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 К. А. Тимофеев

Проверил Е. А. Лещенко

Минск 2024

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Kerberos – это протокол аутентификации, который обеспечивает безопасное взаимодействие между клиентом и сервером путем проверки подлинности обеих сторон перед установлением соединения. Он использует криптографические ключи для обеспечения безопасности передачи данных через незащищенные сети.

Цель данной лабораторной работы заключается в изучении теории работы протокола Kerberos и алгоритма DES, а также в разработке программы, которая реализует протокол распределения ключей Kerberos и использует алгоритм DES для шифрования данных.

# 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Kerberos – это протокол аутентификации компьютерной сети, предназначенный для упрощения и безопасности аутентификации.

Основная идея Kerberos вращается вокруг использования локальной формы личной идентификации, называемой билетами, которые выдаются сервером аутентификации. Каждый билет принадлежит определенным областям, которые определяют, к каким услугам билет предоставляет доступ. Эти билеты зашифрованы, и для их использования требуется несколько уровней расшифровки. Эта система билетов гарантирует, что конфиденциальная информация, такая как пароли, никогда не будет отправлена по сети.

Kerberos получил широкое распространение с момента его создания в Массачусетском технологическом институте в 1980-х годах. Теперь он встроен в бесчисленное множество реализаций, связанных с безопасностью, в Интернете, причем почти все компании ежедневно взаимодействуют хотя бы с одной системой Kerberos.

Наиболее известное использование Kerberos связано с Microsoft Active Directory, службой каталогов по умолчанию, включенной в Windows 2000 и более поздние версии для управления доменами и аутентификации пользователей.

Другие известные применения – Apple, NASA, Google, Министерство обороны США и университеты по всей территории США.

Аутентификация Kerberos работает в 4 этапа: вход пользователя, аутентификация клиента в AS, аутентификация клиента со службой, запрос услуги.

На этапе входа пользователя происходит взаимодействие между Пользователем и Клиентом. Пользователь вводит свое имя пользователя и пароль в клиент. Затем клиент преобразует этот пароль в ключ шифрования, хранящийся локально. Если все выполнено правильно, клиент может начать аутентификацию с помощью AS.

На этапе аутентификации клиента в AS клиент и сервер аутентификации соединяются для аутентификации имени пользователя и обеспечения его принадлежности к системе. Затем AS проверяет, что имя пользователя уже задокументировано в системе. В этом случае Клиент и AS обмениваются зашифрованными проверочными сообщениями для проверки друг друга. В конце концов оба аутентифицируются, соединение устанавливается, и клиент может перейти к аутентификации с помощью службы.

На этапе аутентификации клиента/службы клиент и сервер должны аутентифицировать друг друга, соблюдая практику взаимной аутентификации. Клиент и сервер обмениваются зашифрованными проверочными сообщениями, аналогично предыдущему этапу. Если все они пройдены, клиент и служба аутентифицируются, и клиенту разрешено запрашивать их услуги.

На этапе запроса клиента/услуги клиент может запросить именованную услугу у сервисного сервера. Затем сервер службы проверяет наличие запрошенной службы. Если да, сервер службы предоставляет услугу клиенту. Поскольку клиент прошел проверку подлинности на всех этапах этого процесса, он может продолжать использовать службу до истечения срока действия его разрешений.

# 2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

В результате разработки программы было создано консольное приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES.

На рисунке 2.1 представлена схема работы протокола Kerberos.

