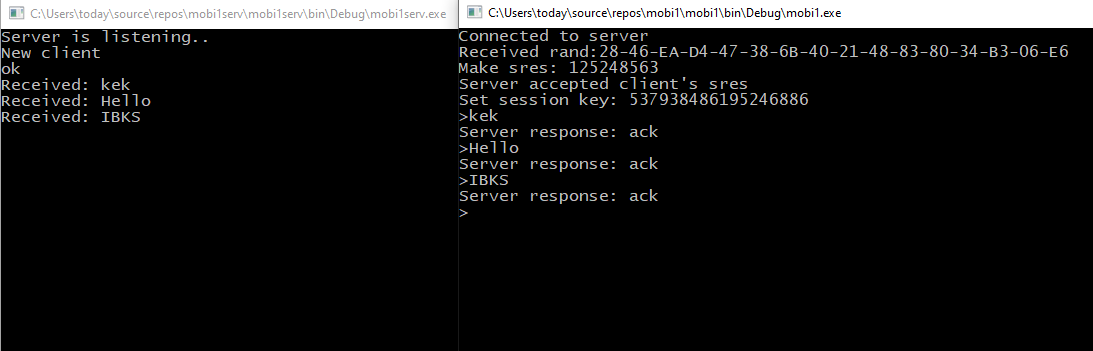


Рисунок 4 – Установленное соединение.

При соединении сервер присылает клиенту случайное число и ждет от него результат хэширования COMP128, в то же время производя те же расчеты.

При совпадении результатов – аутентификация успешно пройдена, при несовпадении соединение сбрасывается.

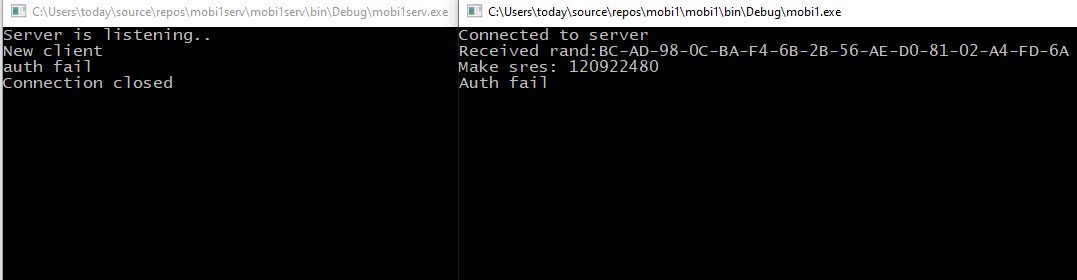


Рисунок 5 – Ошибка аутентификации.

Если аутентификация успешна, клиент и сервер устанавливают сеансовый ключ, который также вычисляется COMP128.

Теперь данные можно шифровать алгоритмом A5/1.

При общении клиента с сервером сообщения шифруются. Ответы сервера «ack» также шифруются алгоритмом A5/1.

**Контрольные вопросы**

1. Какие вариации алгоритма А5 существуют?

Существуют алгоритмы шифрования А5/0, А5/1, А5/2, А5/3.

1. В чем их основное различие?

* A5/0 - в качестве ключа использовались «нулевые» последовательности.
* А5/1 использовали в Европе и США, повышенная криптостойкость.
* А5/2 был специально разработан как экспортный вариант для стран, не входивших в Евросоюз. Для них алгоритм модифицировали, значительно понизив криптостойкость шифра. Криптостойкость А5/2 была понижена добавлением ещё одного регистра (17 бит), управляющего сдвигами остальных. (позиционировался как ослабленная версия A5/1)
* Так же разработан алгоритм А5/3, основанный на алгоритме Касуми и утверждённый для использования в сетях 3G.Эти модификации обозначают A5/x.

1. Что такое TMSI?  Для чего TMSI используется в алгоритмах аутентификации?

**Temporary Mobile Subscriber Identity** (TMSI) — временный идентификатор мобильной станции GSM, то есть это временный IMSI.

А IMSI (**International Mobile Subscriber Identity**) – это идентификационный номер абонента(мобильной станции). Когда мобильный телефон пытается зарегистрироваться в сети, его IMSI передается в AuC.

И для исключения определения (идентификации) абонента и его местоположения путем перехвата сообщений, передаваемых по радиоканалу, каждому абоненту системы связи присваивается «временное удостоверение личности» - временный международный идентификационный номер пользователя TMSI, который действителен только в пределах зоны расположения LAI(Location Area Identity (LAI) — международный идентификатор зоны местоположения).

Этот TMSI будет использоваться при всех последующих доступах к системе. Если мобильная станция переходит в новую область расположения, то её TMSI должен передаваться вместе с идентификационным номером зоны LAI, в которой TMSI был присвоен абоненту.

**Вывод**

В ходе работы изучен алгоритм установления связи между ЦА и мобильной стацией, реализованы алгоритмы шифрования, аутентификации и генерации сеансового ключа.

