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. 953505

Яцына А. М.

Проверил:

Олисейчик В.В.

Минск, 2022

**1. Введение**

Цель: создать приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES.

**2. Блок схемы алгоритмов**

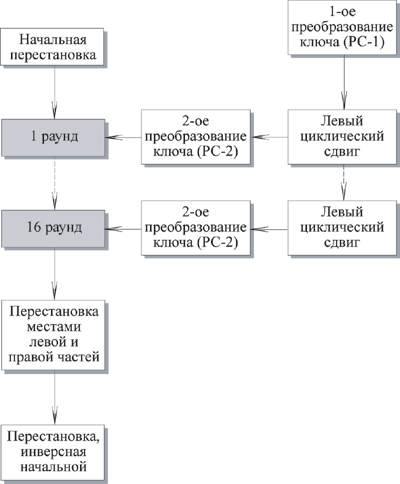


Рис 1. Общая схема DES.



Рис 2. Схема протокола Kerberos.

**3. Результат выполнения программы**

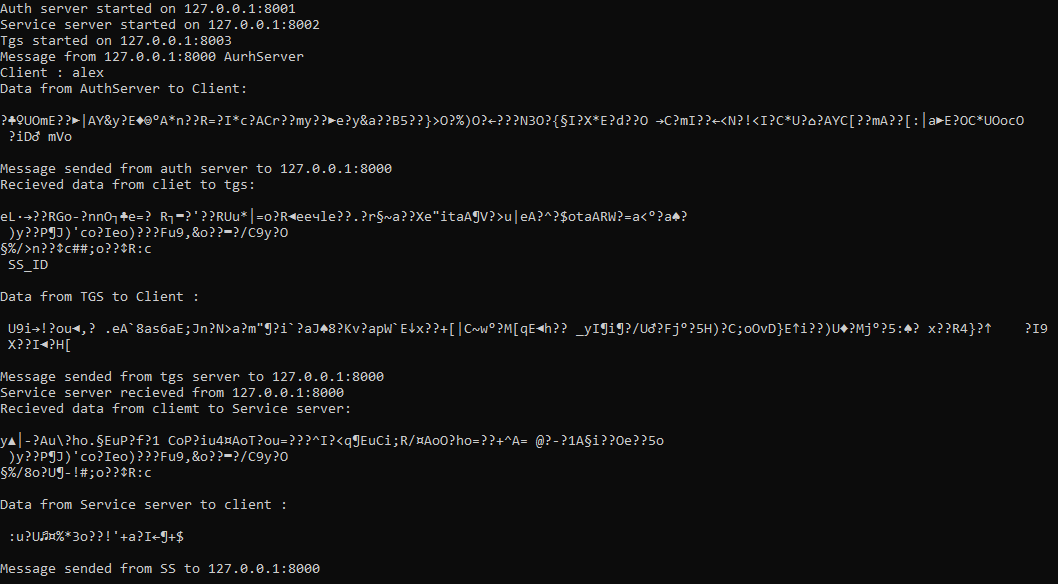


Рис 3. Результат работы серверов.



Рис 2. Результат работы клиента

**4. Вывод**

Выше мы рассмотрели протокол сетевой аутентификации Kerberos. Следует отметить, что этот протокол отличается гибкостью и эффективностью использования, а также обеспечивает повышенный уровень безопасности. С бурным развитием Интернета, локальных сетей, виртуальных частных сетей, электронной коммерции, этот протокол, похоже, является одним из тех, которые удовлетворяют всем требованиям безопасности в сегодняшней информационной среде.