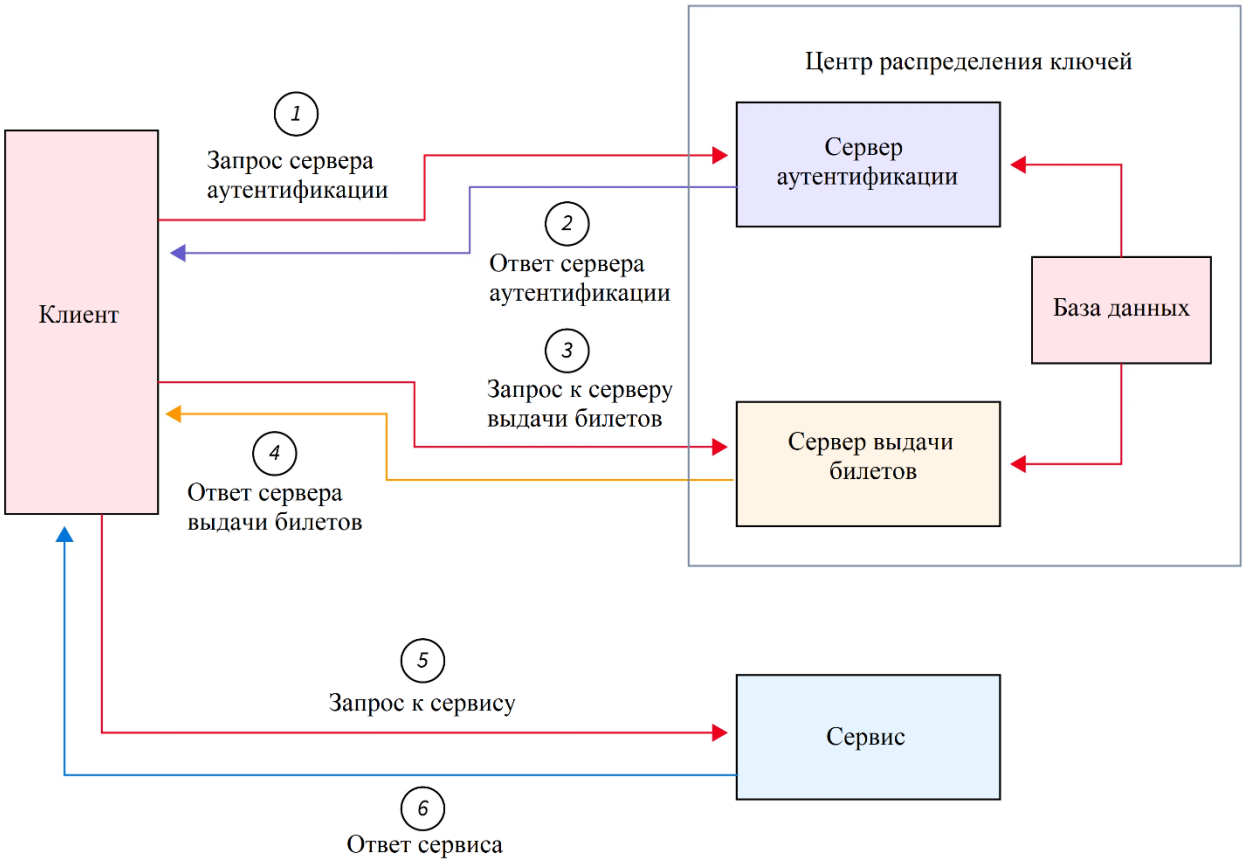


Рисунок 2.1 – Схема работы протокола Kerberos

На рисунке 2.2 представлена блок-схема алгоритма шифрования DES.