**Листинг**

Сервер

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

using System.Numerics;

using System.Net;

using System.Net.Sockets;

namespace mobi1serv

{

class Program

{

static public byte[] A5Encyptor(byte[] msg, byte[] key)

{

A5Enc a5 = new A5Enc();

int[] frame = new int[1]; frame[0] = 0x222;

//bool[] resbits = new bool[msg.Length];

int framesCount = msg.Length \*8 / 228;

if (Encoding.ASCII.GetString(msg) == "ack\0")

framesCount = 1;

byte[] aligned = new byte[29\*framesCount];

msg.CopyTo(aligned, 0);

BitArray msgbits = new BitArray(aligned);

bool[] resbits = new bool[msgbits.Length];

for (int i = 0; i < framesCount; i++)

{

frame[0] = i;

a5.KeySetup(key, frame);

bool[] KeyStream = a5.A5(true);

for (int j = 0; j < 228; j++)

{

resbits[i \* 228 + j] = msgbits[i \* 228 + j] ^ KeyStream[j];

}

}

return a5.FromBoolToByte(resbits, false);

}

class A5Enc

{

private bool[] reg = new bool[19];//рслос

private bool[] reg2 = new bool[22];

private bool[] reg3 = new bool[23];

//конструктор, который позволяет сразу установить начальное состояние регистров и нужное значение

public A5Enc(bool[][] startState)

{

reg = startState[0];

reg2 = startState[1];

reg3 = startState[2];

}

public A5Enc()

{

for (int i = 0; i < 19; i++)

reg[i] = false;

for (int i = 0; i < 22; i++)

reg2[i] = false;

for (int i = 0; i < 23; i++)

reg3[i] = false;

}

//нормальная инициализация регистров, используется при обычном вызове метода A5

public void KeySetup(byte[] key, int[] frame)

{

for (int i = 0; i < 19; i++)

reg[i] = false;

for (int i = 0; i < 22; i++)

reg2[i] = false;

for (int i = 0; i < 23; i++)

reg3[i] = false;

BitArray KeyBits = new BitArray(key);

BitArray FrameBits = new BitArray(frame);

bool[] b = new bool[64];

for (int i = 0; i < 64; i++)

{

clockall();

reg[0] = reg[0] ^ KeyBits[i];

reg2[0] = reg2[0] ^ KeyBits[i];

reg3[0] = reg3[0] ^ KeyBits[i];

}

for (int i = 0; i < 22; i++)

{

clockall();

reg[0] = reg[0] ^ FrameBits[i];

reg2[0] = reg2[0] ^ FrameBits[i];

reg3[0] = reg3[0] ^ FrameBits[i];

}

for (int i = 0; i < 100; i++)

{

clock();

}

}

//частичная инициализация, в регистры грузится только номер фрейма

public void KeySetup(int[] frame)

{

BitArray FrameBits = new BitArray(frame);

for (int i = 0; i < 22; i++)

{

clockall();

reg[0] = reg[0] ^ FrameBits[i];

reg2[0] = reg2[0] ^ FrameBits[i];

reg3[0] = reg3[0] ^ FrameBits[i];

}

for (int i = 0; i < 100; i++)

{

clock();

}

}

private void clock()

{

bool majority = ((reg[8] & reg2[10]) | (reg[8] & reg3[10]) | (reg2[10] & reg3[10])); //вычисляется большинство,чтобы знать, кого сдвинуть

if (reg[8] == majority)

clockone(reg);

if (reg2[10] == majority)

clocktwo(reg2);

if (reg3[10] == majority)

clockthree(reg3);

}

//набор функций реализующих сдвиги регистров

private bool[] clockone(bool[] RegOne)

{

bool temp = false;

for (int i = RegOne.Length - 1; i > 0; i--)

{

if (i == RegOne.Length - 1)

temp = RegOne[13] ^ RegOne[16] ^ RegOne[17] ^ RegOne[18];

RegOne[i] = RegOne[i - 1];

if (i == 1)

RegOne[0] = temp;

}

return RegOne;

}

private bool[] clocktwo(bool[] RegTwo)

{

bool temp = false;

for (int i = RegTwo.Length - 1; i > 0; i--)

{

if (i == RegTwo.Length - 1)

temp = RegTwo[20] ^ RegTwo[21];

RegTwo[i] = RegTwo[i - 1];

if (i == 1)

RegTwo[0] = temp;

}

return RegTwo;

}

private bool[] clockthree(bool[] RegThree)

{

bool temp = false;

for (int i = RegThree.Length - 1; i > 0; i--)

{

if (i == RegThree.Length - 1)

temp = RegThree[7] ^ RegThree[20] ^ RegThree[21] ^ RegThree[22];

RegThree[i] = RegThree[i - 1];

if (i == 1)

RegThree[0] = temp;

}

return RegThree;

}

private void clockall()

{

reg = clockone(reg);

reg2 = clocktwo(reg2);

reg3 = clockthree(reg3);

