Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Информационные сети. Основы безопасности»

|  |  |
| --- | --- |
|  |  |

**ОТЧЕТ**

к лабораторной работе № 2

на тему «Идентификация и аутентификация пользователей. Протокол Kerberos»

Выполнил             Н. В. Климкович

Проверил                           Е. А. Лещенко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_heading=h.30j0zll)

[2 Краткие теоретические сведения 4](#_heading=h.1fob9te)

[3 Результаты выполнения лабораторной работы 13](#_heading=h.tyjcwt)

[Выводы](#_heading=h.3dy6vkm) 14

[Список использованных источников 15](#_heading=h.1t3h5sf)

[Приложение А (обязательное) Листинг исходного кода 16](#_heading=h.4d34og8)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является создание приложения, реализующего протокол распределения ключей *Kerberos*, включая процедуру, реализующую алгоритм *DES*. В интерфейсе приложения должны быть наглядно представлены исходные данные, данные, передаваемые по сети каждой из сторон, проверки, выполняемые каждый из участников.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

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

Протокол *Kerberos* достаточно гибкий и имеет возможности тонкой настройки под конкретные применения, а также существует в нескольких версиях. В ходе данной лабораторной работы будет рассмотрен упрощенный механизм аутентификации, реализованные с помощью протокола *Kerberos* версии 5. Схема протокола *Kerberos* представлена на рисунке 2.1.

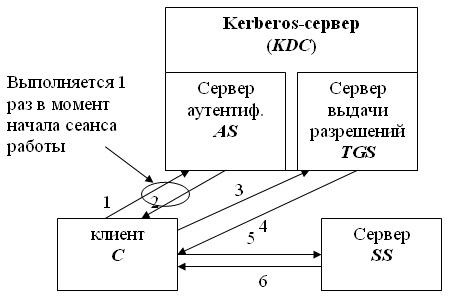


Рисунок 2.1 – Схема протокола Kerberos

При использовании протокола *Kerberos* нельзя напрямую получить доступ к какому-либо целевому серверу. Чтобы запустить 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 определенному рабочему процессу. К этапам этого процесса относятся:

1 Шаг первый: клиент *С* посылает серверу аутентификации *AS*   
свой идентификатор *с*.

2 Шаг второй: сервер аутентификации *AS*, проверив, что клиент *С* имеется в его базе, возвращает ему билет для доступа к серверу выдачи разрешений и ключ для взаимодействия с сервером выдачи разрешений *TGS*. Вся посылка зашифрована на ключе клиента *С*. Таким образом,   
даже если на первом шаге взаимодействия идентификатор *с* послал не клиент *С*, а нарушитель *Х*, то полученную от *AS* посылку *Х* расшифровать не сможет. Получить доступ к содержимому билета *TGT* не может не только нарушитель, но и клиент *С*, так как билет зашифрован на ключе, который распределили между собой сервер аутентификации и сервер выдачи разрешений.

3 Шаг третий: клиент *С* обращается к серверу выдачи разрешений *TGS*. Он пересылает полученный от *AS* билет, зашифрованный на ключе *Kas\_tgs*,   
и аутентификационный блок, содержащий идентификатор *с* 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 *С*   
и сервером *TGS*. С помощью этого ключа расшифровывается аутентификационный блок. Если метка в блоке совпадает с меткой в билете, это доказывается, 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. Если проверка проходит, и действующая   
в системе политика позволяется клиенту *С* обращаться к клиенту *SS*,   
тогда выполняется следующий шаг.

4 Шаг 4: сервер выдачи разрешений *TGS* посылает клиенту *С* ключ шифрования и билет, необходимые для доступа к серверу *SS*. Структура билета такая же, как на шаге номер 2.

5 Шаг пятый: клиент *С* посылает билет, полученный от сервера выдачи разрешений, и свой аутентификационный блок *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.

6 Шаг шестой: 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.   
Именно поэтому *SS* берет отметку времени из аутентификационного блока *С*, изменяет ее, увеличивая на единицу, шифрует на ключе *Kc\_ss* и возвращает *С*.

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

В алгоритме *Kerberos* могут применяться различные алгоритмы блочного симметричного шифрования. Для целей лабораторной работы был использован алгоритм *DES*.