**4. Исходный код программы**

namespace lab2

{

class AuthServer

{

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

public AuthServer()

{

users.Add("alex");

}

public void Listen()

{

UdpClient reciever = new UdpClient(Config.asPort);

Console.WriteLine($"Auth server started on 127.0.0.1:{Config.asPort}");

IPEndPoint remoteIp = null;

try

{

while (true)

{

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

remoteIp.Port = Config.cPort;

var answer = new Message();

var message = CustomConverter<Message>.Deserialize(Encoding.UTF8.GetString(data));

if (message.Type == MessageType.CToAs)

{

var id = message.Data[0];

Console.WriteLine($"Message from {remoteIp.Address}:{remoteIp.Port} AurhServer");

Console.WriteLine($"Client : {id}");

if (users.Contains(id))

{

answer.Type = MessageType.AsToC;

var ticket = new TicketGranting

{

Clidentity = id,

SIdentity = "tgs",

Duration = Config.asTDur.Ticks,

Time = DateTime.Now,

Key = Config.kCTgs.ToString()

};

var temp = CustomConverter<TicketGranting>.Serialize(ticket);

var ticketEncr = DES.Encrypt(DES.Encrypt( temp , Config.kAsTgs),Config.kC);

var kCTgsEncr = DES.Encrypt(Config.kCTgs, Config.kC);

Console.WriteLine($"Data from AuthServer to Client: \n\n{ticketEncr} \n {kCTgsEncr}\n");

answer.Data.Add(ticketEncr);

answer.Data.Add(kCTgsEncr);

}

else

{

answer.Type = MessageType.AccessDenied;

Console.WriteLine("Access denied.");

}

answer.Send(remoteIp);

Console.WriteLine($"Message sended from auth server to {remoteIp.Address}:{remoteIp.Port} ");

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

namespace lab2

{

static class Config

{

public static readonly string kC = "dghfghfs";

public static readonly string kCTgs = "dfgdfg";

public static readonly string kCSs = "sdfghfg";

public static readonly string kAsTgs = "sdffgb";

public static readonly string kTgsSs = "aswerwr";

public static readonly int cPort = 8000;

public static readonly int asPort = 8001;

public static readonly int ssPort = 8002;

public static readonly int tgsPort = 8003;

public static readonly TimeSpan asTDur = new TimeSpan(24, 0, 0);

public static readonly TimeSpan tgsTDur = new TimeSpan(12, 0, 0);

}

}

namespace lab2

{

public class CustomConverter <T>

{

public static string Serialize(object data)

{

return JsonConvert.SerializeObject(data).ToString();

}

public static T Deserialize( string data)

{

return JsonConvert.DeserializeObject<T>(data);

}

public static string ToStringFromBytes(byte[] data)

{

return Encoding.UTF8.GetString(data);

}

public static byte[] ToBytesFromString(string data)

{

return Encoding.UTF8.GetBytes(data);

}

}

}

namespace lab2

{

public class DES

{

private static int sizeOfBlock = 64;

private static int sizeOfChar = 8;

private static int shiftKey = 2;

private static int quantityOfRounds = 16;

private static string EncodeKey { get; set; }

private static string DecodeKey { get; set; }

static string[] Blocks;

private static string ExpendingStringSize(string input)

{

var currInput = new StringBuilder(input);

while ((currInput.Length \* sizeOfChar) % sizeOfBlock != 0)

{

currInput.Append(" ");

}

return currInput.ToString();

}

private static void CutStringIntoBlocks(string input)

{

Blocks = new string[(input.Length \* sizeOfChar) / sizeOfBlock];

var lengthOfBlock = input.Length / Blocks.Length;

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

{

Blocks[i] = input.Substring(i \* lengthOfBlock, lengthOfBlock);

Blocks[i] = StringToBinary(Blocks[i]);

}

}

private static string StringToBinary(string input)

{

var result = new StringBuilder();

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

{

var charB = new StringBuilder(Convert.ToString(input[i], 2));

while (charB.Length < sizeOfChar)

{

charB.Insert(0,"0");

}

result.Append(charB.ToString());

}

return result.ToString();

}

private static string ExpendingKey(string key, int lengthKey)

{

var result = new StringBuilder();

if (key.Length > lengthKey)

{

result.Append(key.Substring(0, lengthKey));

}

else

{

var zeros = new StringBuilder(key);

while (zeros.Length < lengthKey)

{

zeros.Insert(0,"0");