}

//метод возвращающий 114 бит сгенерированного потока

public bool[] A5()

{

bool[] FirstPart = new bool[114];

for (int i = 0; i < 114; i++)

{

clock();

FirstPart[i] = (reg[18] ^ reg2[21] ^ reg3[22]);

}

return FirstPart;

}

//метод возвращающий всю 228 битную последовательность сгенерированного потока

public bool[] A5(bool AsFrame)

{

bool[] FirstPart = new bool[228];

for (int i = 0; i < 228; i++)

{

clock();

FirstPart[i] = (reg[18] ^ reg2[21] ^ reg3[22]);

}

return FirstPart;

}

public byte[] FromBoolToByte(bool[] key, bool lsb)

{

int bytes = key.Length / 8;

if ((key.Length % 8) != 0) bytes++;

byte[] arr2 = new byte[bytes];

int bitIndex = 0, byteIndex = 0;

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

{

if (key[i])

{

if (lsb)

arr2[byteIndex] |= (byte)(((byte)1) << (7 - bitIndex));

else

arr2[byteIndex] |= (byte)(((byte)1) << (bitIndex));

}

bitIndex++;

if (bitIndex == 8)

{

bitIndex = 0;

byteIndex++;

}

}

return arr2;

}

}

static byte[] T0 = new byte[]{

102,177,186,162, 2,156,112, 75, 55, 25, 8, 12,251,193,246,188,

109,213,151, 53, 42, 79,191,115,233,242,164,223,209,148,108,161,

252, 37,244, 47, 64,211, 6,237,185,160,139,113, 76,138, 59, 70,

67, 26, 13,157, 63,179,221, 30,214, 36,166, 69,152,124,207,116,

247,194, 41, 84, 71, 1, 49, 14, 95, 35,169, 21, 96, 78,215,225,

182,243, 28, 92,201,118, 4, 74,248,128, 17, 11,146,132,245, 48,

149, 90,120, 39, 87,230,106,232,175, 19,126,190,202,141,137,176,

250, 27,101, 40,219,227, 58, 20, 51,178, 98,216,140, 22, 32,121,

61,103,203, 72, 29,110, 85,212,180,204,150,183, 15, 66,172,196,

56,197,158, 0,100, 45,153, 7,144,222,163,167, 60,135,210,231,

174,165, 38,249,224, 34,220,229,217,208,241, 68,206,189,125,255,

239, 54,168, 89,123,122, 73,145,117,234,143, 99,129,200,192, 82,

104,170,136,235, 93, 81,205,173,236, 94,105, 52, 46,228,198, 5,

57,254, 97,155,142,133,199,171,187, 50, 65,181,127,107,147,226,

184,218,131, 33, 77, 86, 31, 44, 88, 62,238, 18, 24, 43,154, 23,

80,159,134,111, 9,114, 3, 91, 16,130, 83, 10,195,240,253,119,

177,102,162,186,156, 2, 75,112, 25, 55, 12, 8,193,251,188,246,

213,109, 53,151, 79, 42,115,191,242,233,223,164,148,209,161,108,

37,252, 47,244,211, 64,237, 6,160,185,113,139,138, 76, 70, 59,

26, 67,157, 13,179, 63, 30,221, 36,214, 69,166,124,152,116,207,

194,247, 84, 41, 1, 71, 14, 49, 35, 95, 21,169, 78, 96,225,215,

243,182, 92, 28,118,201, 74, 4,128,248, 11, 17,132,146, 48,245,

90,149, 39,120,230, 87,232,106, 19,175,190,126,141,202,176,137,

27,250, 40,101,227,219, 20, 58,178, 51,216, 98, 22,140,121, 32,

103, 61, 72,203,110, 29,212, 85,204,180,183,150, 66, 15,196,172,

197, 56, 0,158, 45,100, 7,153,222,144,167,163,135, 60,231,210,

165,174,249, 38, 34,224,229,220,208,217, 68,241,189,206,255,125,

54,239, 89,168,122,123,145, 73,234,117, 99,143,200,129, 82,192,

170,104,235,136, 81, 93,173,205, 94,236, 52,105,228, 46, 5,198,

254, 57,155, 97,133,142,171,199, 50,187,181, 65,107,127,226,147,

218,184, 33,131, 86, 77, 44, 31, 62, 88, 18,238, 43, 24, 23,154,

159, 80,111,134,114, 9, 91, 3,130, 16, 10, 83,240,195,119,253

};