Одной из наиболее известных криптографических систем с закрытым ключом является *DES* – *Data* *Encryption* *Standard*.

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. На рисунке 2.2 представлена общая схема *DES*.

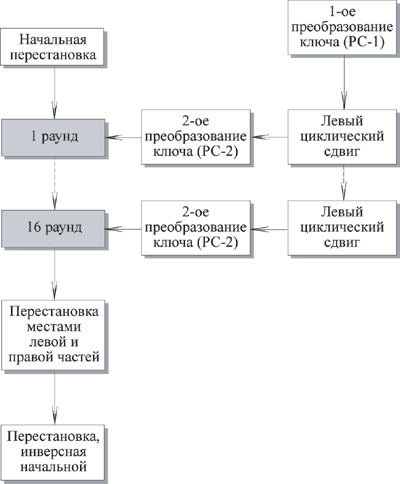


Рисунок 2.2 – Общая схема DES

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.

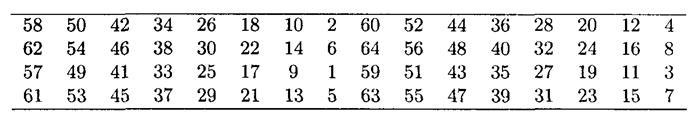
****

Рисунок 2.3 – Начальная перестановка

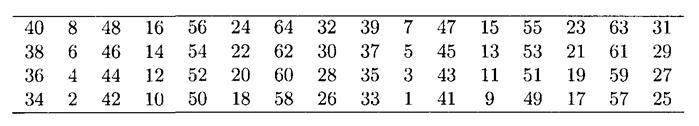
****

Рисунок 2.4 – Заключительная перестановка

Теперь будет рассмотрена последовательность преобразований, используемую в каждом раунде.

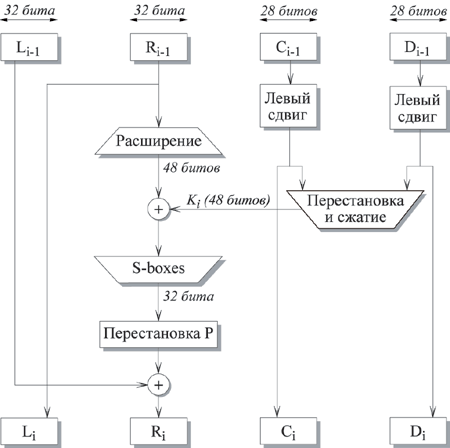


Рисунок 2.5 – *I*-ый раунд *DES*

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:

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*. *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. Например, если часть входного сообщения «*efgh ijkl mnop*», то в результате расширения получается сообщение «*defghi hijklm lmnopq*».

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. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

