1Министерство 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. 953502

Трофимук Г.А

Проверил

Протько М.И.

Минск, 2022

# 1. Введение

1. Изучить теоретические сведения.
2. Создать приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES.

В интерфейсе приложения должны быть наглядно представлены:

* Исходные данные протокола (модули, ключи, секретные данные и т.п.);
* Данные, передаваемые по сети каждой из сторон;
* Проверки, выполняемые каждым из участников.

Процесс взаимодействия между сторонами протокола может быть реализован при помощи буферных переменных. Также необходимо выделить каждый из этапов протоколов для того, чтобы его можно было отделить от остальных.

**2. Теоретические сведения**

**Протокол Kerberos**

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

Протокол Kerberos, достаточно гибкий и имеющий возможности тонкой настройки под конкретные применения, существует в нескольких версиях. Мы рассмотрим упрощенный механизм аутентификации, реализованный с помощью протокола Kerberos версии 5 (рис. 1):



Рисунок 1 Схема протокола Kerberos

Прежде всего стоит сказать, что 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 (Алгоритм шифрования приводится ниже).

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.

Нужно отметить, что 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).

В алгоритме Kerberos могут применяться различные алгоритмы блочного симметричного шифрования. Для целей настоящей работы будем использовать алгоритм DES:

**Алгоритм DES Основные сведения**

Одной из наиболее известных криптографических систем с закрытым ключом является DES – Data Encryption Standard. Эта система первой получила статус государственного стандарта в области шифрования данных. Она разработана специалистами фирмы IBM и вступила в действие в США 1977 году. Алгоритм DES по-прежнему широко применяется и заслуживает внимания при изучении блочных шифров с закрытым ключом.

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.

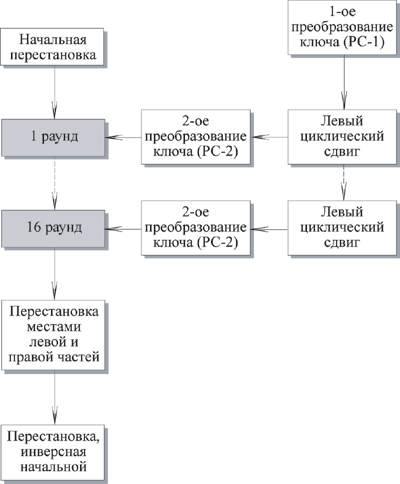


Рисунок 2 Общая схема DES

**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*.

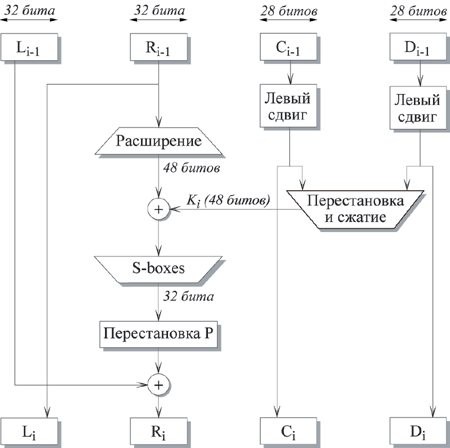


Рисунок 3 - I-ый раунд DES

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. 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.