static byte[] T1 = new byte[]{

19, 11, 80,114, 43, 1, 69, 94, 39, 18,127,117, 97, 3, 85, 43,

27,124, 70, 83, 47, 71, 63, 10, 47, 89, 79, 4, 14, 59, 11, 5,

35,107,103, 68, 21, 86, 36, 91, 85,126, 32, 50,109, 94,120, 6,

53, 79, 28, 45, 99, 95, 41, 34, 88, 68, 93, 55,110,125,105, 20,

90, 80, 76, 96, 23, 60, 89, 64,121, 56, 14, 74,101, 8, 19, 78,

76, 66,104, 46,111, 50, 32, 3, 39, 0, 58, 25, 92, 22, 18, 51,

57, 65,119,116, 22,109, 7, 86, 59, 93, 62,110, 78, 99, 77, 67,

12,113, 87, 98,102, 5, 88, 33, 38, 56, 23, 8, 75, 45, 13, 75,

95, 63, 28, 49,123,120, 20,112, 44, 30, 15, 98,106, 2,103, 29,

82,107, 42,124, 24, 30, 41, 16,108,100,117, 40, 73, 40, 7,114,

82,115, 36,112, 12,102,100, 84, 92, 48, 72, 97, 9, 54, 55, 74,

113,123, 17, 26, 53, 58, 4, 9, 69,122, 21,118, 42, 60, 27, 73,

118,125, 34, 15, 65,115, 84, 64, 62, 81, 70, 1, 24,111,121, 83,

104, 81, 49,127, 48,105, 31, 10, 6, 91, 87, 37, 16, 54,116,126,

31, 38, 13, 0, 72,106, 77, 61, 26, 67, 46, 29, 96, 37, 61, 52,

101, 17, 44,108, 71, 52, 66, 57, 33, 51, 25, 90, 2,119,122, 35

};

static byte[] T2 = new byte[]{

52, 50, 44, 6, 21, 49, 41, 59, 39, 51, 25, 32, 51, 47, 52, 43,

37, 4, 40, 34, 61, 12, 28, 4, 58, 23, 8, 15, 12, 22, 9, 18,

55, 10, 33, 35, 50, 1, 43, 3, 57, 13, 62, 14, 7, 42, 44, 59,

62, 57, 27, 6, 8, 31, 26, 54, 41, 22, 45, 20, 39, 3, 16, 56,

48, 2, 21, 28, 36, 42, 60, 33, 34, 18, 0, 11, 24, 10, 17, 61,

29, 14, 45, 26, 55, 46, 11, 17, 54, 46, 9, 24, 30, 60, 32, 0,

20, 38, 2, 30, 58, 35, 1, 16, 56, 40, 23, 48, 13, 19, 19, 27,

31, 53, 47, 38, 63, 15, 49, 5, 37, 53, 25, 36, 63, 29, 5, 7

};

static byte[] T3 = new byte[]{

1, 5, 29, 6, 25, 1, 18, 23, 17, 19, 0, 9, 24, 25, 6, 31,

28, 20, 24, 30, 4, 27, 3, 13, 15, 16, 14, 18, 4, 3, 8, 9,

20, 0, 12, 26, 21, 8, 28, 2, 29, 2, 15, 7, 11, 22, 14, 10,

17, 21, 12, 30, 26, 27, 16, 31, 11, 7, 13, 23, 10, 5, 22, 19

};

static byte[] T4 = new byte[]{

15, 12, 10, 4, 1, 14, 11, 7, 5, 0, 14, 7, 1, 2, 13, 8,

10, 3, 4, 9, 6, 0, 3, 2, 5, 6, 8, 9, 11, 13, 15, 12

};

static byte Tj(int j, int index)

{

switch (j)

{

case 0: return T0[index];

case 1: return T1[index];

case 2: return T2[index];

case 3: return T3[index];

case 4: return T4[index];

}

return 0xff;

}

static void Comp128(byte[] RAND, byte[] Ki, ref int sres, ref Int64 Kc)

{

//http://prog.bobrodobro.ru/33840

byte[] x = new byte[32]; //256

BitArray bit = new BitArray(128); //128

int m, n, y, z;

Array.Copy(RAND, 0, x, 16, 16);

for (int i = 1; i < 8; ++i)

{

Array.Copy(Ki, 0, x, 0, 16);

for (int j = 0; j < 4; ++j)

{

for (int k = 0; k < (int)Math.Pow(2, j) - 1; k++)

{

for (int l = 0; l < (int)Math.Pow(2, 4 - j) - 1; l++)

{

m = l + k \* (int)Math.Pow(2, 5 - j);

n = m + (int)Math.Pow(2, 4 - j);

y = (x[m] + 2 \* x[n]) % (int)Math.Pow(2, 9 - j);

z = (2 \* x[m] + x[n]) % (int)Math.Pow(2, 9 - j);

x[m] = Tj(j, y);

x[n] = Tj(j, z);

}

}

}

for (int j = 0; j < 31; ++j)

{

for (int k = 0; k < 3; ++k)

{

//BitArray b = new BitArray(new byte[] { Convert.ToByte(j) });

var b = x[j];

var vv = new BitArray(new byte[] { b });

bit[4 \* j + k] = vv[3 - k];

}

}

if (i < 8)

{

for (int j = 0; j < 15; ++j)

{

for (int k = 0; k < 7; ++k)

{

int index = ((8 \* j + k) \* 17) % 128;

x[j + 16] |= (byte)(Convert.ToByte(bit[index]) << (7 - k));