Рисунок 2.2 – Блок-схема алгоритма DES

Результат выполнения программы изображён на рисунке 2.3.

```
Step1 - Client send his identifier

Step2 - Client get
 encrypted tgt: 05BD497222E8B02C9EACDE62C9952C1C1714FA0701F18AF8DBDB8311B27D02530DFC288F78F2023A1F2924E845A90AF05AA606DBA5A6527BC18614BF4C3A8E5733BEBDFF5176FB12 and
 tgsClientKey: 60FC17735C6DE913

Step3 - Client send
 serverId: server1,
 encryptedTgt: 05BD497222E8B02C9EACDE62C9952C1C1714FA0701F18AF8DBDB8311B27D02530DFC288F78F2023A1F2924E845A90AF05AA606DBA5A6527BC18614BF4C3A8E5733BEBDFF5176FB12 and
 tgsClientKey: 60FC17735C6DE913

Step4 - Client get
 encryptedTgs: 89CBF5E35B8C6C10517D7E7C770F41BB50F7A99F351C5AD8E4971A6BC755E14D0E2D83D2FF72B8FDF9BA9BAE55889B04EC0E650021B0F68D8ABCD92695157544849 43B484DA00D76 and
 serverKey: 5CBB44E7F8856B3C

Step5 - Client send
 encryptedTgs: 89CBF5E35B8C6C10517D7E7C770F41BB50F7A99F351C5AD8E4971A6BC755E14D0E2D83D2FF72B8FDF9BA9BAE55889B04EC0E650021B0F68D8ABCD92695157544849 43B484DA00D76 and
 clientServerKey: 5CBB44E7F8856B3C

Step6 - Client connected with server

Client get data from server
Client get data from server

Step1 - Client send his identifier

Step2 - Client get
 encrypted tgt: 05BD497222E8B02C9EACDE62C9952C1C1714FA0701F18AF8DBDB8311B27D02536DFD6CC9694DA0CD1F2924E845A90AF0DB32F2C8FC7E914293ACBC5C159DC91F1B05AC67FEABAED1 and
 tgsClientKey: DCB4F471B318D456

Step3 - Client send
 serverId: server1,
 encryptedTgt: 05BD497222E8B02C9EACDE62C9952C1C1714FA0701F18AF8DBDB8311B27D02536DFD6CC9694DA0CD1F2924E845A90AF0DB32F2C8FC7E914293ACBC5C159DC91F1B05AC67FEABAED1 and
 tgsClientKey: DCB4F471B318D456

Step4 - Client get
 encryptedTgs: 89CBF5E35B8C6C10517D7E7C770F41BB50F7A99F351C5AD8E4971A6BC755E14D0E2D83D2FF72B8FDDE588FD73ADCFA993378A6CD02AEF718D79F6282CAC3AAB9E8047B6E0849E636 and
 serverKey: BB8BE1C2E39A1D78

Step5 - Client send
 encryptedTgs: 89CBF5E35B8C6C10517D7E7C770F41BB50F7A99F351C5AD8E4971A6BC755E14D0E2D83D2FF72B8FDDE588FD73ADCFA993378A6CD02AEF718D79F6282CAC3AAB9E8047B6E0849E636 and
 clientServerKey: BB8BE1C2E39A1D78

Step6 - Client connected with server

Client get data from server
```

Рисунок 2.3 – Результат программы

# ПРИЛОЖЕНИЕ А
## (обязательное)

## Исходный код программы

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Numerics;
using System.Runtime.InteropServices;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace Kerberos;

public static class Des
{
    private static readonly int[] InitialPermutation =
    [
        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,
        56, 48, 40, 32, 24, 16, 8,  0, 58, 50, 42, 34, 26, 18, 10, 2,
        60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6
    ];

    private static readonly int[]  InversePermutation =
    [
        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, 32, 0, 40, 8,  48, 16, 56, 24,
    ];

    private static readonly int[][] SBoxes =
    [
        [
            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 int[] PermutedChoice1 =   // Перестановка ключа
    [
        56, 48, 40, 32, 24, 16, 8,  0,  57, 49, 41, 33, 25, 17,
        9,  1,  58, 50, 42, 34, 26, 18, 10, 2,  59, 51, 43, 35,
        62, 54, 46, 38, 30, 22, 14, 6,  61, 53, 45, 37, 29, 21,
        13, 5,  60, 52, 44, 36, 28, 20, 12, 4,  27, 19, 11, 3
    ];



    private static readonly int[] Rotates =
    [
        1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
    ];


    private static readonly int[] PermutedChoice2 =   // Перестановка со
сжатием
    [
        13, 16, 10, 23, 0,  4,  2,  27, 14, 5,  20, 9,
        22, 18, 11, 3,  25, 7,  15, 6,  26, 19, 12, 1,
        40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
        43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31
    ];


    private static readonly int[] Expansion =
    [
        31, 0,  1,  2,  3,  4,  3,  4,  5,  6,  7,  8,
        7,  8,  9,  10, 11, 12, 11, 12, 13, 14, 15, 16,
        15, 16, 17, 18, 19, 20, 19, 20, 21, 22, 23, 24,
        23, 24, 25, 26, 27, 28, 27, 28, 29, 30, 31, 0
    ];
```

```
    private static readonly int[] Permutation =
    [
        15, 6, 19, 20, 28, 11, 27, 16,
        0, 14, 22, 25, 4, 17, 30, 9,
        1, 7, 23, 13, 31, 26, 2, 8,
        18, 12, 29, 5, 21, 10, 3, 24
    ];



    private static IEnumerable<string> SliceMes(string str)
    {
        ArgumentException
            .ThrowIfNullOrWhiteSpace(str, nameof(str));

        int chunkSize = 8;

        for (int i = 0; i < str.Length; i += chunkSize)
        {
            string chunk = str.Substring(i, Math.Min(chunkSize, str.Length -
i));
            chunk = chunk.PadLeft(chunkSize, '\0');

            byte[] bytes = Encoding.UTF8.GetBytes(chunk);

