Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 2

**Идентификация и аутентификация пользователей. Протокол Kerberos.**

Выполнил студент гр. 853505

Лазарева Е.В.

Проверил

Протько М.И.

Минск, 2021

1. **Введение**

Целью данной лабораторной работы было изучить алгоритм шифрования данных DES и протокол Kerberos и создать приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую алгоритм DES.

1. **Теоретические сведения**

**Протокол Kerberos**

Протокол Kerberos является одной из реализаций протокола аутентификации с использованием третьей стороны, призванной уменьшить количество сообщений, которыми обмениваются стороны.



Рисунок 1 Схема протокола Kerberos

При использовании Kerberos нельзя напрямую получить доступ к какому-либо целевому серверу. Чтобы запустить собственно процедуру аутентификации, необходимо обратиться к специальному серверу аутентификации с запросом, содержащим логин пользователя. Если сервер не находит автора запроса в своей базе данных, запрос отклоняется. В противном случае сервер аутентификации работает по следующему рабочему процессу:

**Рабочий этап:**

Пусть клиент C собирается начать взаимодействие с сервером SS (англ. *Service* *Server* - *сервер*, предоставляющий сетевые сервисы). В несколько упрощенном виде, протокол предполагает следующие шаги:

1. **C->AS: {c}.**

Клиент C посылает серверу аутентификации AS свой идентификатор c (идентификатор передается открытым текстом).

1. **AS->C: {{TGT}KAS\_TGS, KC\_TGS}KC,**

где:

* KC - основной ключ C ;
* KC\_TGS - ключ, выдаваемый C для доступа к серверу выдачи разрешений *TGS* ;
* {TGT} - *Ticket* Granting *Ticket* - билет на доступ к серверу выдачи разрешений

{TGT}={c,*tgs*,t1,p1, KC\_TGS}, где *tgs* - идентификатор сервера выдачи разрешений, t1 - отметка времени, p1 - *период действия* билета.

Запись \{ \cdot \} K_{X}здесь и далее означает, что содержимое фигурных скобок зашифровано на ключе KX .

На этом шаге сервер аутентификации AS, проверив, что клиент C имеется в его базе, возвращает ему билет для доступа к серверу выдачи разрешений и ключ для взаимодействия с сервером выдачи разрешений. Вся посылка зашифрована на ключе клиента C. Таким образом, даже если на первом шаге взаимодействия идентификатор с послал не клиент С, а нарушитель X, то полученную от AS посылку X расшифровать не сможет.

Получить доступ к содержимому билета TGT не может не только нарушитель, но и клиент C, т.к. билет зашифрован на ключе, который распределили между собой сервер аутентификации и сервер выдачи разрешений.

1. **C->*****TGS*: {TGT}KAS\_TGS, {Aut1} KC\_TGS, {ID}**

где {Aut1} - аутентификационный блок - Aut1 = {с,t2}, t2 - метка времени; ID - идентификатор запрашиваемого сервиса (в частности, это может быть идентификатор сервера SS ).

Клиент C на этот раз обращается к серверу выдачи разрешений ТGS. Он пересылает полученный от AS билет, зашифрованный на ключе KAS\_TGS, и аутентификационный блок, содержащий идентификатор c и метку времени, показывающую, когда была сформирована посылка. Сервер выдачи разрешений расшифровывает билет TGT и получает из него информацию о том, кому был выдан билет, когда и на какой срок, ключ шифрования, сгенерированный сервером AS для взаимодействия между клиентом C и сервером *TGS*. С помощью этого ключа расшифровывается аутентификационный блок. Если метка в блоке совпадает с меткой в билете, это доказывает, что посылку сгенерировал на самом деле С (ведь только он знал ключ KC\_TGS и мог правильно зашифровать свой идентификатор). Далее делается проверка времени действия билета и времени отправления посылки **3**). Если проверка проходит и действующая в системе политика позволяет клиенту С обращаться к клиенту SS, тогда выполняется шаг **4**).

1. ***TGS*->C: {{*****TGS*}KTGS\_SS,KC\_SS}KC\_TGS,**

где KC\_SS - ключ для взаимодействия C и SS, {*TGS*} - *Ticket* Granting Service - билет для доступа к SS (обратите внимание, что такой же аббревиатурой в описании протокола обозначается и сервер выдачи разрешений). {*TGS*} ={с,ss,t3,p2, KC\_SS }.

Сейчас сервер выдачи разрешений *TGS* посылает клиенту C ключ шифрования и билет, необходимые для доступа к серверу SS. Структура билета такая же, как на шаге 2): идентификатор того, кому выдали билет; идентификатор того, для кого выдали билет; отметка времени; *период действия*; ключ шифрования.

1. **C->SS: {*****TGS*}KTGS\_SS, {Aut2} KC\_SS**

где Aut2={c,t4}.

Клиент C посылает билет, полученный от сервера выдачи разрешений, и свой аутентификационный блок серверу SS, с которым хочет установить сеанс защищенного взаимодействия. Предполагается, что SS уже зарегистрировался в системе и распределил с сервером *TGS* ключ шифрования KTGS\_SS. Имея этот ключ, он может расшифровать билет, получить ключ шифрования KC\_SS и проверить подлинность *отправителя сообщения*.

1. **SS->C: {t4+1}KC\_SS**

Смысл последнего шага заключается в том, что теперь уже SS должен доказать C свою подлинность. Он может сделать это, показав, что правильно расшифровал предыдущее сообщение. Вот поэтому, SS берет отметку времени из аутентификационного блока C, изменяет ее заранее определенным образом (увеличивает на 1), шифрует на ключе KC\_SS и возвращает C.

Если все шаги выполнены правильно и все проверки прошли успешно, то стороны взаимодействия C и SS, во-первых, удостоверились в подлинности друг друга, а во-вторых, получили *ключ* шифрования для защиты сеанса связи - *ключ* KC\_SS.

В процессе сеанса работы клиент проходит шаги 1) и 2) только один раз. Когда нужно получить билет на *доступ* к другому серверу (назовем его SS1 ), клиент С обращается к серверу выдачи разрешений *TGS* с уже имеющимся у него билетом, т.е. протокол выполняется начиная с шага 3).

**Алгоритм DES Основные сведения**

Стандарт DES построен на комбинированном использовании перестановки, замены и гаммирования. Шифруемые данные должны быть представлены в двоичном виде.

*DES* является классической *сетью Фейстеля* с двумя ветвями. Данные шифруются 64-битными блоками, используя 56-битный ключ. Алгоритм преобразует за несколько *раундов* 64-битный вход в 64-битный выход. Длина ключа равна 56 битам. Процесс шифрования состоит из четырех этапов. На первом из них выполняется начальная перестановка (*IP*) 64-битного исходного текста (забеливание), во время которой биты переупорядочиваются в соответствии со стандартной таблицей. Следующий этап состоит из 16 *раундов* одной и той же функции, которая использует операции сдвига и подстановки. На третьем этапе левая и правая половины выхода последней (16-й) итерации меняются местами. Наконец, на четвертом этапе выполняется перестановка IP-1 результата, полученного на третьем этапе. Перестановка IP-1 инверсна начальной перестановке.