}

}

}

}

int[] SRES = new int[4]; // need only one

bit.CopyTo(SRES, 0);

Int64 \_Kc = 0;

\_Kc = SRES[3];

\_Kc <<= 32;

\_Kc ^= SRES[2] ^ 0xfffff000;

Kc = \_Kc;

sres = SRES[3];

}

static void Main(string[] args)

{

// generate 128 bit ki

byte[] ki = Encoding.ASCII.GetBytes("MySecretPassword");

// generate 128bit RAND

byte[] rand = new byte[16];

Random r = new Random();

// r.NextBytes(rand);

// generate Kc, sres

int xres = 0;

Int64 kc = 0;

//Comp128(rand, ki, ref sres, ref kc);

IPEndPoint ipPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 3333);

// создаем сокет

Socket listenSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

try

{

// связываем сокет с локальной точкой, по которой будем принимать данные

listenSocket.Bind(ipPoint);

// начинаем прослушивание

listenSocket.Listen(10);

Console.WriteLine("Server is listening..");

while (true)

{

Socket handler = listenSocket.Accept();

Console.WriteLine("New client");

// receive tmsi

// generate rand

r.NextBytes(rand);

// send rand

handler.Send(rand);

// get xres

Comp128(rand, ki, ref xres, ref kc);

// получаем сообщение

int bytes = 0; // количество полученных байтов

byte[] data = new byte[256]; // буфер для получаемых данных

bytes = handler.Receive(data);

int irec = BitConverter.ToInt32(data,0);

if (xres == irec)

{

handler.Send(Encoding.ASCII.GetBytes("ok"));

Console.WriteLine("ok");

string res = "close";

do

{

bytes = handler.Receive(data);

byte[] tmp = new byte[bytes];

Array.Copy(data, 0, tmp, 0, bytes);

var dmsg = A5Encyptor(tmp, BitConverter.GetBytes(kc));

res = Encoding.ASCII.GetString(dmsg);

res = res.Substring(0, res.IndexOf("\0"));

Console.WriteLine("Received: " + res);

var emsg = A5Encyptor(Encoding.ASCII.GetBytes("ack\0"), BitConverter.GetBytes(kc));

handler.Send(emsg);

} while (res != "close");

}

else

{

Console.WriteLine("auth fail");

handler.Shutdown(SocketShutdown.Both);

handler.Close();

}

Console.WriteLine("Connection closed");

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

КЛИЕНТ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

using System.Numerics;

using System.Net;

using System.Net.Sockets;

namespace mobi1

{

class Program

{

static public byte[] A5Encyptor(byte[] msg, byte[] key)

{

A5Enc a5 = new A5Enc();

int[] frame = new int[1]; frame[0] = 0x222;

int framesCount = msg.Length\*8 / 228 +1;

byte[] aligned = new byte[29\*framesCount];

msg.CopyTo(aligned, 0);

BitArray msgbits = new BitArray(aligned);

bool[] resbits = new bool[msgbits.Length];

//if ((msgbits.Length % 228) != 0)

// framesCount++;

for (int i = 0; i < framesCount; i++)

{

frame[0] = i;

a5.KeySetup(key, frame);

bool[] KeyStream = a5.A5(true);

for (int j = 0; j < 228; j++)

{

resbits[i \* 228 + j] = msgbits[i \* 228 + j] ^ KeyStream[j];

}

}

return a5.FromBoolToByte(resbits, false);

}

class A5Enc

{

private bool[] reg = new bool[19];

private bool[] reg2 = new bool[22];

private bool[] reg3 = new bool[23];

//конструктор, который позволяет сразу установить начальное состояние регистров и нужное значение

public A5Enc(bool[][] startState)

{

reg = startState[0];

reg2 = startState[1];

reg3 = startState[2];

}

public A5Enc()

{

for (int i = 0; i < 19; i++)

reg[i] = false;

for (int i = 0; i < 22; i++)

reg2[i] = false;

for (int i = 0; i < 23; i++)

reg3[i] = false;

}

//нормальная инициализация регистров, используется при обычном вызове метода A5

public void KeySetup(byte[] key, int[] frame)

{

for (int i = 0; i < 19; i++)

reg[i] = false;

for (int i = 0; i < 22; i++)

reg2[i] = false;

for (int i = 0; i < 23; i++)

reg3[i] = false;

BitArray KeyBits = new BitArray(key);

BitArray FrameBits = new BitArray(frame);

bool[] b = new bool[64];

for (int i = 0; i < 64; i++)

{

clockall();// сдвигаем регистры на 1 позицию игноря биты синхронизации

reg[0] = reg[0] ^ KeyBits[i];

reg2[0] = reg2[0] ^ KeyBits[i];

reg3[0] = reg3[0] ^ KeyBits[i];

}

for (int i = 0; i < 22; i++)

{

clockall();// сдвигаем регистры на 1 позицию игноря биты синхронизации

reg[0] = reg[0] ^ FrameBits[i];

reg2[0] = reg2[0] ^ FrameBits[i];

reg3[0] = reg3[0] ^ FrameBits[i];