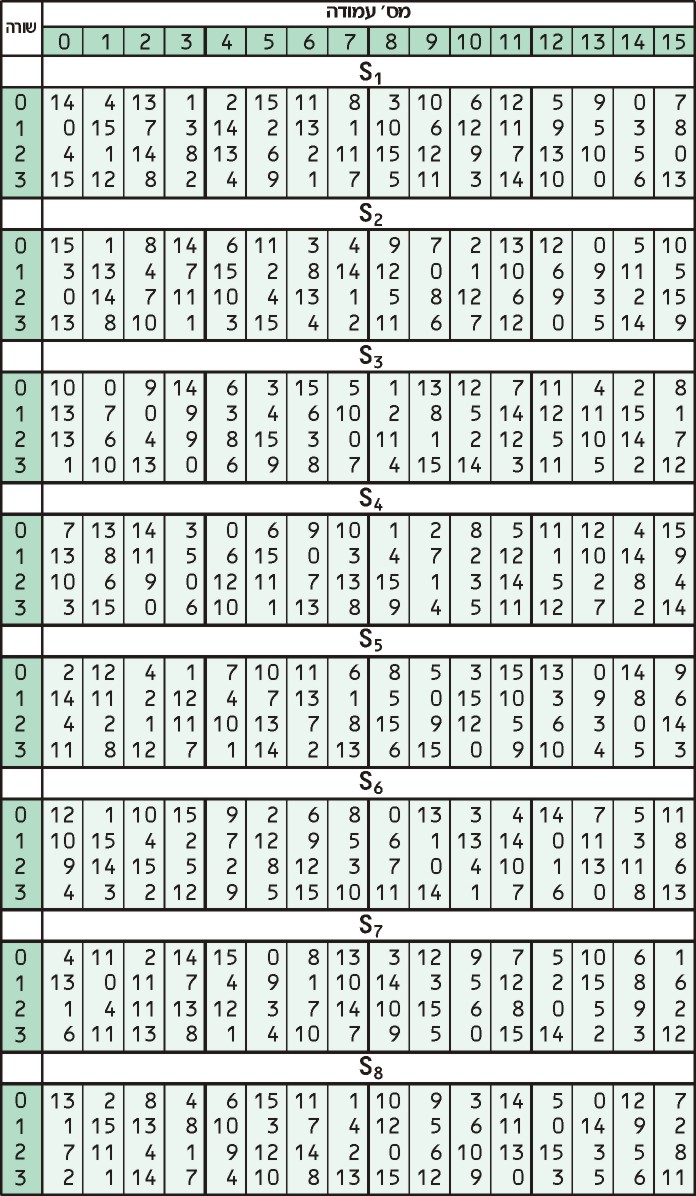


Рисунок 4 - S-boxes

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**

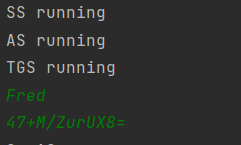
Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма, вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1). Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые 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*).

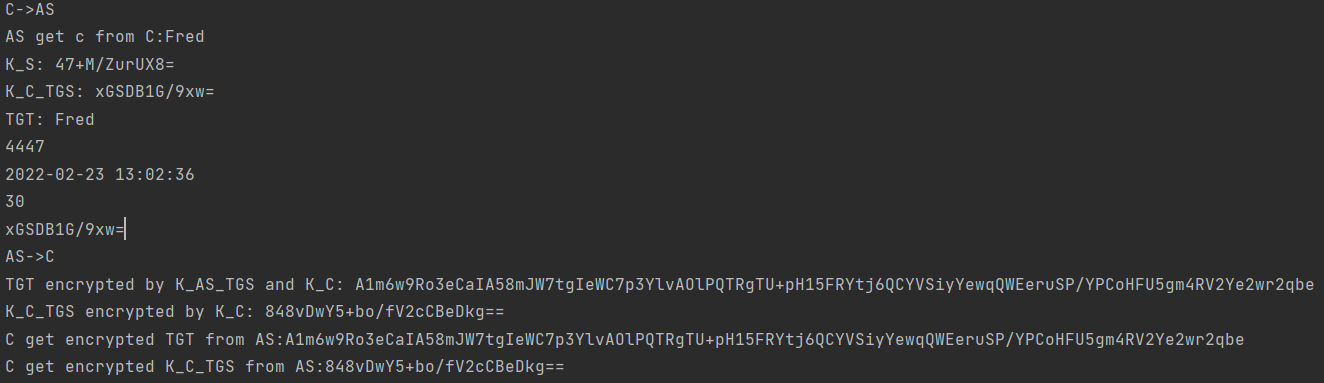
**Дешифрование**

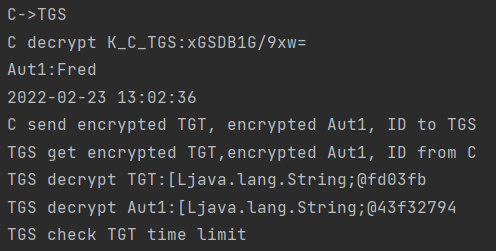
Процесс дешифрования аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности. *K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