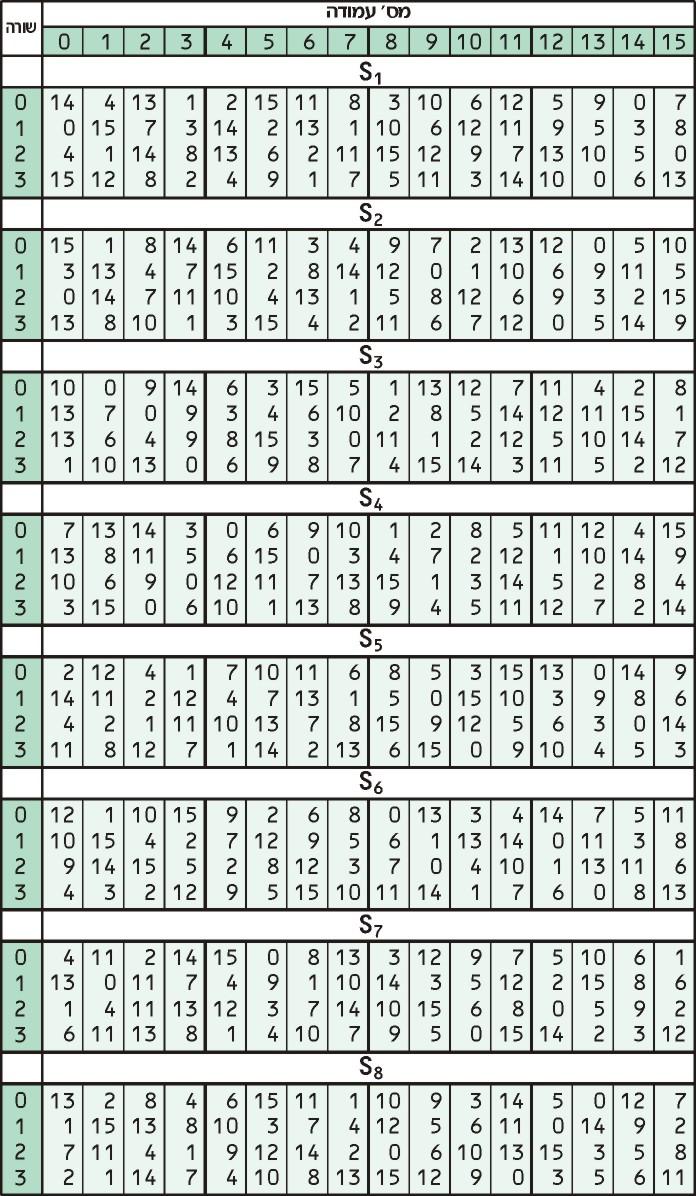


Рисунок 2.6 – *S-boxes*

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*.

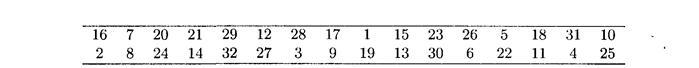


Рисунок 2.7 – Перестановка с помощью *Р*-блоков

Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма (если используется 64-битный ключ, то убираются биты 64, 56, 48, 40, 32, 16, 8), вначале выполняется перестановка в соответствии с таблицей *Permuted Choice* 1 (*РС*-1).

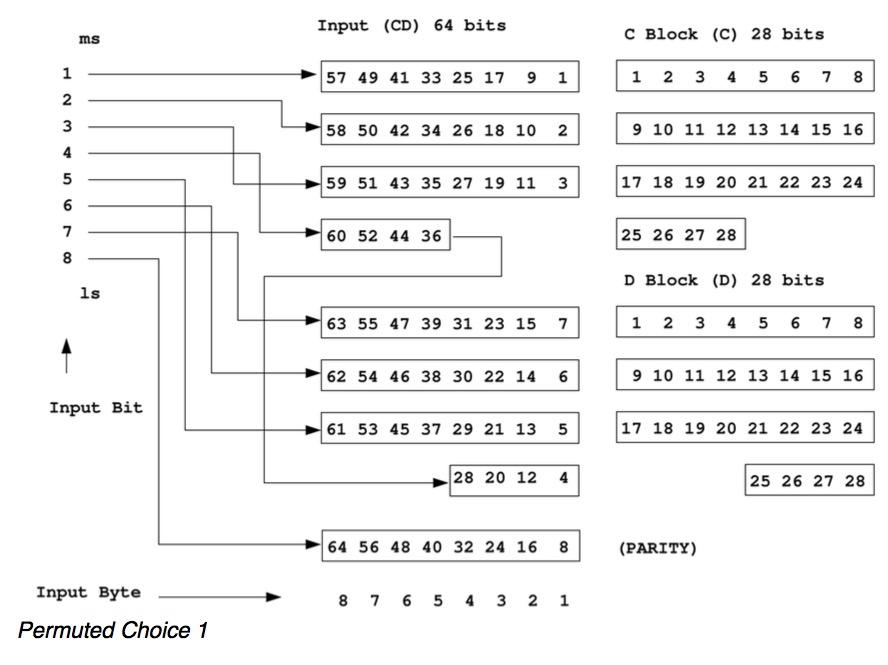


Рисунок 2.8 – Схема *Permuted Choice*

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 цикла.

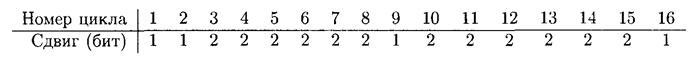


Рисунок 2.9 – Сдвиг ключа в зависимости от номера цикла

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*).

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.

# **3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ**

В ходе выполнения лабораторной было создано приложение, реализующее протокол распределения ключей *Kerberos*, включая процедуру, реализующую алгоритм *DES*. В ходе работы программа выводит исходные данные и данные, передаваемые по сети. Результат работы программы представлен на рисунке 3.1.