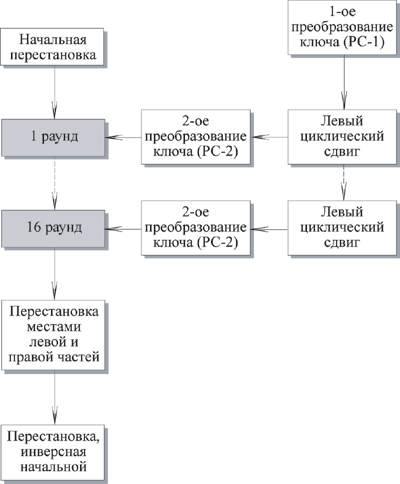


Рисунок 2 Общая схема DES

**Шифрование**

**Начальная перестановка**

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если М- это произвольные 64 бита, то X = IP (M)-переставленные 64 бита. Если применить обратную функцию перестановки Y = IP-1 (X) = IP-1 (IP(M)), то получится первоначальная последовательность бит.

**Последовательность преобразований отдельного раунда**

Теперь рассмотрим последовательность преобразований, используемую в каждом *раунде*.

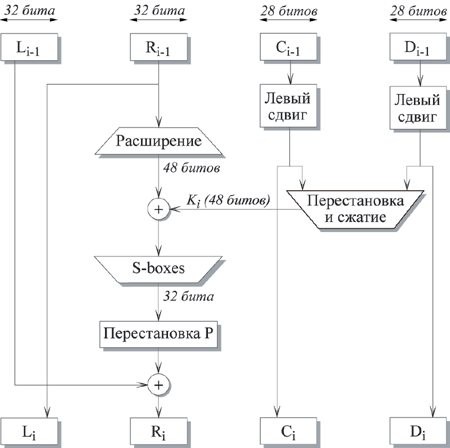


Рисунок 3 - I-ый раунд DES

64-битный входной блок проходит через 16 *раундов*, при этом на каждой итерации получается промежуточное 64-битное значение. Левая и правая части каждого промежуточного значения трактуются как отдельные 32-битные значения, обозначенные *L* и *R*. Каждую итерацию можно описать следующим образом:

Li = Ri-1

Ri = Li-1  F(Ri-1, Ki)

Где обозначает операцию XOR.

Таким образом, выход левой половины *Li* равен входу правой половины *Ri-1*. Выход правой половины *Ri*является результатом применения операции XOR к *Li-1* и функции *F*, зависящей от *Ri-1* и *Ki*.

Рассмотрим функцию *F* более подробно.

*Ri*, которое подается на вход функции *F*, имеет длину 32 бита. Вначале Ri расширяется до 48 бит, используя таблицу, которая определяет перестановку плюс расширение на 16 бит. Расширение происходит следующим образом. 32 бита разбиваются на группы по 4 бита и затем расширяются до 6 бит, присоединяя крайние биты из двух соседних групп.

После этого для полученного 48-битного значения выполняется операция XOR с 48-битным *подключом Ki*. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32-битное значение.

Подстановка состоит из восьми *S-boxes,* каждый из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения *S-box* определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход.

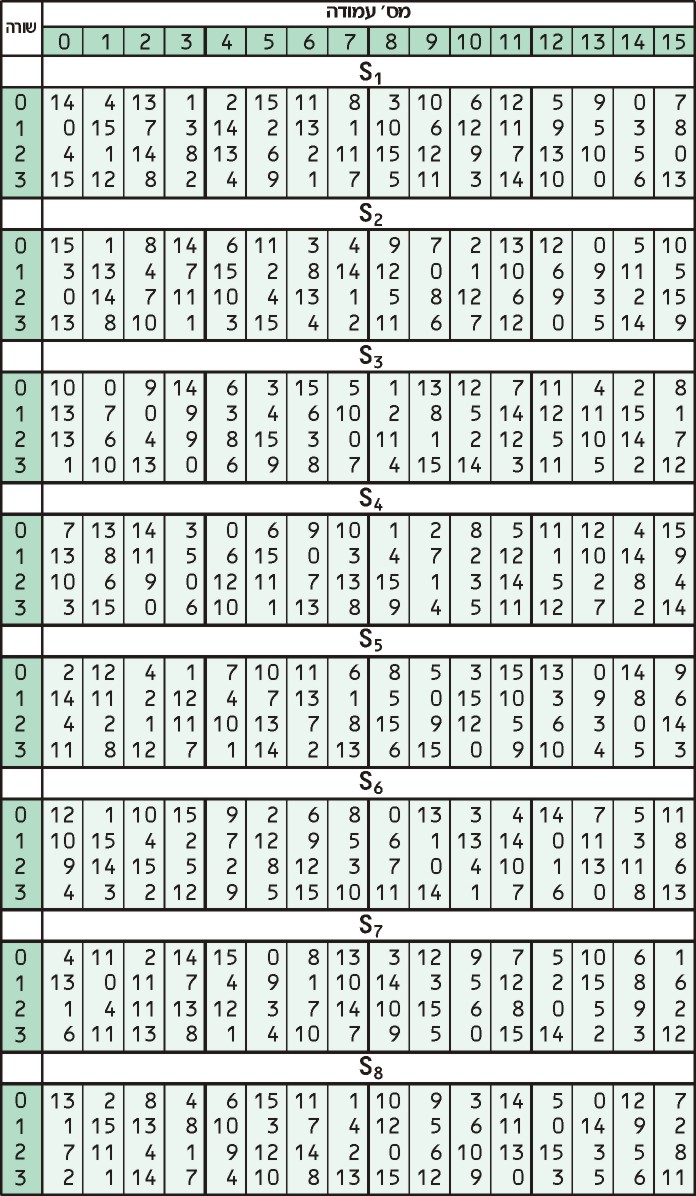


Рисунок 4 - S-boxes

Далее полученное 32-битное значение обрабатывается с помощью перестановки *Р*, целью которой является максимальное переупорядочивание бит, чтобы в следующем *раунде* шифрования с большой вероятностью каждый бит обрабатывался другим *S-box*.