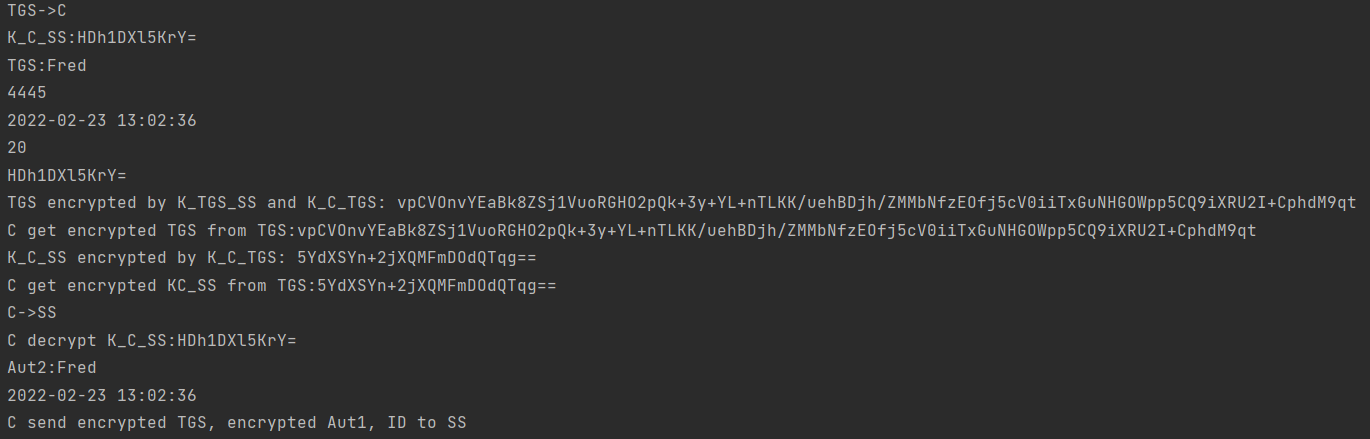
# 3. Результаты выполнения программы

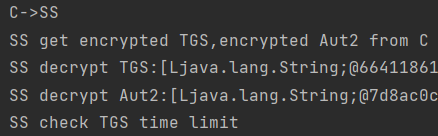
При вводимых значениях Fred и 47+M/ZurUX8=:

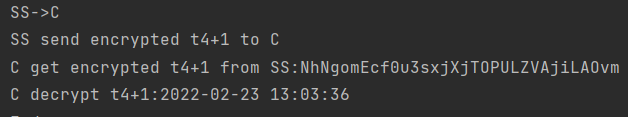




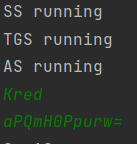


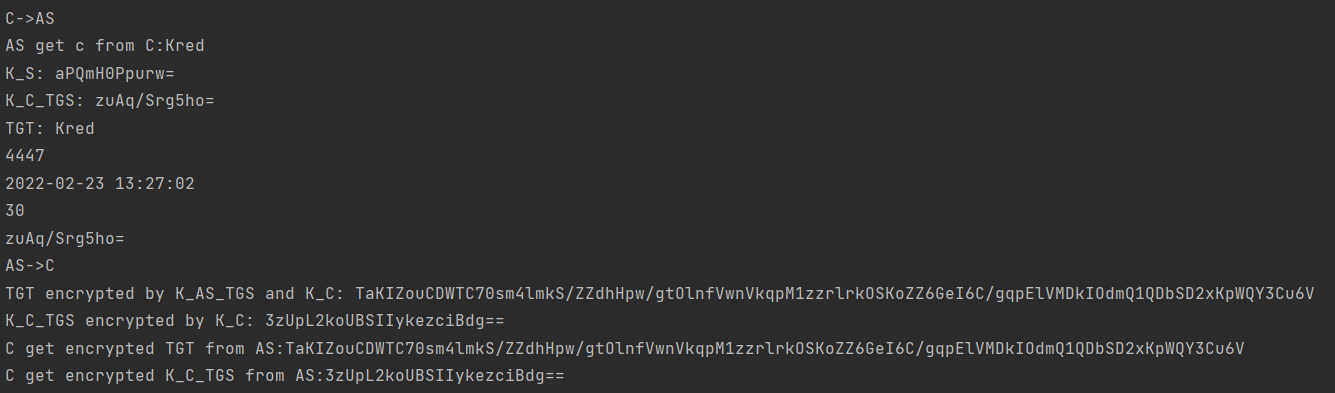


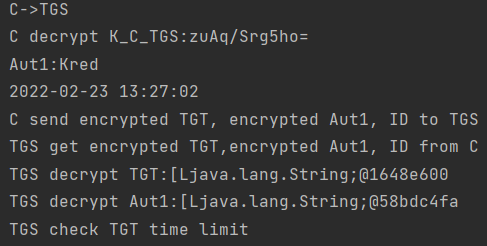


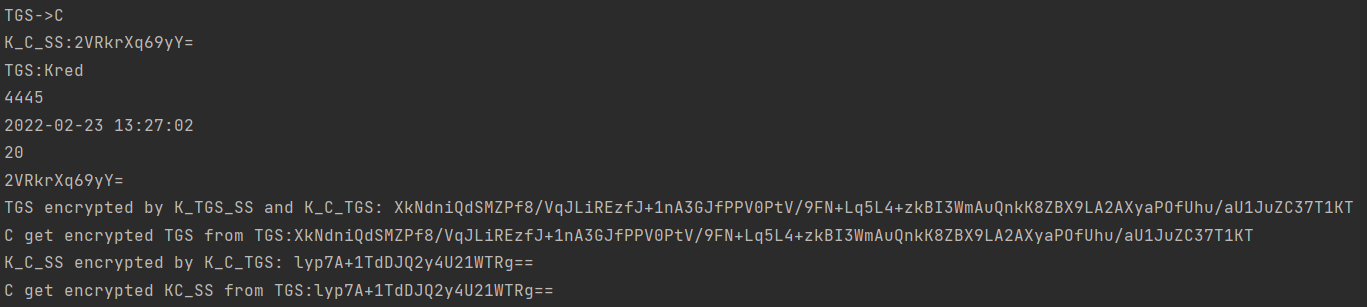


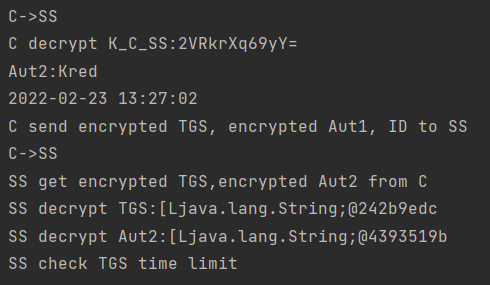
При вводимых значениях Fred и 47+M/ZurUX8=:

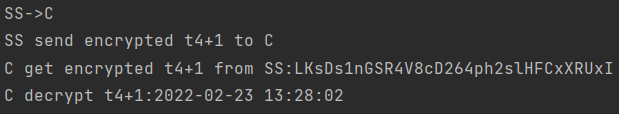












# 4. Выводы

Kerberos— [сетевой протокол](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F), который предлагает механизм взаимной [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) клиента и сервера перед установлением связи между ними, причём в протоколе учтён тот факт, что начальный обмен информацией между клиентом и сервером происходит в незащищенной среде, а передаваемые пакеты могут быть перехвачены и модифицированы. Другими словами, протокол идеально подходит для применения в Интернет и аналогичных сетях. 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.

DES ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Data Encryption Standard) — алгоритм для [симметричного](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80) [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), разработанный фирмой [IBM](https://ru.wikipedia.org/wiki/IBM) и утверждённый правительством [США](https://ru.wikipedia.org/wiki/%D0%A1%D0%A8%D0%90) в 1977 году как официальный стандарт ([FIPS](https://ru.wikipedia.org/wiki/FIPS) 46-3). Размер блока для DES равен 64 [битам](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82). В основе алгоритма лежит [сеть Фейстеля](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D1%8C_%D0%A4%D0%B5%D0%B9%D1%81%D1%82%D0%B5%D0%BB%D1%8F) с 16 циклами ([раундами](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%83%D0%BD%D0%B4_(%D0%B2_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%B8))) и [ключом](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)), имеющим длину 56 [бит](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82). Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований.