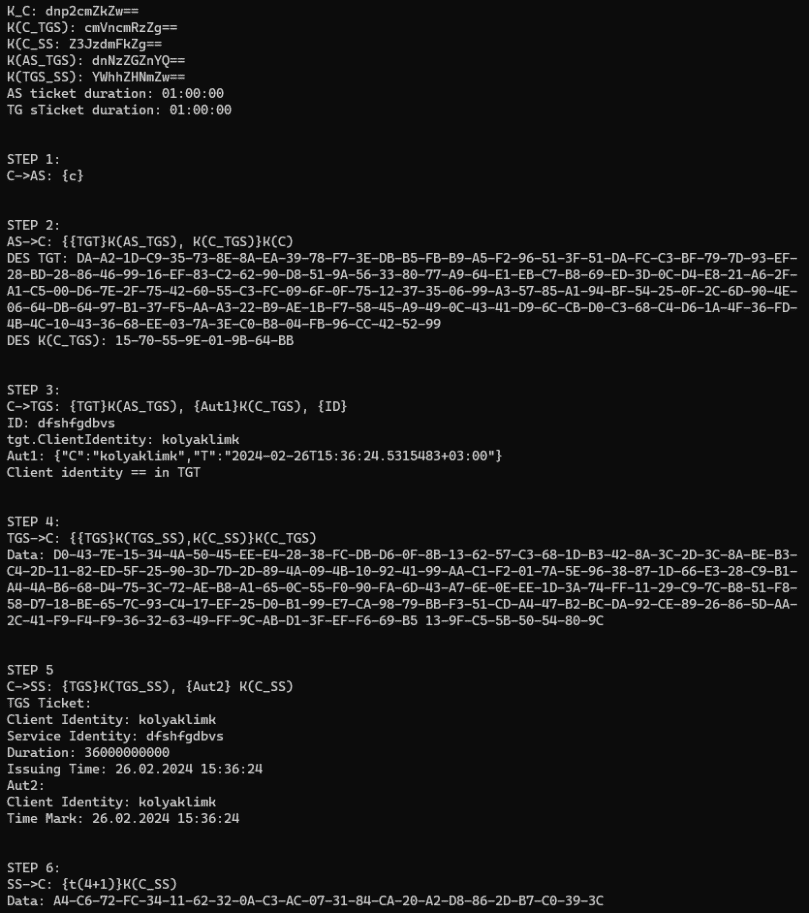


Рисунок 3.1 – Результат работы программы

Результат работы программы на стороне клиента представлен на рисунке 3.2.

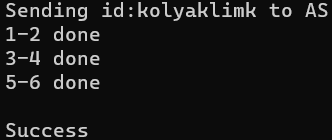


Рисунок 3.2 – Результат тестирования исполняемого файла

Таким образом, в ходе данной лабораторной работы было создано приложение, реализующее протокол с использованием алгоритма *DES*.

# **ВЫВОДЫ**

В ходе лабораторной работы было создано приложение, реализующее протокол распределения ключей Kerberos, включая алгоритм DES.   
В интерфейсе приложения наглядно представлены исходные данные, данные, передаваемые по сети каждой из сторон, и проверки, выполняемые каждым   
из участников.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Что такое Kerberos? [Электронный ресурс]. – Режим доступа: https://www.keepersecurity.com/ru\_RU/resources/glossary/what-is-kerberos/. – Дата доступа: 23.02.2024.

[2] Идентификация и аутентификация. Протокол *Kerberos* [Электронный ресурс]. – Режим доступа: https://intuit.ru/studies/courses/531/387/lecture/8998/. – Дата доступа: 23.02.2024.