            yield return Convert.ToHexString(bytes);
        }
    }

    private static string ToBin(string hexString)
    {
        return string.Join(string.Empty, hexString.Select(c =>
        {
            return Convert.ToString(Convert.ToInt32(c.ToString(), 16),
2).PadLeft(4, '0');
        }));
    }


    private static string Permute(string block, IEnumerable<int> box)
    {
        return string.Join(string.Empty, box.Select(i => block[i]));
    }

    private static string RotateLeft(int block, int i)
    {
        return Convert.ToString((block << i & 0x0fffffff) | (block >> (28 -
i)), 2).PadLeft(28, '0');
    }


    private static IEnumerable<string> KeyGen(string block1, string block2)
    {
        foreach (var i in Rotates)   // раунды
        {
            block1 = RotateLeft(Convert.ToInt32(block1, 2), i);
            block2 = RotateLeft(Convert.ToInt32(block2, 2), i);

            yield return Permute(block1 + block2, PermutedChoice2);
        }
    }
```

```
    private static string Xor(string arg_1, string arg_2)
    {
        return string.Join(string.Empty, arg_1.Zip(arg_2).Select((lr) =>
(lr.First ^ lr.Second).ToString()));
    }

    private static string F(string block, string key)
    {
        List<string> final = [];

        var l = Xor(Permute(block, Expansion), key);

        List<string> list = [];

        for (int i = 0; i < l.Length; i += 6)
        {
            list.Add(l.Substring(i, Math.Min(6, l.Length - i)));
        }

        foreach (var (i, j) in list.Select((item, index) => (item, index)))
        {
            int[][] tempBox =
            [
                SBoxes[j][0..16],
                SBoxes[j][16..32],
                SBoxes[j][32..48],
                SBoxes[j][48..64]
            ];


final.Add(Convert.ToString(tempBox[Convert.ToInt32(i[0].ToString() + i[^1],
2)][Convert.ToUInt32(i[1..^1], 2)], 2).PadLeft(4, '0'));
        }


        return Permute(string.Join(string.Empty, final), Permutation);
    }



    private static IEnumerable<string> Des_(string block, IEnumerable<string>
keyArray)
    {
        int keyCenter = block.Length / 2;

        string L = block[..keyCenter];
        string R = block[keyCenter..];

        foreach (var i in keyArray)
        {
            (R, L) = (Xor(F(R, i), L), R);
        }

        var l = Permute(R + L, InversePermutation);

        for (int i = 0; i < l.Length; i += 8)
        {
            yield return l.Substring(i, Math.Min(8, l.Length - i));
        }
    }
```

```
    private static IEnumerable<string> Cut(this string @string, int length)
    {
        for (int i = 0; i < @string.Length; i += length)
        {
            yield return @string.Substring(i, Math.Min(length, @string.Length
- i));
        }
    }


    public static string Encrypt(string data, string key)
    {
        StringBuilder ans = new StringBuilder();

        foreach (var i in SliceMes(data))
        {
            string binMess = ToBin(i);
            string binKey = ToBin(key);

            string permutedBlock = Permute(binMess, InitialPermutation);
// Начальная перестановка
            string permutedKey = Permute(binKey, PermutedChoice1);
// Начальная перестановка ключа. В итоге преобразуется в 56 бит.


            int keyCenter = permutedKey.Length / 2;

            string L = permutedKey[..keyCenter];
            string R = permutedKey[keyCenter..];

            List<string> keyList = KeyGen(L, R).ToList();

            var l = Des_(permutedBlock, keyList).Select(i =>
Convert.ToString(Convert.ToInt32(i, 2), 16).PadLeft(2, '0').ToUpper());

            ans.Append(string.Join(string.Empty, l));
        }

        return ans.ToString();
    }

    public static string Decrypt(string data, string key)
    {
        List<string> ans = [];

        foreach (var i in data.Cut(16))
        {
            string binMess = ToBin(i);
            string binKey = ToBin(key);


            string permutedBlock = Permute(binMess, InitialPermutation);
            string permutedKey = Permute(binKey, PermutedChoice1);


            int keyCenter = permutedKey.Length / 2;

            string L = permutedKey[..keyCenter];
            string R = permutedKey[keyCenter..];