}

result.Append(zeros.ToString());

}

return result.ToString();

}

private static string EncodeOneRound(string input, string key)

{

var L = input.Substring(0, input.Length / 2);

var R = input.Substring(input.Length / 2, input.Length / 2);

return (R + XOR(L, f(R, key)));

}

private static string f(string val1, string val2)

{

return XOR(val1, val2);

}

private static string XOR(string val1, string val2)

{

var result = new StringBuilder();

for (var i = 0; i < val1.Length; i++)

{

bool a = Convert.ToBoolean(Convert.ToInt32(val1[i].ToString()));

bool b = Convert.ToBoolean(Convert.ToInt32(val2[i].ToString()));

if (a ^ b)

result.Append("1");

else

result.Append("0");

}

return (result.ToString());

}

private static string DecodeOneRound(string input, string key)

{

var L =input.Substring(0,input.Length / 2);

var R = input.Substring(input.Length /2, input.Length / 2);

return XOR(f(L, key), R) + L;

}

private static string NextKey(string key)

{

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

{

key = key[key.Length - 1] + key;

key = key.Remove(key.Length - 1);

}

return key;

}

private static string PrevKey(string key)

{

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

{

key = key + key[0];

key = key.Remove(0, 1);

}

return key;

}

private static string FromBinaryToString(string input)

{

var result = new StringBuilder();

while (input.Length > 0)

{

var charB = input.Substring(0, sizeOfChar);

input = input.Remove(0, sizeOfChar);

var a = 0;

var degree = charB.Length - 1;

foreach (char c in charB)

a += Convert.ToInt32(c.ToString()) \* (int)Math.Pow(2, degree--);

result.Append(((char)a).ToString());

}

return result.ToString();

}

public static string Encrypt(string input, string key)

{

input = ExpendingStringSize(input);

CutStringIntoBlocks(input);

EncodeKey = key;

key = ExpendingKey(key, input.Length / (2 \* Blocks.Length));

key = StringToBinary(key);

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

{

for (int j = 0; j < Blocks.Length; j++)

{

Blocks[j] = EncodeOneRound(Blocks[j], key);

}

key = NextKey(key);

}

key = PrevKey(key);

DecodeKey = FromBinaryToString(key);

var result = new StringBuilder();

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

{

result.Append(FromBinaryToString(Blocks[i].ToString()));

}

return result.ToString();

}

public static string Decrypt(string input, string key ="")

{

ExpendingStringSize(input);

CutStringIntoBlocks(input);

key = ExpendingKey(key, input.Length / (2 \* Blocks.Length));

key = StringToBinary(key);

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

{

key = NextKey(key);

}

key = FromBinaryToString(PrevKey(key));

key = StringToBinary(key);

input = StringToBinary(input);

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

{

for (int j = 0; j < Blocks.Length; j++)

{

Blocks[j] = DecodeOneRound(Blocks[j], key);

}

key = PrevKey(key);

}

var result = new StringBuilder();

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

{

result.Append(FromBinaryToString(Blocks[i].ToString()));

}

return result.ToString();

}

public static bool CheckTime(DateTime time1, DateTime time2, long duration)

{

TimeSpan timeSpan = new TimeSpan(duration);

if (time2 < time1 + timeSpan)

return true;

else

return false;

}

}

}

namespace lab2

{

public enum MessageType

{

CToAs,

AsToC,

CToTgs,

TgsToC,

CToSs,

SsToC,

TicketNotValid,

AccessDenied

}

public class Message

{

public MessageType Type { get; set; }

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

public Message()

{

Data = new List<string>();

}

public Message(MessageType messageType = 0)

{

Data = new List<string>();

Type = messageType;

}

public void Send(IPEndPoint remoteIp)

{

UdpClient sender = new UdpClient();

try

{

var js = JsonConvert.SerializeObject(this);

byte[] dgram = Encoding.UTF8.GetBytes(js);

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

}

finally

{

sender.Close();