# 6. КОД ПРОГРАММЫ

**Main.java:**

package com.company;

import javax.crypto.\*;

import javax.crypto.spec.SecretKeySpec;

import java.security.NoSuchAlgorithmException;

import java.time.LocalDateTime;

import java.util.Base64;

public class Main {

public static void main(String[] args) {

Thread serviceServerThread = new Thread(() -> {

while (true)

{

ServiceServer serviceServer = new ServiceServer();

serviceServer.run();

}

});

serviceServerThread.start();

Thread authenticationThread = new Thread(() -> {

while (true)

{

AuthenticationServer authenticationServer = new AuthenticationServer();

authenticationServer.run();

}

});

authenticationThread.start();

Thread ticketGrantingThread = new Thread(() -> {

while (true)

{

TicketGrantingServer ticketGrantingServer = new TicketGrantingServer();

ticketGrantingServer.run();

}

});

ticketGrantingThread.start();

Client client = new Client();

client.authorization();

}

}

**Client.java:**

package com.company;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.\*;

import java.time.LocalDateTime;

public class Client

{

private DatagramSocket socket;

private InetAddress address;

private byte[] encryptedTGT = new byte[Constants.defaultSize];

private byte[] encryptedTGS = new byte[Constants.defaultSize];

private String c;//идентификатор

private String K\_C;//основной ключ C

private String K\_C\_TGS;//ключ, выдаваемый C для доступа к серверу выдачи разрешений TGS

private String K\_C\_SS;

private String Aut1;//аутентификационный блок - Aut1 = {с,t2}, t2 - метка времени

private String Aut2;//где Aut2={c,t4}.

private String t4SS;//t4+1

public Client()

{

try {

socket = new DatagramSocket();

} catch (SocketException e) {

e.printStackTrace();

}

try {

address = InetAddress.getByName("localhost");

} catch (UnknownHostException e) {

e.printStackTrace();

}

}

public void authorization() {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

try {

c = reader.readLine();

K\_C = reader.readLine();

} catch (IOException e) {

e.printStackTrace();

return;

}

DatagramPacket packet;

DesEncrypter encrypterK\_C = new DesEncrypter(DesEncrypter.getSK(K\_C));

SendPacket(c, Constants.authenticationServerPort);

packet = ReceivePacket();

System.out.println("C get encrypted TGT from AS:"+ (new String(packet.getData(), 0, packet.getLength())));

encryptedTGT = encrypterK\_C.decrypt((new String(packet.getData(), 0, packet.getLength()).getBytes()));

packet = ReceivePacket();

System.out.println("C get encrypted K\_C\_TGS from AS:"+ (new String(packet.getData(), 0, packet.getLength())));

K\_C\_TGS = encrypterK\_C.decryptToStr((new String(packet.getData(), 0, packet.getLength()).getBytes()));

System.out.println("C->TGS");

System.out.println("C decrypt K\_C\_TGS:"+ K\_C\_TGS);

DesEncrypter encrypterK\_C\_TGS = new DesEncrypter(DesEncrypter.getSK(K\_C\_TGS));

Aut1 = c + "\n" + LocalDateTime.now().format(Constants.formatter);

System.out.println("Aut1:"+ Aut1);

String msg = new String(encryptedTGT, 0, encryptedTGT.length) + "\n";

byte[] encryptedAut1 = encrypterK\_C\_TGS.encrypt(Aut1);

msg += new String(encryptedAut1, 0, encryptedAut1.length) + "\n";

msg += Constants.serviceServerPort;

SendPacket(msg, Constants.ticketGrantingServerPort);

System.out.println("C send encrypted TGT, encrypted Aut1, ID to TGS");

////

packet = ReceivePacket();

System.out.println("C get encrypted TGS from TGS:"+ (new String(packet.getData(), 0, packet.getLength())));

encryptedTGS = encrypterK\_C\_TGS.decrypt((new String(packet.getData(), 0, packet.getLength()).getBytes()));

packet = ReceivePacket();

System.out.println("C get encrypted KC\_SS from TGS:"+ (new String(packet.getData(), 0, packet.getLength())));

K\_C\_SS = encrypterK\_C\_TGS.decryptToStr((new String(packet.getData(), 0, packet.getLength()).getBytes()));

System.out.println("C->SS");

System.out.println("C decrypt K\_C\_SS:"+ K\_C\_SS);

DesEncrypter encrypterK\_C\_SS = new DesEncrypter(DesEncrypter.getSK(K\_C\_SS));