**Создание подключей**

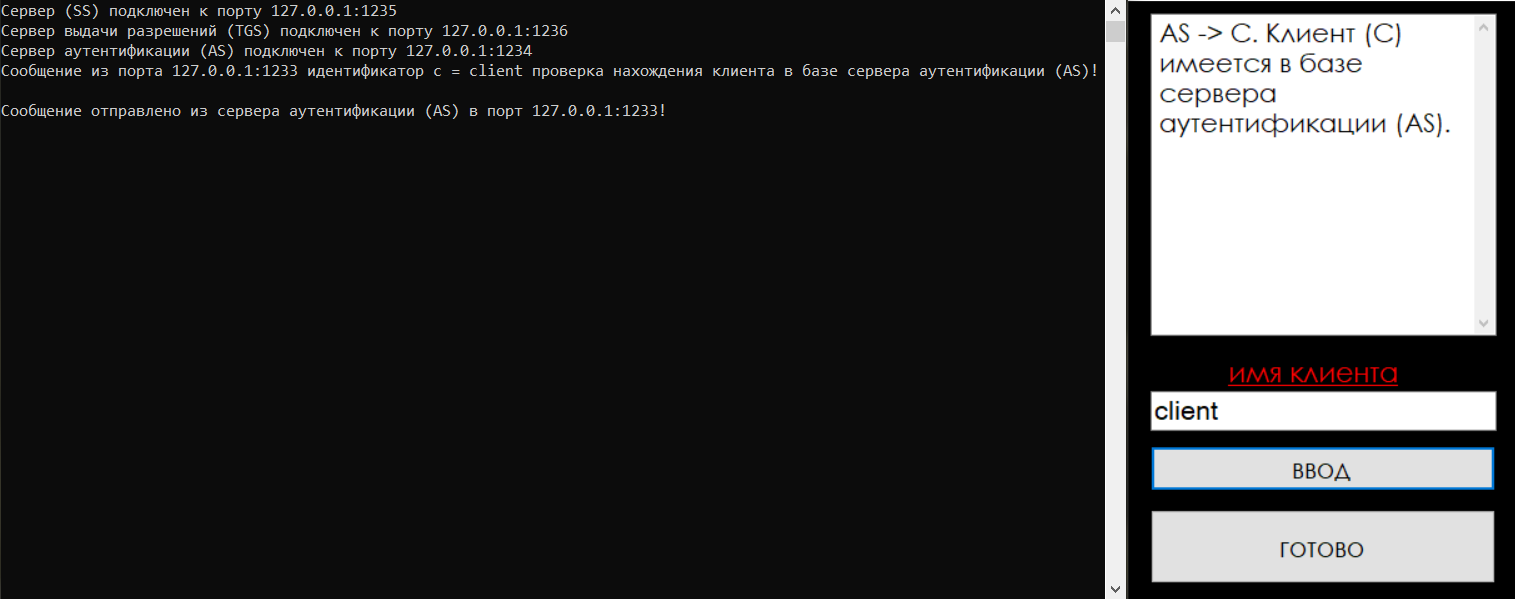
Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма, вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1). Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как C0 и D0 соответственно. На каждом *раунде Ci* и *Di*независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *раунда*. Полученные значения являются входом следующего *раунда*. Они также представляют собой вход в Permuted Choice 2 (РС-2), который создает 48-битное выходное значение, являющееся входом функции *F*(*Ri-1*, *Ki*).

**Дешифрование**

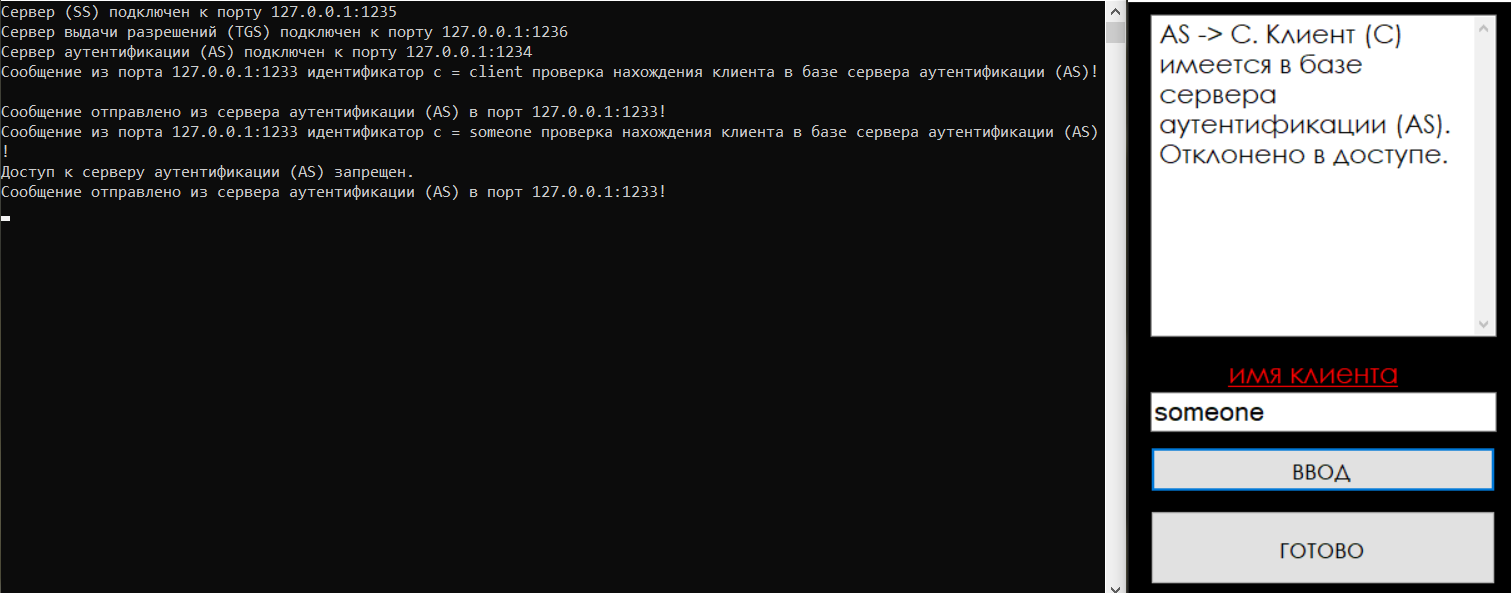
Процесс дешифрования аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности. *K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

1. **Результат работы программы**

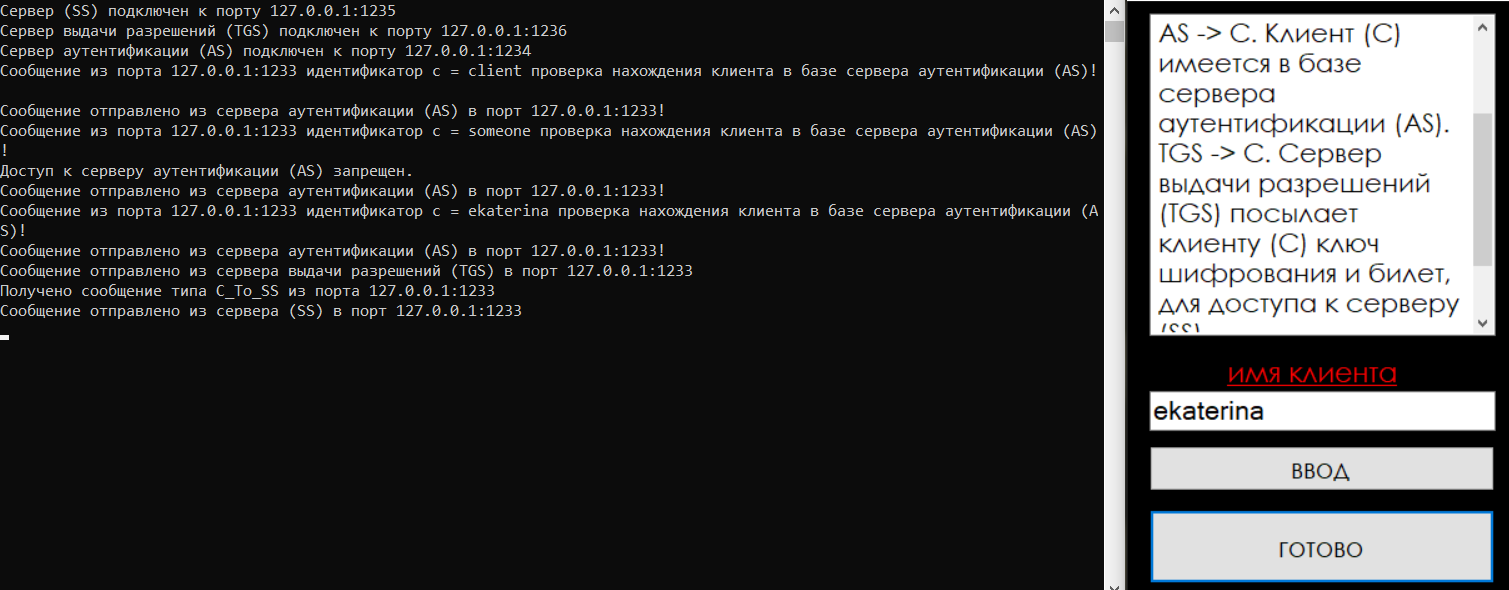
Получение ключей TGS при правильном логине:



Отказ в доступе при неверном логине:



Обработка запроса при обращении к SS :



1. **Код программы**

**Lab2\_Server**

AuthServer.cs

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

namespace Lab2\_Server

{

class AuthServer

{

private readonly List<string> Users = new List<string>();

public void Listener()

{

UdpClient reciever = new UdpClient(Config.AS\_port);

Console.WriteLine($"Сервер аутентификации (AS) подключен к порту 127.0.0.1:{Config.AS\_port}");

IPEndPoint IP = null;

try

{

while (true)

{

byte[] data = reciever.Receive(ref IP);

IP.Port = Config.C\_port;

Message ReMessage = new Message();

var message = Serilizer<Message>.Deserilaze(data);

if (message.Type == MessageType.C\_To\_AS)

{

var id = Helper.ToString(message.Data[0].ToArray());

Console.WriteLine($"Сообщение из порта {IP.Address}:{IP.Port} идентификатор c = {id} проверка нахождения клиента в базе сервера аутентификации (AS)!");

if (Users.Contains(id))

{

ReMessage.Type = MessageType.AS\_To\_C;

TicketGranting ticket = new TicketGranting()

{

ClientIdentity = id,

Duration = Config.AS\_Duration.Ticks,

IssuingTime = DateTime.Now,

ServiceIdentity = Config.TGS,

Key = Helper.ToString(Config.C\_To\_TGS\_Key)

};

var ticket\_bytes = Helper.ExtendData(Serilizer<TicketGranting>.Serilize(ticket));

var c\_to\_tgs\_key\_bytes = Helper.ExtendData(Config.C\_To\_TGS\_Key);

var tb\_enc = DES.Encrypt(ticket\_bytes, Config.AS\_To\_TGS\_Key);

tb\_enc = DES.Encrypt(tb\_enc, Config.C\_Key);

var c\_to\_tgs\_key\_enc = DES.Encrypt(c\_to\_tgs\_key\_bytes, Config.C\_Key);

ReMessage.Data.Add(new List<byte>(tb\_enc));

ReMessage.Data.Add(new List<byte>(c\_to\_tgs\_key\_enc));

}

else

{

ReMessage.Type = MessageType.AccessDenied;

Console.WriteLine("Доступ к серверу аутентификации (AS) запрещен.");

}

ReMessage.Send(IP);

Console.WriteLine($"Сообщение отправлено из сервера аутентификации (AS) в порт {IP.Address}:{IP.Port}!");

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

}

public AuthServer()

{

Users.Add("ekaterina");

Users.Add("client");

}

}

}

Config.cs

using System;

namespace Lab2\_Server

{

public static class Config

{

public static readonly int C\_port = 1233;

public static readonly int AS\_port = 1234;

public static readonly int SS\_port = 1235;

public static readonly int TGS\_port = 1236;

public static readonly byte[] C\_Key = Helper.ExtendKey("C\_Key");

public static readonly byte[] C\_To\_TGS\_Key = Helper.ExtendKey("C\_To\_TGS\_Key");

public static readonly byte[] C\_To\_SS\_Key = Helper.ExtendKey("C\_To\_SS\_Key");

public static readonly byte[] AS\_To\_TGS\_Key = Helper.ExtendKey("AS\_To\_TGS\_Key");

public static readonly byte[] TGS\_To\_SS\_Key = Helper.ExtendKey("TGS\_To\_SS\_Key");

public static readonly TimeSpan AS\_Duration = new TimeSpan(24, 0, 0);

public static readonly TimeSpan TGS\_Duration = new TimeSpan(12, 0, 0);

public static readonly string TGS = "TGS";

public static readonly string id\_SS = "id\_SS";

}

}

DES.cs

using System.Collections;

namespace Lab2\_Server

{

class DES

{

private static readonly byte Rouns = 16;

public static BitArray last\_key;

static readonly byte[] E = new byte[]

{

32, 1, 2, 3, 4, 5,

4, 5, 6, 7, 8, 9,

8, 9, 10, 11, 12, 13,

12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21,

20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29,

28, 29, 30, 31, 32, 1

};

static readonly byte[,] S = new byte[,]

{

{

14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,

0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,

4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,

15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13

},

{

15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,

3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,

0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,

13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9

},

{

10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,

13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,

13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,

1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12

},

{

7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,

13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,

10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,

3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14

},

{

2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,

14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,

4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,

11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3

},

{

12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,

10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,

9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,

4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13

},

{

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,

13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,

1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,

6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12

},

{

13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,

1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,

7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,

2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

}

};

static readonly byte[] CD = new byte[]

{

57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

};

static readonly byte[] IP = new byte[]

{

58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7

};

static readonly byte[] IP\_reverce = new byte[]

{

40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25

};

static readonly byte[] Choice\_key = new byte[]

{

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4,

26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40,

51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

};

static readonly byte[] P = new byte[]

{

16, 7, 20, 21, 29, 12, 28, 17,

1, 15, 23, 26, 5, 18, 31, 10,

2, 8, 24, 14, 32, 27, 3, 9,

19, 13, 30, 6, 22, 11, 4, 25

};

static readonly byte[] Shifts = new byte[]

{

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

private static BitArray GetBitArray(byte data)

{

byte[] bytes = new byte[1] { data };

return new BitArray(bytes);

}

private static BitArray RightShift(BitArray data, int count)

{

BitArray temp = new BitArray(count);

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

{

temp[i] = data[i];

}

BitArray res = ((BitArray)data.Clone()).RightShift(count);

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

{

res[data.Length - count + i] = temp[i];

}

return res;

}

private static BitArray LeftShift(BitArray data, int count)

{

BitArray temp = new BitArray(count);

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

{

temp[i] = data[data.Length - count + i];

}

BitArray res = ((BitArray)data.Clone()).LeftShift(count);

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

{

res[i] = temp[i];

}

return res;

}

private static void GetLR(BitArray data, out BitArray Left, out BitArray Right)

{

Left = new BitArray(data.Length / 2);