}

}

}

}

namespace lab2

{

internal class Program

{

static void Main(string[] args)

{

var authServer = new AuthServer();

var tgs = new TicketGrantingServer();

var server = new Server();

try

{

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

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

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

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

Console.ReadKey();

}

}

}

namespace lab2

{

class Server

{

public void Listen()

{

UdpClient reciever = new UdpClient(Config.ssPort);

Console.WriteLine($"Service server started on 127.0.0.1:{Config.ssPort}");

IPEndPoint remoteIp = null;

try

{

while (true)

{

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

remoteIp.Port = Config.cPort;

var message = CustomConverter<Message>.Deserialize(Encoding.UTF8.GetString(data));

Console.WriteLine($"Service server recieved from {remoteIp.Address}:{remoteIp.Port}");

if (message.Type == MessageType.CToSs)

{

var tgsJs = DES.Decrypt(message.Data[0], Config.kTgsSs);

var tgs = JsonConvert.DeserializeObject<TicketGranting>(tgsJs);

var autJs = DES.Decrypt(message.Data[1], Config.kCSs);

var aut = JsonConvert.DeserializeObject<TimeMark>(autJs);

Console.WriteLine($"Recieved data from cliemt to Service server: \n\n {message.Data[0]} \n {message.Data[1]}\n");

var answer = new Message();

if(DES.CheckTime(tgs.Time, aut.Time, tgs.Duration))

{

answer.Type = MessageType.SsToC;

var timeJs = JsonConvert.SerializeObject(aut.Time.Ticks + 1);

var encr = DES.Encrypt(timeJs, Config.kCSs);

Console.WriteLine($"Data from Service server to client : \n\n {encr}\n");

answer.Data.Add(encr);

}

else

{

answer.Type = MessageType.TicketNotValid;

}

answer.Send(remoteIp);

Console.WriteLine($"Message sended from SS to {remoteIp.Address}:{remoteIp.Port}");

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

namespace lab2

{

class TicketGranting

{

public string Clidentity { get; set; }

public string SIdentity { get; set; }

public DateTime Time { get; set; }

public long Duration { get; set; }

public string Key { get; set; }

}

}

namespace lab2

{

class TicketGrantingServer

{

public void Listen()

{

UdpClient reciever = new UdpClient(Config.tgsPort);

Console.WriteLine($"Tgs started on 127.0.0.1:{Config.tgsPort}");

IPEndPoint remoteIp = null;

try

{

while (true)

{

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

remoteIp.Port = Config.cPort;

var message = CustomConverter<Message>.Deserialize(Encoding.UTF8.GetString(data));

if (message.Type == MessageType.CToTgs)

{

var tgtJs = DES.Decrypt(message.Data[0], Config.kAsTgs);

var tgt = JsonConvert.DeserializeObject<TicketGranting>(tgtJs);

var autJs = DES.Decrypt(message.Data[1], Config.kCTgs);

var aut = JsonConvert.DeserializeObject<TimeMark>(autJs);

var id = message.Data[2];

Console.WriteLine($"Recieved data from cliet to tgs: \n\n{message.Data[0]} \n {message.Data[1]} \n {message.Data[2]}\n");

var answer = new Message();

if (tgt.Clidentity == aut.Cl)

{

if (DES.CheckTime(tgt.Time, aut.Time, tgt.Duration))

{

answer.Type = MessageType.TgsToC;

var ticket = new TicketGranting()

{

Clidentity = aut.Cl,

SIdentity = id,

Duration = Config.tgsTDur.Ticks,

Time = DateTime.Now,

Key = Config.kCSs

};

var ticketEncr =DES.Encrypt( DES.Encrypt(JsonConvert.SerializeObject(ticket), Config.kTgsSs),Config.kCTgs);

var kCSsEncr = DES.Encrypt(Config.kCSs, Config.kCTgs);

Console.WriteLine($"Data from TGS to Client : \n\n {ticketEncr} \n {kCSsEncr} \n");

answer.Data.Add(ticketEncr);

answer.Data.Add(kCSsEncr);