[3] Стандарт шифрования *DES* [Электронный ресурс]. – Режим доступа:   
https://kaf401.rloc.ru/Criptfiles/DES.htm/. – Дата доступа: 23.02.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

Листинг 1 – Программный код класса *Additionally*

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab2

{

public class Additionally

{

public static byte[] ExtendKey(string data)

{

StringBuilder stringBuilder = new StringBuilder(data);

if (data.Length > 7)

{

return Encoding.UTF8.GetBytes(stringBuilder.Remove(7, data.Length - 7).ToString());

}

for (int i = 0; i < 7 - data.Length; i++)

{

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

}

return Encoding.UTF8.GetBytes(stringBuilder.ToString());

}

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

{

if (t2 < t1 + new TimeSpan(duration)) return true;

return false;

}

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();

}

}

}

Листинг 2 – Программный код класса *AS*

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Text;

namespace Lab2

{

class AS

{

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

public AS()

{

Users.Add("kolyaklimk");

}

public void Listen()

{

UdpClient receiver = new UdpClient(Settings.AS\_port);

IPEndPoint remoteIP = null;

try

{

while (true)

{

byte[] data = receiver.Receive(ref remoteIP);

remoteIP.Port = Settings.C\_port;

TransportData responseMessage = new TransportData();

Console.WriteLine("\n\nSTEP 1: ");

Console.WriteLine("C->AS: {c}");

var trdata = UTF8Serialiser<TransportData>.Deserialise(data);

if (trdata.Type == EnumSteps.\_1)

{

Console.WriteLine("\n\nSTEP 2: ");

Console.WriteLine("AS->C: {{TGT}K(AS\_TGS), K(C\_TGS)}K(C)");

var id = Encoding.UTF8.GetString(trdata.Data[0].ToArray());

if (Users.Contains(id))

{

responseMessage.Type = EnumSteps.\_2;

TgsClass ticket = new TgsClass()

{

ClientIdentity = id,

Duration = Settings.ASTicketDuration.Ticks,

IssuingTime = DateTime.Now,

ServiceIdentity = Settings.tgs,

Key = Encoding.UTF8.GetString(Settings.K\_C\_TGS)

};

var ticket\_bytes = Additionally.ExtendData(UTF8Serialiser<TgsClass>.Serialise(ticket));

var k\_c\_tgs\_bytes = Additionally.ExtendData(Settings.K\_C\_TGS);

var tb\_enc = DES.Encrypt(ticket\_bytes, Settings.K\_AS\_TGS);

tb\_enc = DES.Encrypt(tb\_enc, Settings.K\_C);

var k\_c\_tgs\_enc = DES.Encrypt(k\_c\_tgs\_bytes, Settings.K\_C);

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

responseMessage.Data.Add(new List<byte>(k\_c\_tgs\_enc));

Console.WriteLine($"DES TGT: {BitConverter.ToString(tb\_enc)}");

Console.WriteLine($"DES K(C\_TGS): {BitConverter.ToString(k\_c\_tgs\_enc)}");

}

else

{

responseMessage.Type = EnumSteps.NoAccess;

Console.WriteLine("Access denied on the authentication server");

}

responseMessage.Send(remoteIP);

}

}

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

}

}

}

}

Листинг 3 – Программный код класса *DES*

using System.Collections;

using System.Text;

namespace Lab2

{

public class DES

{

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[] 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[] Shifts = new byte[]

{

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

};

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[] 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\_reverse = 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[] 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

}

};

private static readonly byte Rounds = 16;

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 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 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 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 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 BitArray getBitArray(byte data)

{

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

return new BitArray(bytes);

}

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[R.Length - 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 ? true : false;

}

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 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 GetLastKey(byte[] key)

{

BitArray K = GetFirstKey(key);

for (int i = 0; i < Rounds; 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 < Rounds; 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\_reverse);

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

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

{

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

}

}

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 < Rounds; 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\_reverse);

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

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

{

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

}

}

return res;

}

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

{

var data\_bytes = Encoding.ASCII.GetBytes(data);

var key\_bytes = Encoding.ASCII.GetBytes(key);

return Encrypt(data\_bytes, key\_bytes);

}

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

{

var data\_bytes = Encoding.ASCII.GetBytes(data);

var key\_bytes = Encoding.ASCII.GetBytes(key);

return Decrypt(data\_bytes, key\_bytes);

}

}

}

Листинг 4 – Программный код класса *EnumSteps*

namespace Lab2

{

public enum EnumSteps

{

\_1,

\_2,

\_3,

\_4,

\_5,

\_6,

NoValid,

NoAccess

}

}

Листинг 5 – Программный код класса *Program*

using System;

using System.Threading.Tasks;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