Right = new BitArray(data.Length / 2);

for (int i = 0; i < data.Length / 2; i++)

{

Left[i] = data[i];

}

for (int i = data.Length / 2; i < data.Length; i++)

{

Right[i - data.Length / 2] = data[i];

}

}

private static BitArray F(BitArray R, BitArray k)

{

BitArray R\_extend = Permutation(R, E);

R\_extend = R\_extend.Xor(k);

BitArray B\_res = new BitArray(32);

for (int i = 0; i < 8; i++)

{

BitArray B = new BitArray(6);

for (int j = 0; j < 6; j++)

{

B[j] = R\_extend[i \* 6 + j];

}

GetAB(B, out byte a, out byte b);

byte s = S[i, a \* 16 + b];

BitArray s\_bits = GetBitArray(s);

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

{

B\_res[i \* 4 + t] = s\_bits[t];

}

}

B\_res = Permutation(B\_res, P);

return B\_res;

}

private static BitArray Permutation(BitArray data, byte[] map)

{

BitArray res = new BitArray(map.Length);

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

{

res[i] = data[map[i] - 1];

}

return res;

}

private static byte[] GetBytes(BitArray data)

{

byte[] bytes = new byte[data.Length / 8];

data.CopyTo(bytes, 0);

return bytes;

}

private static byte GetByte(BitArray data)

{

if (data.Length <= 8)

{

byte[] bytes = new byte[1];

data.CopyTo(bytes, 0);

return bytes[0];

}

else return 0;

}

private static void GetAB(BitArray R, out byte a, out byte b)

{

BitArray A = new BitArray(2);

BitArray B = new BitArray(4);

A[0] = R[0];

A[1] = R[^1];

for (int i = 1; i < R.Length - 1; i++)

{

B[i - 1] = R[i];

}

a = GetByte(A);

b = GetByte(B);

}

private static BitArray Merge(BitArray L, BitArray R)

{

BitArray res = new BitArray(L.Length \* 2);

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

{

res[i] = L[i];

res[L.Length + i] = R[i];

}

return res;

}

private static BitArray ExtendKey(BitArray key)

{

BitArray res = new BitArray(64);

for (int i = 0; i < 8; i++)

{

int count = 0;

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

{

if (key[i \* 7 + j])

count++;

res[i \* 8 + j] = key[i \* 7 + j];

}

res[i \* 8 + 7] = count % 2 == 0;

}

return res;

}

private static BitArray GenereteNextKey(BitArray key, int shift\_count)

{

GetLR(key, out BitArray L, out BitArray R);

return Merge(LeftShift(L, shift\_count), LeftShift(R, shift\_count));

}

private static BitArray GeneretePrevKey(BitArray key, int shift\_count)

{

GetLR(key, out BitArray L, out BitArray R);

return Merge(RightShift(L, shift\_count), RightShift(R, shift\_count));

}

private static BitArray GetFirstKey(byte[] key)

{

BitArray byte\_key = new BitArray(key);

return Permutation(ExtendKey(byte\_key), CD);

}

private static BitArray GetLastKey(byte[] key)

{

BitArray K = GetFirstKey(key);

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

{

K = GenereteNextKey(K, Shifts[i]);

}

return K;

}

public static byte[] Encrypt(byte[] data, byte[] key)

{

BitArray bitKey = GetFirstKey(key);

byte[] res = new byte[data.Length];

for (int i = 0; i < data.Length; i += 8)

{

byte[] block = new byte[8];

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

{

if (j >= data.Length)

break;

block[j - i] = data[j];

}

BitArray bit\_block = new BitArray(block);

bit\_block = Permutation(bit\_block, IP);

GetLR(bit\_block, out BitArray L, out BitArray R);

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

{

BitArray K = GenereteNextKey(bitKey, Shifts[k]);

BitArray temp\_R = new BitArray(R);

R = L.Xor(F(R, Permutation(K, Choice\_key)));

L = temp\_R;

bitKey = K;

}

BitArray kek = Merge(L, R);

BitArray bit\_res = Permutation(kek, IP\_reverce);

byte[] res\_block = GetBytes(bit\_res);

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

{

res[j] = res\_block[j - i];

}

last\_key = bitKey;

}

return res;

}

public static byte[] Decrypt(byte[] data, byte[] key)

{

BitArray bitKey = GetLastKey(key);

byte[] res = new byte[data.Length];

for (int i = 0; i < data.Length; i += 8)

{

byte[] block = new byte[8];

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

{

block[j - i] = data[j];

}

BitArray bit\_block = new BitArray(block);

bit\_block = Permutation(bit\_block, IP);

GetLR(bit\_block, out BitArray L, out BitArray R);

BitArray K = bitKey;

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

{

BitArray temp\_L = new BitArray(L);

L = R.Xor(F(L, Permutation(K, Choice\_key)));

R = temp\_L;

K = GeneretePrevKey(bitKey, Shifts[Shifts.Length - 1 - k]);

bitKey = K;

}

BitArray kek = Merge(L, R);

BitArray bit\_res = Permutation(kek, IP\_reverce);

byte[] res\_block = GetBytes(bit\_res);

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

{

res[j] = res\_block[j - i];

}

}

return res;

}

}

}

Helper.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab2\_Server

{

class Helper

{

public static byte[] RecoverData(List<byte> data)

{

int i = data.Count - 1;

while (data[i] == 0)

{

data.RemoveRange(i, data.Count - i);

i--;

}

return data.ToArray();

}

public static byte[] ExtendData(byte[] data)

{

int diff = 8 - (data.Length % 8);

byte[] res = new byte[data.Length + diff];

data.CopyTo(res, 0);

return res;

}

public static string ToString(byte[] source)

{

return Encoding.UTF8.GetString(source);

}

public static byte[] ToByteArray(string str)

{

return Encoding.UTF8.GetBytes(str);

}

public static byte[] ExtendKey(string data)

{

StringBuilder stringBuilder = new StringBuilder(data);

if (data.Length > 7)

{

return ToByteArray(stringBuilder.Remove(7, data.Length - 7).ToString());

}

int diff = 7 - data.Length;

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

{

stringBuilder.Append(data[i % data.Length]);

}

return ToByteArray(stringBuilder.ToString());

}

public static bool CheckTime(DateTime t1, DateTime t2, long duration)

{

TimeSpan ts = new TimeSpan(duration);

if (t2 < t1 + ts)

{

return true;

}

return false;

}

}

}

Message.cs

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

namespace Lab2\_Server

{

class Message

{

public Message(MessageType messageType = 0)

{

Type = messageType;

Data = new List<List<byte>>();

}

public Message() { Data = new List<List<byte>>(); }

public MessageType Type { get; set; }

public List<List<byte>> Data { get; set; }

public void Send(IPEndPoint remoteIP)

{

UdpClient sender = new UdpClient();

try

{

byte[] dgram = Serilizer<Message>.Serilize(this);

sender.Send(dgram, dgram.Length, remoteIP);

}

finally

{

sender.Close();

}

}

public override string ToString()

{

return $"Получено сообщение типа {Type} ";