}

for (int i = 0; i < 100; i++)

{

clock();//сдвиг регистров на 1 позиц с учетом битов синхронизации

}

}

//частичная инициализация, в регистры грузится только номер фрейма

public void KeySetup(int[] frame)

{

BitArray FrameBits = new BitArray(frame);

for (int i = 0; i < 22; i++)

{

clockall();

reg[0] = reg[0] ^ FrameBits[i];

reg2[0] = reg2[0] ^ FrameBits[i];

reg3[0] = reg3[0] ^ FrameBits[i];

}

for (int i = 0; i < 100; i++)

{

clock();

}

}

private void clock()

{

bool majority = ((reg[8] & reg2[10]) | (reg[8] & reg3[10]) | (reg2[10] & reg3[10]));

if (reg[8] == majority)

clockone(reg);

if (reg2[10] == majority)

clocktwo(reg2);

if (reg3[10] == majority)

clockthree(reg3);

}

//набор функций реализующих сдвиги регистров

private bool[] clockone(bool[] RegOne)

{

bool temp = false;

for (int i = RegOne.Length - 1; i > 0; i--)

{

if (i == RegOne.Length - 1)

temp = RegOne[13] ^ RegOne[16] ^ RegOne[17] ^ RegOne[18];

RegOne[i] = RegOne[i - 1];

if (i == 1)

RegOne[0] = temp;

}

return RegOne;

}

private bool[] clocktwo(bool[] RegTwo)

{

bool temp = false;

for (int i = RegTwo.Length - 1; i > 0; i--)

{

if (i == RegTwo.Length - 1)

temp = RegTwo[20] ^ RegTwo[21];

RegTwo[i] = RegTwo[i - 1];

if (i == 1)

RegTwo[0] = temp;

}

return RegTwo;

}

private bool[] clockthree(bool[] RegThree)

{

bool temp = false;

for (int i = RegThree.Length - 1; i > 0; i--)

{

if (i == RegThree.Length - 1)

temp = RegThree[7] ^ RegThree[20] ^ RegThree[21] ^ RegThree[22];

RegThree[i] = RegThree[i - 1];

if (i == 1)

RegThree[0] = temp;

}

return RegThree;

}

private void clockall()

{

reg = clockone(reg);

reg2 = clocktwo(reg2);

reg3 = clockthree(reg3);

}

//метод возвращающий 114 бит сгенерированного потока

public bool[] A5()

{

bool[] FirstPart = new bool[114];

for (int i = 0; i < 114; i++)

{

clock();

FirstPart[i] = (reg[18] ^ reg2[21] ^ reg3[22]);

}

return FirstPart;

}

//метод возвращающий всю 228 битную последовательность сгенерированного потока

public bool[] A5(bool AsFrame)

{

bool[] FirstPart = new bool[228];

for (int i = 0; i < 228; i++)

{

clock();

FirstPart[i] = (reg[18] ^ reg2[21] ^ reg3[22]);

}

return FirstPart;

}

public byte[] FromBoolToByte(bool[] key, bool lsb)

{

int bytes = key.Length / 8;

if ((key.Length % 8) != 0) bytes++;

byte[] arr2 = new byte[bytes];

int bitIndex = 0, byteIndex = 0;

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

{

if (key[i])

{

if (lsb)

arr2[byteIndex] |= (byte)(((byte)1) << (7 - bitIndex));

else

arr2[byteIndex] |= (byte)(((byte)1) << (bitIndex));

}

bitIndex++;

if (bitIndex == 8)

{

bitIndex = 0;

byteIndex++;

}

}

return arr2;