Aut2 = c + "\n" + LocalDateTime.now().format(Constants.formatter);

System.out.println("Aut2:"+ Aut2);

msg = new String(encryptedTGS, 0, encryptedTGT.length) + "\n";

byte[] encryptedAut2 = encrypterK\_C\_SS.encrypt(Aut2);

msg += new String(encryptedAut2, 0, encryptedAut2.length) + "\n";

SendPacket(msg, Constants.serviceServerPort);

System.out.println("C send encrypted TGS, encrypted Aut1, ID to SS");

packet = ReceivePacket();

System.out.println("C get encrypted t4+1 from SS:"+ (new String(packet.getData(), 0, packet.getLength())));

try {

t4SS = encrypterK\_C\_SS.decryptToStr((new String(packet.getData(), 0, packet.getLength()).getBytes()));

System.out.println("C decrypt t4+1:" + t4SS);

System.out.println("End");

}

catch (Exception e)

{

System.out.println("C can't decrypt t4+1 from SS");

System.exit(-1);

}

}

private DatagramPacket ReceivePacket()

{

byte[] buf = new byte[Constants.defaultSize];

DatagramPacket packet = new DatagramPacket(buf, buf.length);

try {

socket.receive(packet);

} catch (IOException e) {

e.printStackTrace();

}

return packet;

}

private void SendPacket(String msg, int receiver)

{

byte[] buf = msg.getBytes();

DatagramPacket packet = new DatagramPacket(buf, buf.length, address, receiver);

try {

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

}

private void SendPacket(byte[] buf, int receiver)

{

DatagramPacket packet = new DatagramPacket(buf, buf.length, address, receiver);

try {

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

}

public void close() {

socket.close();

}

}

**AuthenticationServer.java:**

package com.company;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.time.LocalDateTime;

import java.util.Base64;

import java.util.HashMap;

import java.util.Objects;

public class AuthenticationServer {

private DatagramSocket socket;

private boolean running;

private byte[] buf = new byte[Constants.defaultSize];

private HashMap<String, String> clientsMap = new HashMap<>();

private String K\_C\_TGS;//ключ, выдаваемый C для доступа к серверу выдачи разрешений TGS ;

private String TGT;//Ticket Granting Ticket - билет на доступ к серверу выдачи разрешений

private Integer p1 = 30;//период действия билета

public AuthenticationServer() {

try {

socket = new DatagramSocket(Constants.authenticationServerPort);

} catch (SocketException e) {

e.printStackTrace();

}

clientsMap.put("Fred","47+M/ZurUX8=");

clientsMap.put("Kred","aPQmH0Ppurw=");

clientsMap.put("Vred","eiW/7Fj9tRo=");

}

public void run() {

running = true;

System.out.println("AS running");

while (running) {

DatagramPacket packet = new DatagramPacket(buf, buf.length);

try {

socket.receive(packet);

System.out.println("C->AS");

} catch (IOException e) {

e.printStackTrace();

}

InetAddress address = packet.getAddress();

int port = packet.getPort();

String c= new String(packet.getData(), 0, packet.getLength());

if (!clientsMap.containsKey(c)) {

System.out.println("Запрос отклонён");

System.exit(-1);

}

System.out.println("AS get c from C:" + c);

DesEncrypter encrypterK\_AS\_TGS = new DesEncrypter(Constants.K\_AS\_TGS);

byte[] decodedKey = Base64.getDecoder().decode(clientsMap.get(c));

SecretKey originalKey = new SecretKeySpec(decodedKey, 0, decodedKey.length, "DES");

DesEncrypter encrypterK\_C = new DesEncrypter(originalKey);

System.out.println("K\_S: "+clientsMap.get(c));

K\_C\_TGS = Base64.getEncoder().encodeToString(Objects.requireNonNull(DesEncrypter.generateSK()).getEncoded());

System.out.println("K\_C\_TGS: "+K\_C\_TGS);

TGT = c + "\n";

TGT += Constants.authenticationServerPort + "\n";

TGT += LocalDateTime.now().format(Constants.formatter) + "\n";

TGT += p1 + "\n";

TGT += K\_C\_TGS;

System.out.println("TGT: "+TGT);