            List<string> keyList = KeyGen(L, R).ToList();
```

```
            var l = Des_(permutedBlock,
keyList.AsEnumerable().Reverse()).Select(i =>
Convert.ToString(Convert.ToInt32(i, 2), 16).PadLeft(2, '0').ToUpper());

            ans.Add(string.Join(string.Empty, l));
        }


        return string.Join(string.Empty, ans.Select(s => new
string(s.Cut(2).Select(c => Convert.ToInt32(c, 16)).Where(i => i !=
0).Select(i => (char)i).ToArray())));
    }

    public static string CreateRandomKey()
    {
        char[] array = new char[16];

        for (int i = 0; i < array.Length; i++)
        {
            array[i] = "1234567890ABCDEF"[Random.Shared.Next(array.Length)];
        }

        return new string(array);
    }
}

using Kerberos;


public class Program
{
    private static void Main(string[] args)
    {
        var client1 = new Client(Des.CreateRandomKey());
        var client2 = new Client(Des.CreateRandomKey());
        var client3 = new Client(Des.CreateRandomKey());

        var server1 = servers.Single(s => s.Identifier == "server1");
        var server2 = servers.Single(s => s.Identifier == "server2");

        Kdc.AddClient(client1);
        Kdc.AddClient(client2);

        Console.WriteLine(client1.GetData(server1));
        Console.WriteLine(client1.GetData(server1));

        Thread.Sleep(TimeSpan.FromSeconds(16));

        Console.WriteLine(client1.GetData(server1));
    }


    public class Client
    {
        internal string _identifier;

        private string? _encryptedTgt;
        private string? _tgsClientKey;

        private Dictionary<string, string> _servers;
```

```
        public Client(string identifier)
        {
            _identifier = identifier;
            _servers = [];
        }


        public string GetData(Server server)
        {
            if (!_servers.ContainsKey(server.Identifier))
                ConnectToServer(server);

            try
            {
                return server.GetData(_identifier,
_servers[server.Identifier]);
            }
            catch (InvalidOperationException)
            {
                ConnectToServer(server);
                return server.GetData(_identifier,
_servers[server.Identifier]);
            }
        }


        private void ConnectToServer(Server server)
        {
            if (_encryptedTgt is null)
            {
                (_encryptedTgt, _tgsClientKey) = GetTgtWithKey();
            }

            string encryptedTgs;
            string clientServerKey;

            try
            {
                (encryptedTgs, clientServerKey) =
GetServerKey(server.Identifier, _encryptedTgt, _tgsClientKey!);
            }
            catch (InvalidOperationException)
            {
                (_encryptedTgt, _tgsClientKey) = GetTgtWithKey();
                (encryptedTgs, clientServerKey) =
GetServerKey(server.Identifier, _encryptedTgt, _tgsClientKey!);
            }

            CreateServerSession(server, encryptedTgs, clientServerKey);

            _servers[server.Identifier] = clientServerKey;
        }


        private (string, string) GetTgtWithKey()
        {
            Console.WriteLine("\nStep1 - Client send his identifier\n");

            string sessionKeys = Kdc.As.GetSessionKeys(_identifier);
```

```
            string decryptedSessionKeys = Des.Decrypt(sessionKeys,
_identifier);

            var encryptedTgt = decryptedSessionKeys.Split('|')[0];
            var tgsClientKey = decryptedSessionKeys.Split('|')[1];

            Console.WriteLine($"Step2 - Client get \n encrypted tgt:
{encryptedTgt} and \n tgsClientKey: {tgsClientKey}\n");

            return (encryptedTgt, tgsClientKey);
        }

        private (string, string) GetServerKey(string serverId, string
encryptedTgt, string tgsClientKey)
        {
            string aut1 = $"{_identifier}|{DateTime.Now}";

            string messageToTgs = $"{encryptedTgt}|{Des.Encrypt(aut1,
tgsClientKey)}|{serverId}";


            string keyWithTicket = Kdc.Tgs.GetServerKey(messageToTgs);

            Console.WriteLine($"Step3 - Client send \n serverId: {serverId},
\n encryptedTgt: {encryptedTgt} and \n tgsClientKey: {tgsClientKey}\n");

            var encryptedTgsWithServeKey = Des.Decrypt(keyWithTicket,
tgsClientKey);

            var encryptedTgs = encryptedTgsWithServeKey.Split('|')[0];
            var serverKey = encryptedTgsWithServeKey.Split('|')[1];