AS AS = new AS();

TGS TGS = new TGS();

Server SS = new Server();

try

{

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

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

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

Console.WriteLine($"K\_C: {Convert.ToBase64String(Settings.K\_C)}");

Console.WriteLine($"K(C\_TGS): {Convert.ToBase64String(Settings.K\_C\_TGS)}");

Console.WriteLine($"K(C\_SS: {Convert.ToBase64String(Settings.K\_C\_SS)}");

Console.WriteLine($"K(AS\_TGS): {Convert.ToBase64String(Settings.K\_AS\_TGS)}");

Console.WriteLine($"K(TGS\_SS): {Convert.ToBase64String(Settings.K\_TGS\_SS)}");

Console.WriteLine($"AS ticket duration: {Settings.ASTicketDuration}");

Console.WriteLine($"TG sTicket duration: {Settings.TGSTicketDuration}");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

Console.ReadLine();

}

}

}

Листинг 6 – Программный код класса *Server*

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

namespace Lab2

{

class Server

{

public void Listen()

{

UdpClient receiver = new UdpClient(Settings.SS\_port);

IPEndPoint remoteIP = null;

try

{

while (true)

{

byte[] data = receiver.Receive(ref remoteIP);

remoteIP.Port = Settings.C\_port;

var message = UTF8Serialiser<TransportData>.Deserialise(data);

if (message.Type == EnumSteps.\_5)

{

Console.WriteLine("\n\nSTEP 5");

Console.WriteLine("C->SS: {TGS}K(TGS\_SS), {Aut2} K(C\_SS)");

var tgs\_bytes = Additionally.RecoverData(

new List<byte>(DES.Decrypt(message.Data[0].ToArray(), Settings.K\_TGS\_SS)));

var tgs = UTF8Serialiser<TgsClass>.Deserialise(tgs\_bytes);

Console.WriteLine("TGS Ticket:");

Console.WriteLine($"Client Identity: {tgs.ClientIdentity}");

Console.WriteLine($"Service Identity: {tgs.ServiceIdentity}");

Console.WriteLine($"Duration: {tgs.Duration}");

Console.WriteLine($"Issuing Time: {tgs.IssuingTime}");

var aut2\_bytes = Additionally.RecoverData(

new List<byte>(DES.Decrypt(message.Data[1].ToArray(), Settings.K\_C\_SS)));

var aut2 = UTF8Serialiser<TimeMark>.Deserialise(aut2\_bytes);

Console.WriteLine("Aut2:");

Console.WriteLine($"Client Identity: {aut2.C}");

Console.WriteLine($"Time Mark: {aut2.T}");

TransportData ReMessage = new TransportData();

if (Additionally.TimeComparison(tgs.IssuingTime, aut2.T, tgs.Duration))

{

Console.WriteLine("\n\nSTEP 6:");

Console.WriteLine("SS->C: {t(4+1)}K(C\_SS)");

ReMessage.Type = EnumSteps.\_6;

DateTime reTime = aut2.T;

var time\_bytes = UTF8Serialiser<long>.Serialise(aut2.T.Ticks + 1);

var bytes = DES.Encrypt(Additionally.ExtendData(time\_bytes), Settings.K\_C\_SS);

Console.WriteLine($"Data: {BitConverter.ToString(bytes)}");

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

}

else

{

ReMessage.Type = EnumSteps.NoValid;

Console.WriteLine("TicketNotValid in SS;");

}

ReMessage.Send(remoteIP);

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

}

}

}

Листинг 7 – Программный код класса *Settings*

using System;

namespace Lab2

{

public static class Settings

{

public static readonly byte[] K\_C = Additionally.ExtendKey("vzvrfdgvsdfasedg");

public static readonly byte[] K\_C\_TGS = Additionally.ExtendKey("regrdsfgzsvres");

public static readonly byte[] K\_C\_SS = Additionally.ExtendKey("grsvadfgeevcvx");

public static readonly byte[] K\_AS\_TGS = Additionally.ExtendKey("vssdfgagvfds");

public static readonly byte[] K\_TGS\_SS = Additionally.ExtendKey("ahadsfgfdasgarer");

public const string tgs = "nbdfsfvadh";

public const string ID\_SS = "dfshfgdbvs";

public static readonly TimeSpan ASTicketDuration = new TimeSpan(1, 0, 0);

public static readonly TimeSpan TGSTicketDuration = new TimeSpan(1, 0, 0);

public const int C\_port = 8111;

public const int AS\_port = 8222;

public const int SS\_port = 8333;

public const int TGS\_port = 8444;