}

}

static byte[] T0 = new byte[]{

102,177,186,162, 2,156,112, 75, 55, 25, 8, 12,251,193,246,188,

109,213,151, 53, 42, 79,191,115,233,242,164,223,209,148,108,161,

252, 37,244, 47, 64,211, 6,237,185,160,139,113, 76,138, 59, 70,

67, 26, 13,157, 63,179,221, 30,214, 36,166, 69,152,124,207,116,

247,194, 41, 84, 71, 1, 49, 14, 95, 35,169, 21, 96, 78,215,225,

182,243, 28, 92,201,118, 4, 74,248,128, 17, 11,146,132,245, 48,

149, 90,120, 39, 87,230,106,232,175, 19,126,190,202,141,137,176,

250, 27,101, 40,219,227, 58, 20, 51,178, 98,216,140, 22, 32,121,

61,103,203, 72, 29,110, 85,212,180,204,150,183, 15, 66,172,196,

56,197,158, 0,100, 45,153, 7,144,222,163,167, 60,135,210,231,

174,165, 38,249,224, 34,220,229,217,208,241, 68,206,189,125,255,

239, 54,168, 89,123,122, 73,145,117,234,143, 99,129,200,192, 82,

104,170,136,235, 93, 81,205,173,236, 94,105, 52, 46,228,198, 5,

57,254, 97,155,142,133,199,171,187, 50, 65,181,127,107,147,226,

184,218,131, 33, 77, 86, 31, 44, 88, 62,238, 18, 24, 43,154, 23,

80,159,134,111, 9,114, 3, 91, 16,130, 83, 10,195,240,253,119,

177,102,162,186,156, 2, 75,112, 25, 55, 12, 8,193,251,188,246,

213,109, 53,151, 79, 42,115,191,242,233,223,164,148,209,161,108,

37,252, 47,244,211, 64,237, 6,160,185,113,139,138, 76, 70, 59,

26, 67,157, 13,179, 63, 30,221, 36,214, 69,166,124,152,116,207,

194,247, 84, 41, 1, 71, 14, 49, 35, 95, 21,169, 78, 96,225,215,

243,182, 92, 28,118,201, 74, 4,128,248, 11, 17,132,146, 48,245,

90,149, 39,120,230, 87,232,106, 19,175,190,126,141,202,176,137,

27,250, 40,101,227,219, 20, 58,178, 51,216, 98, 22,140,121, 32,

103, 61, 72,203,110, 29,212, 85,204,180,183,150, 66, 15,196,172,

197, 56, 0,158, 45,100, 7,153,222,144,167,163,135, 60,231,210,

165,174,249, 38, 34,224,229,220,208,217, 68,241,189,206,255,125,

54,239, 89,168,122,123,145, 73,234,117, 99,143,200,129, 82,192,

170,104,235,136, 81, 93,173,205, 94,236, 52,105,228, 46, 5,198,

254, 57,155, 97,133,142,171,199, 50,187,181, 65,107,127,226,147,

218,184, 33,131, 86, 77, 44, 31, 62, 88, 18,238, 43, 24, 23,154,

159, 80,111,134,114, 9, 91, 3,130, 16, 10, 83,240,195,119,253

};

static byte[] T1 = new byte[]{

19, 11, 80,114, 43, 1, 69, 94, 39, 18,127,117, 97, 3, 85, 43,

27,124, 70, 83, 47, 71, 63, 10, 47, 89, 79, 4, 14, 59, 11, 5,

35,107,103, 68, 21, 86, 36, 91, 85,126, 32, 50,109, 94,120, 6,

53, 79, 28, 45, 99, 95, 41, 34, 88, 68, 93, 55,110,125,105, 20,

90, 80, 76, 96, 23, 60, 89, 64,121, 56, 14, 74,101, 8, 19, 78,

76, 66,104, 46,111, 50, 32, 3, 39, 0, 58, 25, 92, 22, 18, 51,

57, 65,119,116, 22,109, 7, 86, 59, 93, 62,110, 78, 99, 77, 67,

12,113, 87, 98,102, 5, 88, 33, 38, 56, 23, 8, 75, 45, 13, 75,

95, 63, 28, 49,123,120, 20,112, 44, 30, 15, 98,106, 2,103, 29,

82,107, 42,124, 24, 30, 41, 16,108,100,117, 40, 73, 40, 7,114,

82,115, 36,112, 12,102,100, 84, 92, 48, 72, 97, 9, 54, 55, 74,

113,123, 17, 26, 53, 58, 4, 9, 69,122, 21,118, 42, 60, 27, 73,

118,125, 34, 15, 65,115, 84, 64, 62, 81, 70, 1, 24,111,121, 83,

104, 81, 49,127, 48,105, 31, 10, 6, 91, 87, 37, 16, 54,116,126,

31, 38, 13, 0, 72,106, 77, 61, 26, 67, 46, 29, 96, 37, 61, 52,

101, 17, 44,108, 71, 52, 66, 57, 33, 51, 25, 90, 2,119,122, 35

};

static byte[] T2 = new byte[]{

52, 50, 44, 6, 21, 49, 41, 59, 39, 51, 25, 32, 51, 47, 52, 43,

37, 4, 40, 34, 61, 12, 28, 4, 58, 23, 8, 15, 12, 22, 9, 18,

55, 10, 33, 35, 50, 1, 43, 3, 57, 13, 62, 14, 7, 42, 44, 59,

62, 57, 27, 6, 8, 31, 26, 54, 41, 22, 45, 20, 39, 3, 16, 56,

48, 2, 21, 28, 36, 42, 60, 33, 34, 18, 0, 11, 24, 10, 17, 61,

29, 14, 45, 26, 55, 46, 11, 17, 54, 46, 9, 24, 30, 60, 32, 0,

20, 38, 2, 30, 58, 35, 1, 16, 56, 40, 23, 48, 13, 19, 19, 27,

31, 53, 47, 38, 63, 15, 49, 5, 37, 53, 25, 36, 63, 29, 5, 7

};

static byte[] T3 = new byte[]{

1, 5, 29, 6, 25, 1, 18, 23, 17, 19, 0, 9, 24, 25, 6, 31,

28, 20, 24, 30, 4, 27, 3, 13, 15, 16, 14, 18, 4, 3, 8, 9,

20, 0, 12, 26, 21, 8, 28, 2, 29, 2, 15, 7, 11, 22, 14, 10,

17, 21, 12, 30, 26, 27, 16, 31, 11, 7, 13, 23, 10, 5, 22, 19

};

static byte[] T4 = new byte[]{

15, 12, 10, 4, 1, 14, 11, 7, 5, 0, 14, 7, 1, 2, 13, 8,

10, 3, 4, 9, 6, 0, 3, 2, 5, 6, 8, 9, 11, 13, 15, 12

};

static byte Tj(int j, int index)