            Console.WriteLine($"Step4 - Client get \n encryptedTgs:
{encryptedTgs} and \n serverKey: {serverKey}\n");

            return (encryptedTgs, serverKey);
        }

        private void CreateServerSession(Server server, string encryptedTgs,
string clientServerKey)
        {
            Console.WriteLine($"Step5 - Client send \n encryptedTgs:
{encryptedTgs} and \n clientServerKey: {clientServerKey}\n");

            DateTime t4 = DateTime.Now;
            t4 = DateTime.Parse(t4.ToString());


            string aut2 = $"{_identifier}|{t4}";

            string encryptedTimeToCheck =
server.CreateSession($"{encryptedTgs}|{Des.Encrypt(aut2, clientServerKey)}");


            DateTime timeToCheck;

            try
            {
                timeToCheck =
DateTime.Parse(Des.Decrypt(encryptedTimeToCheck, clientServerKey));
            }
            catch (FormatException ex)
```

```
            {
                throw new InvalidOperationException("Invalid datetime
format.", ex);
            }


            if (timeToCheck - TimeSpan.FromSeconds(1) != t4)
                throw new InvalidOperationException("Incorrect response from
the server.");

            Console.WriteLine("Step6 - Client connected with server\n");
        }
    }


    public static class Kdc
    {
        private static HashSet<string> _clients;


        static Kdc()
        {
            _clients = [];
        }


        public static void AddClient(Client client) =>
_clients.Add(client._identifier);

        public static class As
        {
            public static string GetSessionKeys(string clientIdentifier)
            {
                if (!_clients.Contains(clientIdentifier))
                    throw new ArgumentException("Client with this identifier
is not registered in this server.");

                var tgsId = Tgs._id;
                var tgsKey = Tgs._key;


                var tgsClientKey = Des.CreateRandomKey();
//Random.Shared.Next(100).ToString();

                string tgt =
$"{clientIdentifier}|{tgsId}|{DateTime.Now}|{TimeSpan.FromSeconds(15)}|{tgsCl
ientKey}";

                return Des.Encrypt(Des.Encrypt(tgt, tgsKey) + '|' +
tgsClientKey, clientIdentifier);
            }
        }

        public static class Tgs
        {
            internal static string _id;
            internal static string _key;

            static Tgs()
            {
                _id = "123";
                _key = Des.CreateRandomKey();
```

```
            }


            public static string GetServerKey(string message)
            {
                string encryptedTgt;
                string encryptedAut1;
                string serverId;

                try
                {
                    encryptedTgt = message.Split('|')[0];
                    encryptedAut1 = message.Split('|')[1];
                    serverId = message.Split('|')[2];
                }
                catch (IndexOutOfRangeException e)
                {
                    throw new InvalidOperationException("Invalid format of
message.", e);
                }

                string serverKey;

                try
                {
                    serverKey = servers.Where(s => s.Identifier ==
serverId).Single().Key;
                }
                catch (InvalidOperationException e)
                {
                    throw new InvalidOperationException("Invalid server id.",
e);
                }


                string tgt;

                try
                {
                    tgt = Des.Decrypt(encryptedTgt, _key);
                }
                catch (InvalidOperationException e)
                {
                    throw new InvalidOperationException("Can't decrypt TGT.",
e);
                }

                string clientId;
                string tgsId;
                string t1;
                string p1;
                string clientTgsKey;

                try
                {
                    clientId = tgt.Split('|')[0];
                    tgsId = tgt.Split('|')[1];
                    t1 = tgt.Split('|')[2];
                    p1 = tgt.Split('|')[3];
                    clientTgsKey = tgt.Split('|')[4];
                }
                catch (IndexOutOfRangeException e)
```

```
                {
                    throw new InvalidOperationException("Invalid format of
TGT.", e);
                }

                if (tgsId != _id)
                    throw new InvalidOperationException("Invalid tgs id.");

                string aut1;

                try
                {
                    aut1 = Des.Decrypt(encryptedAut1, clientTgsKey);
                }
                catch (InvalidOperationException e)
                {
                    throw new InvalidOperationException("Can't decrypt TGT.",
e);
                }

                string clientId2;
                string t2;

                try
                {
                    clientId2 = aut1.Split('|')[0];
                    t2 = aut1.Split('|')[1];
                }
                catch (IndexOutOfRangeException e)
                {
                    throw new InvalidOperationException("Invalid format of
Aut1.", e);
                }