}

}

Листинг 8 – Программный код класса *TGS*

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Text;

namespace Lab2

{

class TGS

{

public void Listen()

{

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

IPEndPoint remoteIP = null;

try

{

while (true)

{

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

remoteIP.Port = Settings.C\_port;

Console.WriteLine("\n\nSTEP 3:");

Console.WriteLine("C->TGS: {TGT}K(AS\_TGS), {Aut1}K(C\_TGS), {ID}");

var message = UTF8Serialiser<TransportData>.Deserialise(data);

if (message.Type == EnumSteps.\_3)

{

var tgt\_json = Additionally.RecoverData(new List<byte>(DES.Decrypt(message.Data[0].ToArray(), Settings.K\_AS\_TGS)));

var tgt = UTF8Serialiser<TgsClass>.Deserialise(tgt\_json);

var aut1\_json = Additionally.RecoverData(new List<byte>(DES.Decrypt(message.Data[1].ToArray(), Settings.K\_C\_TGS)));

var a = Encoding.UTF8.GetString(aut1\_json);

var aut1 = UTF8Serialiser<TimeMark>.Deserialise(aut1\_json);

var ID = Encoding.UTF8.GetString(message.Data[2].ToArray());

Console.WriteLine($"ID: {ID}");

Console.WriteLine($"tgt.ClientIdentity: {tgt.ClientIdentity}");

Console.WriteLine($"Aut1: {a}");

if (tgt.ClientIdentity == aut1.C)

{

Console.WriteLine("Client identity == in TGT");

}

else

{

Console.WriteLine("Client identity != in TGT");

}

TransportData ReMessage = new TransportData();

if (Additionally.TimeComparison(tgt.IssuingTime, aut1.T, tgt.Duration))

{

Console.WriteLine("\n\nSTEP 4:");

Console.WriteLine("TGS->C: {{TGS}K(TGS\_SS),K(C\_SS)}K(C\_TGS)");

ReMessage.Type = EnumSteps.\_4;

var TGS = new TgsClass()

{

ClientIdentity = aut1.C,

ServiceIdentity = ID,

Duration = Settings.TGSTicketDuration.Ticks,

IssuingTime = DateTime.Now,

Key = Encoding.UTF8.GetString(Settings.K\_C\_SS)

};

var ticket\_bytes = Additionally.ExtendData(UTF8Serialiser<TgsClass>.Serialise(TGS));

var k\_c\_ss\_bytes = Additionally.ExtendData(Settings.K\_C\_SS);

var tgs\_enc = DES.Encrypt(ticket\_bytes, Settings.K\_TGS\_SS);

tgs\_enc = DES.Encrypt(tgs\_enc, Settings.K\_C\_TGS);

var k\_c\_ss\_enc = DES.Encrypt(k\_c\_ss\_bytes, Settings.K\_C\_TGS);

Console.WriteLine($"Data: {BitConverter.ToString(tgs\_enc)} {BitConverter.ToString(k\_c\_ss\_enc)}");

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

ReMessage.Data.Add(new List<byte>(k\_c\_ss\_enc));

}

else

{

ReMessage.Type = EnumSteps.NoValid;

Console.WriteLine("TicketNotValid in TGS");

}

ReMessage.Send(remoteIP);

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

}

}

}

Листинг 9 – Программный код класса *TgsClass*

using System;

namespace Lab2

{

[Serializable]

public class TgsClass

{

public TgsClass() { }

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; }

}

public class TimeMark

{

public string C { get; set; }

public DateTime T { get; set; }

}

}

Листинг 10 – Программный код класса *TransportData*

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Text;

namespace Lab2

{

public class TransportData

{

public TransportData(EnumSteps messageType = 0)

{

Type = messageType;

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

}

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

public EnumSteps Type { get; set; }

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

public void Send(IPEndPoint remoteIP)