}

}

}

MessageType.cs

namespace Lab2\_Server

{

enum MessageType

{

C\_To\_AS,

AS\_To\_C,

С\_To\_TGS,

TGS\_To\_C,

C\_To\_SS,

SS\_To\_C,

TicketNotValid,

AccessDenied

}

}

Serilizer.cs

using System.Text;

using System.Text.Json;

namespace Lab2\_Server

{

class Serilizer<T>

{

public static byte[] Serilize(T source)

{

var json = JsonSerializer.Serialize<T>(source);

return Encoding.UTF8.GetBytes(json);

}

public static T Deserilaze(byte[] source)

{

var json = Encoding.UTF8.GetString(source);

return JsonSerializer.Deserialize<T>(json);

}

}

}

Server.cs

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

namespace Lab2\_Server

{

class Server

{

public void Listener()

{

Console.WriteLine($"Сервер (SS) подключен к порту 127.0.0.1:{Config.SS\_port}");

UdpClient reciever = new UdpClient(Config.SS\_port);

IPEndPoint IP = null;

try

{

while (true)

{

byte[] data = reciever.Receive(ref IP);

IP.Port = Config.C\_port;

var message = Serilizer<Message>.Deserilaze(data);

Console.WriteLine(message.ToString() + $"из порта {IP.Address}:{IP.Port}");

if (message.Type == MessageType.C\_To\_SS)

{

var tgs\_bytes = Helper.RecoverData(

new List<byte>(DES.Decrypt(message.Data[0].ToArray(), Config.TGS\_To\_SS\_Key)));

var tgs = Serilizer<TicketGranting>.Deserilaze(tgs\_bytes);

var c = Helper.RecoverData(

new List<byte>(DES.Decrypt(message.Data[1].ToArray(), Config.C\_To\_SS\_Key)));

var c2 = Serilizer<TimeMark>.Deserilaze(c);

Message ReMessage = new Message();

if (Helper.CheckTime(tgs.IssuingTime, c2.T, tgs.Duration))

{

ReMessage.Type = MessageType.SS\_To\_C;

DateTime reTime = c2.T;

var time\_bytes = Serilizer<long>.Serilize(c2.T.Ticks + 1);

var bytes = DES.Encrypt(Helper.ExtendData(time\_bytes),

Config.C\_To\_SS\_Key);

ReMessage.Data.Add(new List<byte>(bytes));

}

else ReMessage.Type = MessageType.TicketNotValid;

ReMessage.Send(IP);

Console.WriteLine($"Сообщение отправлено из сервера (SS) в порт {IP.Address}:{IP.Port}");

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

}

}

}

TicketGranting.cs

using System;

namespace Lab2\_Server

{

[Serializable]

class TicketGranting

{

public TicketGranting() { }

public string ClientIdentity { get; set; }

public string ServiceIdentity { get; set; }

public DateTime IssuingTime { get; set; }

public long Duration { get; set; }

public string Key { get; set; }

}

}

TicketGrantingServer.cs

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

namespace Lab2\_Server

{

class TicketGrantingServer

{

public void Listener()

{

UdpClient reciever = new UdpClient(Config.TGS\_port);

Console.WriteLine($"Сервер выдачи разрешений (TGS) подключен к порту 127.0.0.1:{Config.TGS\_port}");

IPEndPoint IP = null;

try

{

while (true)

{

byte[] data = reciever.Receive(ref IP);

IP.Port = Config.C\_port;

var message = Serilizer<Message>.Deserilaze(data);

if (message.Type == MessageType.С\_To\_TGS)

{

var tgt\_json = Helper.RecoverData(

new List<byte>(DES.Decrypt(message.Data[0].ToArray(), Config.AS\_To\_TGS\_Key)));

var tgt = Serilizer<TicketGranting>.Deserilaze(tgt\_json);

var c\_json = Helper.RecoverData(

new List<byte>(DES.Decrypt(message.Data[1].ToArray(), Config.C\_To\_TGS\_Key)));

var a = Helper.ToString(c\_json);

var c = Serilizer<TimeMark>.Deserilaze(c\_json);

var id = Helper.ToString(message.Data[2].ToArray());

Message ReMessage = new Message();

if (tgt.ClientIdentity == c.C)

{

if (Helper.CheckTime(tgt.IssuingTime, c.T, tgt.Duration))

{

ReMessage.Type = MessageType.TGS\_To\_C;

var TGS = new TicketGranting()

{

ClientIdentity = c.C,

ServiceIdentity = id,

Duration = Config.TGS\_Duration.Ticks,

IssuingTime = DateTime.Now,

Key = Helper.ToString(Config.C\_To\_SS\_Key)

};

var ticket\_bytes = Helper.ExtendData(Serilizer<TicketGranting>.Serilize(TGS));

var c\_to\_ss\_key\_bytes = Helper.ExtendData(Config.C\_To\_SS\_Key);

var tb\_enc = DES.Encrypt(ticket\_bytes, Config.TGS\_To\_SS\_Key);

tb\_enc = DES.Encrypt(tb\_enc, Config.C\_To\_TGS\_Key);

var c\_to\_ss\_key\_enc = DES.Encrypt(c\_to\_ss\_key\_bytes, Config.C\_To\_TGS\_Key);

ReMessage.Data.Add(new List<byte>(tb\_enc));

ReMessage.Data.Add(new List<byte>(c\_to\_ss\_key\_enc));

}

else

{

ReMessage.Type = MessageType.TicketNotValid;

Console.WriteLine("Билет в сервере выдачи разрешений(TGS) недействителен.");

}

}

else

{

ReMessage.Type = MessageType.AccessDenied;

Console.WriteLine("Доступ к серверу выдачи разрешений (TGS) запрещен.");

}

ReMessage.Send(IP);

Console.WriteLine($"Сообщение отправлено из сервера выдачи разрешений (TGS) в порт {IP.Address}:{IP.Port}");

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

}

}

}

TimeMark.cs

using System;

namespace Lab2\_Server

{

class TimeMark

{

public string C { get; set; }

public DateTime T { get; set; }

}

}

Program.cs

using System;

using System.Threading.Tasks;

namespace Lab2\_Server

{

class Program

{

static void Main(string[] args)

{

AuthServer AS = new AuthServer();

TicketGrantingServer TGS = new TicketGrantingServer();

Server SS = new Server();

try

{

Task.Run(() => AS.Listener());

Task.Run(() => TGS.Listener());

Task.Run(() => SS.Listener());

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

Console.ReadLine();

}

}

}

**Lab2\_Client**

Client.cs

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Windows.Forms;

namespace Lab2\_Client

{

class Client

{

public string Login { get; set; }

private byte[] TicketGrantingTicket { get; set; }

private byte[] TicketGrantingService { get; set; }

private byte[] C\_To\_TGS\_Key { get; set; }

private byte[] C\_To\_SS\_Key { get; set; }

DateTime T4 { get; set; }

readonly TextBox \_textBox;

public Client(TextBox textBox)

{

\_textBox = textBox;

}

private readonly IPEndPoint ASEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Config.AS\_port);

private readonly IPEndPoint SSEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Config.SS\_port);

private readonly IPEndPoint TGSEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Config.TGS\_port);

public void Register(string login)

{

Login = login;