                if (clientId != clientId2)
                    throw new InvalidOperationException("The client who
requested access to the server is not the one who requested the ticket.");

                try
                {
                    if (DateTime.Parse(t1) + TimeSpan.Parse(p1) <
DateTime.Parse(t2))
                        throw new InvalidOperationException("The mandate has
expired.");
                }
                catch (FormatException ex)
                {
                    throw new InvalidOperationException("Invalid datetime
format.", ex);
                }

                string clientServerKey = Des.CreateRandomKey();
//Random.Shared.NextSingle().ToString();

                string tgs =
$"{clientId}|{serverId}|{DateTime.Now}|{TimeSpan.FromSeconds(15)}|{clientServ
erKey}";

                return Des.Encrypt($"{Des.Encrypt(tgs,
serverKey)}|{clientServerKey}", clientTgsKey);
            }
        }
```

```
    }


    static List<Server> servers =
    [
        new("server1", Des.CreateRandomKey()),
        new("server2", Des.CreateRandomKey())
    ];

    public class Server
    {
        public string Identifier { get; init; }
        internal string Key { get; init; }

        private Dictionary<string, string> _clients = [];


        public Server(string identifier, string key)
        {
            Identifier = identifier;
            Key = key;
        }


        public string CreateSession(string message)
        {
            string encryptedTgs;
            string encryptedAut2;

            try
            {
                encryptedTgs = message.Split('|')[0];
                encryptedAut2 = message.Split('|')[1];
            }
            catch (IndexOutOfRangeException e)
            {
                throw new InvalidOperationException("Invalid format of
message.", e);
            }

            string tgs;

            try
            {
                tgs = Des.Decrypt(encryptedTgs, Key);
            }
            catch (InvalidOperationException e)
            {
                throw new InvalidOperationException("Can't decrypt TGS.", e);
            }

            string clientId;
            string serverId;
            string t3;
            string p2;
            string clientServerKey;

            try
            {
                clientId = tgs.Split('|')[0];
                serverId = tgs.Split('|')[1];
                t3 = tgs.Split('|')[2];
```

```
                p2 = tgs.Split('|')[3];
                clientServerKey = tgs.Split('|')[4];
            }
            catch (IndexOutOfRangeException e)
            {
                throw new InvalidOperationException("Invalid format of TGS.",
e);
            }

            if (serverId != Identifier)
                throw new InvalidOperationException("Tgs server id not equal
with this server id.");

            string aut2;

            try
            {
                aut2 = Des.Decrypt(encryptedAut2, clientServerKey);
            }
            catch (InvalidOperationException e)
            {
                throw new InvalidOperationException("Can't decrypt aut2.",
e);
            }

            string clientId2;
            string t4;

            try
            {
                clientId2 = aut2.Split('|')[0];
                t4 = aut2.Split('|')[1];
            }
            catch (IndexOutOfRangeException e)
            {
                throw new InvalidOperationException("Invalid format of
Aut1.", e);
            }

            if (clientId != clientId2)
                throw new InvalidOperationException("Client id in tgs not
equal to client id in aut block.");

            try
            {
                if (DateTime.Parse(t3) + TimeSpan.Parse(p2) <
DateTime.Parse(t4))
                    throw new InvalidOperationException("The tgs has
expired.");
            }
            catch (FormatException ex)
            {
                throw new InvalidOperationException("Invalid datetime
format.", ex);
            }

            _clients[clientId] = clientServerKey;

            Task.Delay(DateTime.Parse(t3) + TimeSpan.Parse(p2) -
DateTime.Now).ContinueWith(t =>
            {
                _clients.Remove(clientId);
```

```
            });

            string timeToCheck = (DateTime.Parse(t4) +
TimeSpan.FromSeconds(1)).ToString();
            return Des.Encrypt(timeToCheck, clientServerKey);
        }

        public string GetData(string clientId, string clientServerKey)
        {
            if (!_clients.TryGetValue(clientId, out var clientServerKey2))
                throw new InvalidOperationException("This client not
registered on server.");

            if (clientServerKey != clientServerKey2)
                throw new InvalidOperationException("Invalid server key.");



            return "Client get data from server";
        }
    }
}


//Task.Delay(new TimeSpan(0, 0, 15)).ContinueWith(t =>
//{
//    Console.WriteLine("Timeout");
//});

// Thread.Sleep(TimeSpan.FromSeconds(60));
```