{

UdpClient sender = new UdpClient();

try

{

byte[] dgram = UTF8Serialiser<TransportData>.Serialise(this);

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

}

finally

{

sender.Close();

}

}

public override string ToString()

{

StringBuilder sb = new StringBuilder();

sb.AppendLine($"Message Type: {Type}");

if (Data != null && Data.Count > 0)

{

sb.AppendLine("Data:");

foreach (var dataItem in Data)

{

sb.AppendLine(BitConverter.ToString(dataItem.ToArray()));

}

}

else

{

sb.AppendLine("No data present.");

}

return sb.ToString();

}

}

}

Листинг 11 – Программный код класса *UTF8Serialiser*

using System.Text;

using System.Text.Json;

namespace Lab2

{

public class UTF8Serialiser<T>

{

public static byte[] Serialise(T source)

{

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

return Encoding.UTF8.GetBytes(json);

}

public static T Deserialise(byte[] source)

{

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

return JsonSerializer.Deserialize<T>(json);

}

}

}

Листинг 11 – Программный код класса *Client*

using Lab2;

using System.Net;

using System.Net.Sockets;

using System.Text;

namespace Lab2\_Client

{

class Client

{

public string Login { get; set; }

private byte[] TicketGrantingTicket { get; set; }

private byte[] TicketGrantingService { get; set; }

private byte[] K\_C\_TGS { get; set; }

private byte[] K\_C\_SS { get; set; }

DateTime T4 { get; set; }

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

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

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

public void LogOn(string login)

{

Login = login;

Console.WriteLine($"Sending id:{login} to AS");

TransportData trdata = new TransportData(EnumSteps.\_1);

trdata.Data.Add(new List<byte>(Encoding.UTF8.GetBytes(login)));

trdata.Send(ASEndPoint);

}

public void Connection()

{

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

IPEndPoint remoteIP = null;

try

{

while (true)

{

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

TransportData trdata = UTF8Serialiser<TransportData>.Deserialise(data);

switch (trdata.Type)

{

case EnumSteps.\_2:

{

TicketGrantingTicket = DES

.Decrypt(trdata.Data[0].ToArray(), Settings.K\_C);

K\_C\_TGS = Additionally

.RecoverData(new List<byte>(DES.Decrypt(trdata.Data[1].ToArray(), Settings.K\_C)));

var a = Encoding.UTF8.GetString(K\_C\_TGS);

Console.WriteLine("1-2 done");

trdata = new TransportData(EnumSteps.\_3);

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

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

var Aut1 = Additionally.ExtendData(UTF8Serialiser<TimeMark>.Serialise(mark));

trdata.Data.Add(new List<byte>(DES.Encrypt(Aut1, K\_C\_TGS)));

trdata.Data.Add(new List<byte>(Encoding.UTF8.GetBytes(Settings.ID\_SS)));

trdata.Send(TGSEndPoint);

break;

}

case EnumSteps.\_4:

{

TicketGrantingService = DES

.Decrypt(trdata.Data[0].ToArray(), K\_C\_TGS);

K\_C\_SS = Additionally

.RecoverData(new List<byte>(DES.Decrypt(trdata.Data[1].ToArray(), K\_C\_TGS)));

TransportData msg = new TransportData(EnumSteps.\_5);

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

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

var Aut2 = Additionally.ExtendData(UTF8Serialiser<TimeMark>.Serialise(mark));

T4 = mark.T;

msg.Data.Add(new List<byte>(DES.Encrypt(Aut2, K\_C\_SS)));

msg.Send(SSEndPoint);

Console.WriteLine("3-4 done");

break;

}

case EnumSteps.\_6:

{

var t = DES.Decrypt(trdata.Data[0].ToArray(), K\_C\_SS);

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

var asd = Encoding.UTF8.GetString(checkT\_bytes);

var checkT = UTF8Serialiser<long>.Deserialise(checkT\_bytes);

Console.WriteLine("5-6 done\n");

if (T4.Ticks + 1 == checkT)

{

Console.WriteLine($"Success");

}

break;

}

case EnumSteps.NoValid:

Console.WriteLine("Ticket is not valid");

break;

case EnumSteps.NoAccess:

Console.WriteLine("No access");

break;

default:

Console.WriteLine("Invalid type of message");

break;

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}