{

switch (j)

{

case 0: return T0[index];

case 1: return T1[index];

case 2: return T2[index];

case 3: return T3[index];

case 4: return T4[index];

}

return 0xff;

}

static void Comp128(byte[] RAND, byte[] Ki, ref int sres, ref Int64 Kc)

{

//http://prog.bobrodobro.ru/33840

byte[] x = new byte[32]; //256

BitArray bit = new BitArray(128); //128

int m, n, y, z;

Array.Copy(RAND, 0, x, 16, 16);

for (int i = 1; i < 8; ++i)

{

Array.Copy(Ki, 0, x, 0, 16);

for (int j =0; j < 4; ++j)

{

for (int k = 0; k < (int)Math.Pow(2, j) - 1; k++)

{

for (int l = 0; l < (int)Math.Pow(2, 4 - j) - 1; l++)

{

m = l + k \* (int)Math.Pow(2, 5 - j);

n = m + (int)Math.Pow(2, 4 - j);

y = (x[m] + 2 \* x[n]) % (int)Math.Pow(2, 9 - j);

z = (2 \* x[m] + x[n]) % (int)Math.Pow(2, 9 - j);

x[m] = Tj(j, y);

x[n] = Tj(j, z);

}

}

}

for (int j = 0; j < 31; ++j)

{

for (int k=0; k < 3; ++k)

{

//BitArray b = new BitArray(new byte[] { Convert.ToByte(j) });

var b = x[j];

var vv = new BitArray(new byte[] { b });

bit[4 \* j + k] = vv[3-k];

}

}

if (i<8)

{

for (int j =0; j < 15; ++j)

{

for (int k = 0; k < 7; ++k)

{

int index = ((8 \* j + k) \* 17) % 128;

x[j + 16] |= (byte) (Convert.ToByte(bit[index]) << (7 - k));

}

}

}

}

int[] SRES = new int[4]; // need only one

bit.CopyTo(SRES,0);

Int64 \_Kc = 0;

\_Kc = SRES[3];

\_Kc <<= 32;

\_Kc ^= SRES[2] ^ 0xfffff000;

Kc = \_Kc;

sres = SRES[3];

}

static void Main(string[] args)

{

// generate 128 bit ki

byte[] ki = Encoding.ASCII.GetBytes("MySecretPassword");//"MySecretPassword"

// generate 128bit RAND

byte[] rand = new byte[16];

//Random r = new Random();

// r.NextBytes(rand);

// generate Kc, sres

int sres = 0;

Int64 kc = 0;

//Comp128(rand, ki, ref sres, ref kc);

try

{

IPEndPoint ipPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 3333);

Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

// подключаемся к удаленному хосту

socket.Connect(ipPoint);

Console.WriteLine("Connected to server");

// получаем ответ

byte[] data = new byte[256]; // буфер для ответа

StringBuilder builder = new StringBuilder();

int bytes = 0; // количество полученных байт

//send tmsi

// get RAND

bytes = socket.Receive(data, data.Length, 0);

rand = data;

Console.WriteLine("Received rand:" + BitConverter.ToString(rand).Substring(0,47));

// make sres

Comp128(rand, ki, ref sres, ref kc);

Console.WriteLine("Make sres: {0}",sres);

// send sres

socket.Send(BitConverter.GetBytes(sres));

// receive result

bytes = socket.Receive(data, data.Length, 0);

if (Encoding.ASCII.GetString(data).Substring(0,2) == "ok")

{

Console.WriteLine("Server accepted client's sres");

Console.WriteLine("Set session key: {0}", kc );

string input = "close";

do

{

Console.Write(">");

input = Console.ReadLine();

var emsg = A5Encyptor(Encoding.ASCII.GetBytes(input), BitConverter.GetBytes(kc));

socket.Send(emsg);

bytes = socket.Receive(data);

byte[] tmp = new byte[bytes];

Array.Copy(data, 0, tmp, 0, bytes);

var dmsg = A5Encyptor(tmp, BitConverter.GetBytes(kc));

string res = Encoding.ASCII.GetString(dmsg);

res = res.Substring(0, res.IndexOf("\0"));

Console.WriteLine("Server response: " + res);

} while (input != "close");

}

else

{

Console.WriteLine("Auth fail");

socket.Shutdown(SocketShutdown.Both);

socket.Close();

Console.Read();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}