}

else

{

answer.Type = MessageType.TicketNotValid;

Console.WriteLine($"Ticket not validin tgs");

}

}

else

{

answer.Type = MessageType.AccessDenied;

Console.WriteLine("Access denied in tgs.");

}

answer.Send(remoteIp);

Console.WriteLine($"Message sended from tgs server to {remoteIp.Address}:{remoteIp.Port} ");

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

namespace lab2

{

class TimeMark

{

public string Cl { get; set; }

public DateTime Time { get; set; }

}

}

namespace lab2\_Client

{

class Client

{

public string Login { get; set; }

private string Tgt { get; set; }

private string Tgs { get; set; }

private string KCTgs { get; set; }

private string KCSs { get; set; }

private DateTime Time { get; set; }

private readonly IPEndPoint AsEP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Config.asPort);

private readonly IPEndPoint SsEP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Config.ssPort);

private readonly IPEndPoint TgsEP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Config.tgsPort);

public void Reg(string login)

{

Login = login;

var message = new Message(MessageType.CToAs);

message.Data.Add(Login);

message.Send(AsEP);

}

public void GetPerm()

{

if(KCTgs != null && Tgt!=null)

{

var message = new Message(MessageType.CToTgs);

message.Data.Add(Tgt);

var mark = new TimeMark()

{

Cl = Login,

Time = DateTime.Now

};

var aut = JsonConvert.SerializeObject(mark).ToString();

message.Data.Add(DES.Encrypt(aut, KCTgs));

message.Data.Add("SS\_ID");

message.Send(TgsEP);

}

}

public void Listen()

{

var reciever = new UdpClient(Config.cPort);

IPEndPoint remoteIp = null;

try

{

while(true)

{

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

var message = CustomConverter<Message>.Deserialize(Encoding.UTF8.GetString(data));

switch (message.Type)

{

case MessageType.AsToC:

Tgt = DES.Decrypt(message.Data[0], Config.kC);

KCTgs = DES.Decrypt(message.Data[1], Config.kC);

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

{

if(!KCTgs[i].Equals(" "))

{

KCTgs = KCTgs.Remove(i);

break;

}

}

Console.WriteLine("AS to C successfull");

break;

case MessageType.TgsToC:

Tgs = DES.Decrypt(message.Data[0], KCTgs);

KCSs = DES.Decrypt(message.Data[1], KCTgs);

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

{

if (!KCSs[i].Equals(" "))

{

KCSs = KCSs.Remove(i);

break;

}

}

var locMessage = new Message(MessageType.CToSs);

locMessage.Data.Add(Tgs);

var mark = new TimeMark()

{

Cl = Login,

Time = DateTime.Now

};

Time = mark.Time;

var aut = JsonConvert.SerializeObject(mark);

locMessage.Data.Add(DES.Encrypt(aut, KCSs));

locMessage.Send(SsEP);

Console.WriteLine("TGS to C successfull");

break;

case MessageType.SsToC:

var time = DES.Decrypt(message.Data[0], KCSs);

var forCheck = JsonConvert.DeserializeObject<long>(time);

if (Time.Ticks + 1 == forCheck)

{

Console.WriteLine("successfull");

}

break;

case MessageType.TicketNotValid:

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

break;

case MessageType.AccessDenied:

Console.WriteLine("Access denied");

break;

default:

Console.WriteLine("Something wrong in type message");

break;

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

}

}

}

namespace lab2\_Client

{

static class Config

{

public static readonly string kC = "dghfghfs";

public static readonly int cPort = 8000;

public static readonly int asPort = 8001;

public static readonly int ssPort = 8002;

public static readonly int tgsPort = 8003;

}

}

namespace lab2\_Client

{

internal class Program

{

static void Main(string[] args)

{

var client = new Client();

try

{

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

Task.Run(() => client.Reg("alex"));

Thread.Sleep(5000);

Task.Run(() => client.GetPerm());

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

Console.ReadKey();

}

}

}