Message message = new Message(MessageType.C\_To\_AS);

message.Data.Add(new List<byte>(Helper.ToByteArray(login)));

message.Send(ASEndPoint);

}

public void Listener()

{

UdpClient reciever = new UdpClient(Config.C\_port);

IPEndPoint IP = null;

try

{

while (true)

{

byte[] data = reciever.Receive(ref IP);

Message message = Serilazer<Message>.Deserilaze(data);

switch (message.Type)

{

case MessageType.AS\_To\_C:

TicketGrantingTicket = DES.Decrypt(message.Data[0].ToArray(), Config.C\_Key);

C\_To\_TGS\_Key = Helper.RecoverData(new List<byte>(DES.Decrypt(message.Data[1].ToArray(), Config.C\_Key)));

var a = Helper.ToString(C\_To\_TGS\_Key);

Print("AS -> C. Клиент (C) имеется в базе сервера аутентификации (AS).");

break;

case MessageType.TGS\_To\_C:

TicketGrantingService = DES.Decrypt(message.Data[0].ToArray(), C\_To\_TGS\_Key);

C\_To\_SS\_Key = Helper.RecoverData(new List<byte>(DES.Decrypt(message.Data[1].ToArray(), C\_To\_TGS\_Key)));

Message msg = new Message(MessageType.C\_To\_SS);

msg.Data.Add(new List<byte>(TicketGrantingService));

var mark = new TimeMark() { C = Login, T = DateTime.Now };

var Aut2 = Helper.ExtendData(Serilazer<TimeMark>.Serilize(mark));

T4 = mark.T;

msg.Data.Add(new List<byte>(DES.Encrypt(Aut2, C\_To\_SS\_Key)));

msg.Send(SSEndPoint);

Print("TGS -> C. Сервер выдачи разрешений (TGS) посылает клиенту (C) ключ шифрования и билет, для доступа к серверу (SS).");

break;

case MessageType.SS\_To\_C:

var t = DES.Decrypt(message.Data[0].ToArray(), C\_To\_SS\_Key);

var checkT\_bytes = Helper.RecoverData(new List<byte>(t));

var asd = Helper.ToString(checkT\_bytes);

var checkT = Serilazer<long>.Deserilaze(checkT\_bytes);

if (T4.Ticks + 1 == checkT)

{

Print("Выполнено!");

}

break;

case MessageType.TicketNotValid:

Print("Билет недействителен.");

break;

case MessageType.AccessDenied:

Print("Отклонено в доступе.");

break;

default:

MessageBox.Show("Неверный тип сообщения.");

break;

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

public void GetRes()

{

if (C\_To\_TGS\_Key != null &&

TicketGrantingTicket != null)

{

Message message = new Message(MessageType.С\_To\_TGS);

message.Data.Add(new List<byte>(TicketGrantingTicket));

TimeMark mark = new TimeMark() { C = Login, T = DateTime.Now };

var Aut1 = Helper.ExtendData(Serilazer<TimeMark>.Serilize(mark));

message.Data.Add(new List<byte>(DES.Encrypt(Aut1, C\_To\_TGS\_Key)));

message.Data.Add(new List<byte>(Helper.ToByteArray(Config.id\_SS)));

message.Send(TGSEndPoint);

}

}

public void Print(string message)

{

\_textBox.Invoke(new Action(() =>

{

\_textBox.AppendText(message + "\r\n");

}));

}

}

}

Config.cs

namespace Lab2\_Client

{

public static class Config

{

public static readonly byte[] C\_Key = Helper.ExtendKey("C\_Key");

public static readonly int C\_port = 1233;

public static readonly int AS\_port = 1234;

public static readonly int SS\_port = 1235;

public static readonly int TGS\_port = 1236;

public static readonly string tgs = "TGS";

public static readonly string id\_SS = "id\_SS";

}

}

DES.cs

using System.Collections;

namespace Lab2\_Client

{

class DES

{

private static readonly byte Rouns = 16;

public static BitArray last\_key;

static readonly byte[] E = new byte[]

{

32, 1, 2, 3, 4, 5,

4, 5, 6, 7, 8, 9,

8, 9, 10, 11, 12, 13,

12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21,

20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29,

28, 29, 30, 31, 32, 1

};

static readonly byte[,] S = new byte[,]

{

{

14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,

0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,

4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,

15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13

},

{

15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,

3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,

0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,

13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9

},

{

10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,

13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,

13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,

1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12

},

{

7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,

13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,

10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,

3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14

},

{

2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,

14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,

4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,

11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3

},

{

12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,

10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,

9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,

4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13

},

{

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,

13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,

1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,

6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12

},

{

13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,

1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,

7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,

2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

}

};

static readonly byte[] CD = new byte[]

{

57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

};

static readonly byte[] IP = new byte[]

{

58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7

};

static readonly byte[] IP\_reverce = new byte[]

{

40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25

};

static readonly byte[] Choice\_key = new byte[]

{

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4,

26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40,

51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

};

static readonly byte[] P = new byte[]

{

16, 7, 20, 21, 29, 12, 28, 17,

1, 15, 23, 26, 5, 18, 31, 10,

2, 8, 24, 14, 32, 27, 3, 9,

19, 13, 30, 6, 22, 11, 4, 25

};

static readonly byte[] Shifts = new byte[]

{

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

private static BitArray GetBitArray(byte data)

{

byte[] bytes = new byte[1] { data };

return new BitArray(bytes);

}

private static BitArray RightShift(BitArray data, int count)

{

BitArray temp = new BitArray(count);

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

{

temp[i] = data[i];

}

BitArray res = ((BitArray)data.Clone()).RightShift(count);

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

{

res[data.Length - count + i] = temp[i];

}

return res;

}

private static BitArray LeftShift(BitArray data, int count)

{

BitArray temp = new BitArray(count);

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

{

temp[i] = data[data.Length - count + i];

}

BitArray res = ((BitArray)data.Clone()).LeftShift(count);

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

{

res[i] = temp[i];

}

return res;

}

private static void GetLR(BitArray data, out BitArray Left, out BitArray Right)

{

Left = new BitArray(data.Length / 2);

Right = new BitArray(data.Length / 2);

for (int i = 0; i < data.Length / 2; i++)

{

Left[i] = data[i];

}

for (int i = data.Length / 2; i < data.Length; i++)

{

Right[i - data.Length / 2] = data[i];

}

}

private static BitArray F(BitArray R, BitArray k)

{

BitArray R\_extend = Permutation(R, E);

R\_extend = R\_extend.Xor(k);

BitArray B\_res = new BitArray(32);

for (int i = 0; i < 8; i++)

{

BitArray B = new BitArray(6);

for (int j = 0; j < 6; j++)

{

B[j] = R\_extend[i \* 6 + j];

}

GetAB(B, out byte a, out byte b);

byte s = S[i, a \* 16 + b];

BitArray s\_bits = GetBitArray(s);

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

{

B\_res[i \* 4 + t] = s\_bits[t];

}

}

B\_res = Permutation(B\_res, P);

return B\_res;

}

private static BitArray Permutation(BitArray data, byte[] map)

{

BitArray res = new BitArray(map.Length);

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

{

res[i] = data[map[i] - 1];

}

return res;

}

private static byte[] GetBytes(BitArray data)

{

byte[] bytes = new byte[data.Length / 8];

data.CopyTo(bytes, 0);

return bytes;

}

private static byte GetByte(BitArray data)

{

if (data.Length <= 8)

{

byte[] bytes = new byte[1];

data.CopyTo(bytes, 0);

return bytes[0];

}

else return 0;

}

private static void GetAB(BitArray R, out byte a, out byte b)

{

BitArray A = new BitArray(2);

BitArray B = new BitArray(4);

A[0] = R[0];

A[1] = R[^1];

for (int i = 1; i < R.Length - 1; i++)

{

B[i - 1] = R[i];

}

a = GetByte(A);

b = GetByte(B);

}

private static BitArray Merge(BitArray L, BitArray R)

{

BitArray res = new BitArray(L.Length \* 2);

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

{

res[i] = L[i];

res[L.Length + i] = R[i];

}

return res;

}

private static BitArray ExtendKey(BitArray key)

{

BitArray res = new BitArray(64);

for (int i = 0; i < 8; i++)

{

int count = 0;

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

{

if (key[i \* 7 + j])

count++;

res[i \* 8 + j] = key[i \* 7 + j];

}

res[i \* 8 + 7] = count % 2 == 0;

}

return res;

}

private static BitArray GenereteNextKey(BitArray key, int shift\_count)

{

GetLR(key, out BitArray L, out BitArray R);

return Merge(LeftShift(L, shift\_count), LeftShift(R, shift\_count));

}

private static BitArray GeneretePrevKey(BitArray key, int shift\_count)

{

GetLR(key, out BitArray L, out BitArray R);

return Merge(RightShift(L, shift\_count), RightShift(R, shift\_count));

}

private static BitArray GetFirstKey(byte[] key)

{

BitArray byte\_key = new BitArray(key);

return Permutation(ExtendKey(byte\_key), CD);

}

private static BitArray GetLastKey(byte[] key)

{

BitArray K = GetFirstKey(key);

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

{

K = GenereteNextKey(K, Shifts[i]);

}

return K;

}

public static byte[] Encrypt(byte[] data, byte[] key)

{

BitArray bitKey = GetFirstKey(key);

byte[] res = new byte[data.Length];

for (int i = 0; i < data.Length; i += 8)

{

byte[] block = new byte[8];

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

{

if (j >= data.Length)

break;

block[j - i] = data[j];

}

BitArray bit\_block = new BitArray(block);

bit\_block = Permutation(bit\_block, IP);

GetLR(bit\_block, out BitArray L, out BitArray R);

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

{

BitArray K = GenereteNextKey(bitKey, Shifts[k]);

BitArray temp\_R = new BitArray(R);

R = L.Xor(F(R, Permutation(K, Choice\_key)));

L = temp\_R;

bitKey = K;

}

BitArray kek = Merge(L, R);

BitArray bit\_res = Permutation(kek, IP\_reverce);

byte[] res\_block = GetBytes(bit\_res);

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

{

res[j] = res\_block[j - i];

}

last\_key = bitKey;

}

return res;

}

public static byte[] Decrypt(byte[] data, byte[] key)

{

BitArray bitKey = GetLastKey(key);

byte[] res = new byte[data.Length];

for (int i = 0; i < data.Length; i += 8)

{

byte[] block = new byte[8];

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

{

block[j - i] = data[j];

}

BitArray bit\_block = new BitArray(block);

bit\_block = Permutation(bit\_block, IP);

GetLR(bit\_block, out BitArray L, out BitArray R);

BitArray K = bitKey;

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

{

BitArray temp\_L = new BitArray(L);

L = R.Xor(F(L, Permutation(K, Choice\_key)));

R = temp\_L;

K = GeneretePrevKey(bitKey, Shifts[Shifts.Length - 1 - k]);

bitKey = K;

}

BitArray kek = Merge(L, R);

BitArray bit\_res = Permutation(kek, IP\_reverce);

byte[] res\_block = GetBytes(bit\_res);

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

{

res[j] = res\_block[j - i];

}

}

return res;

}

}

}

Helper.cs

using System.Collections.Generic;

using System.Text;

namespace Lab2\_Client

{

class Helper

{

public static byte[] ExtendData(byte[] data)

{

int diff = 8 - (data.Length % 8);

byte[] res = new byte[data.Length + diff];

data.CopyTo(res, 0);

return res;

}

public static byte[] RecoverData(List<byte> data)

{

int i = data.Count - 1;

while (data[i] == 0)

{

data.RemoveRange(i, data.Count - i);

i--;

}

return data.ToArray();

}

public static byte[] ExtendKey(string data)

{

StringBuilder stringBuilder = new StringBuilder(data);

if (data.Length > 7)

{

return ToByteArray(stringBuilder.Remove(7, data.Length - 7).ToString());

}

int diff = 7 - data.Length;

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

{

stringBuilder.Append(data[i % data.Length]);

}

return ToByteArray(stringBuilder.ToString());

}

public static string ToString(byte[] source)

{

return Encoding.UTF8.GetString(source);

}

public static byte[] ToByteArray(string str)

{

return Encoding.UTF8.GetBytes(str);

}

}

}

Message.cs

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

namespace Lab2\_Client

{

[Serializable]

class Message

{

public Message(MessageType messageType = 0)

{

Type = messageType;

Data = new List<List<byte>>();

}

public Message() { }

public MessageType Type { get; set; }

public List<List<byte>> Data { get; set; }

public void Send(IPEndPoint IP)

{

UdpClient sender = new UdpClient();

try

{

byte[] dgram = Serilazer<Message>.Serilize(this);

sender.Send(dgram, dgram.Length, IP);

}

finally

{

sender.Close();

}

}

}

}

MessageType.cs

namespace Lab2\_Client

{

enum MessageType

{

C\_To\_AS,

AS\_To\_C,

С\_To\_TGS,

TGS\_To\_C,

C\_To\_SS,

SS\_To\_C,

TicketNotValid,

AccessDenied

}

}

Serilazer.cs

using System.Text;

using System.Text.Json;

namespace Lab2\_Client

{

class Serilazer<T>

{

public static byte[] Serilize(T source)

{

var json = JsonSerializer.Serialize<T>(source);

return Encoding.UTF8.GetBytes(json);

}

public static T Deserilaze(byte[] source)

{

var json = Encoding.UTF8.GetString(source);

return JsonSerializer.Deserialize<T>(json);

}

}

}

TicketGranting.cs

using System;

namespace Lab2\_Client

{

[Serializable]

class TicketGranting

{

public TicketGranting() { }

public string ClientIdentity { get; set; }

public string ServiceIdentity { get; set; }

public DateTime IssuingTime { get; set; }

public long Duration { get; set; }

public string Key { get; set; }

}

}

TimeMark.cs

using System;

namespace Lab2\_Client

{

class TimeMark

{

public string C { get; set; }

public DateTime T { get; set; }

}

}

Program.cs

using System;

using System.Windows.Forms;

namespace Lab2\_Client

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

1. **Вывод**

В данной лабораторной работе были изучены алгоритм шифрования данных DES и протокол Kerberos и создано приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую алгоритм DES.