System.out.println("AS->C");

buf = encrypterK\_C.encrypt(encrypterK\_AS\_TGS.encrypt(TGT));

packet = new DatagramPacket(buf, buf.length, address, port);

try {

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

System.out.println("TGT encrypted by K\_AS\_TGS and K\_C: "+new String(buf,0,buf.length));

buf = encrypterK\_C.encrypt(K\_C\_TGS);

packet = new DatagramPacket(buf, buf.length, address, port);

try {

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

System.out.println("K\_C\_TGS encrypted by K\_C: "+new String(buf,0,buf.length));

}

socket.close();

}

}

**Constants.java:**

package com.company;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import java.security.NoSuchAlgorithmException;

import java.time.format.DateTimeFormatter;

public class Constants {

public final static int defaultSize = 256;

public final static int serviceServerPort = 4445;

public final static int ticketGrantingServerPort = 4446;

public final static int authenticationServerPort = 4447;

public final static DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

public static SecretKey K\_AS\_TGS;

public static SecretKey K\_TGS\_SS;

static {

try {

K\_AS\_TGS = KeyGenerator.getInstance("DES").generateKey();

K\_TGS\_SS = KeyGenerator.getInstance("DES").generateKey();

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

}

}

enum TGTComponents{

c,

tgs,

t1,

p1,

K\_C\_TGS

}

enum TGSComponents{

c,

ss,

t3,

p2,

K\_C\_SS

}

**DesEncrypter.java:**

package com.company;

import javax.crypto.\*;

import javax.crypto.spec.SecretKeySpec;

import java.io.\*;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.util.Base64;

class DesEncrypter {

Cipher ecipher;

Cipher dcipher;

public DesEncrypter(SecretKey key) {

try {

ecipher = Cipher.getInstance("DES");

dcipher = Cipher.getInstance("DES");

ecipher.init(Cipher.ENCRYPT\_MODE, key);

dcipher.init(Cipher.DECRYPT\_MODE, key);

} catch (NoSuchAlgorithmException | NoSuchPaddingException | InvalidKeyException e) {

e.printStackTrace();

}

}

public byte[] encrypt(String str){

byte[] utf8;

byte[] enc = new byte[0];

try {

utf8 = str.getBytes("UTF8");

enc = ecipher.doFinal(utf8);

} catch (UnsupportedEncodingException | IllegalBlockSizeException | BadPaddingException e) {

e.printStackTrace();

}

return Base64.getEncoder().encode(enc);

}

public byte[] encrypt(byte[] utf8){

byte[] enc = new byte[0];

try {

enc = ecipher.doFinal(utf8);

} catch (IllegalBlockSizeException | BadPaddingException e) {

e.printStackTrace();

}

return Base64.getEncoder().encode(enc);

}

public String decryptToStr(byte[] str) {

byte[] dec = Base64.getDecoder().decode(str);

byte[] utf8;

try {

utf8 = dcipher.doFinal(dec);

return new String(utf8, "UTF8");

} catch (IllegalBlockSizeException | BadPaddingException | UnsupportedEncodingException e) {

e.printStackTrace();

System.out.println(e);

return null;

}

}

public byte[] decrypt(byte[] str) {

byte[] dec = Base64.getDecoder().decode(str);

byte[] utf8;

try {

utf8 = dcipher.doFinal(dec);

return utf8;

} catch (IllegalBlockSizeException | BadPaddingException e) {

e.printStackTrace();

return null;

}

}

public static SecretKey getSK(String str)

{

byte[] decodedKey = Base64.getDecoder().decode(str);

return new SecretKeySpec(decodedKey, 0, decodedKey.length, "DES");

}

public static SecretKey generateSK()

{

try {

return KeyGenerator.getInstance("DES").generateKey();

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

return null;

}

}

}

**ServiceServer.java:**

package com.company;

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.time.LocalDateTime;

public class ServiceServer

{

private DatagramSocket socket;

private boolean running;

private byte[] buf = new byte[256];

private byte[] encryptedTGS;

private byte[] encryptedAut2;

private String[] TGS;

private String[] Aut2;

public ServiceServer()

{

try {

socket = new DatagramSocket(Constants.serviceServerPort);

} catch (SocketException e) {

e.printStackTrace();

}

}

public void run() {

running = true;

System.out.println("SS running");

while (running) {

DesEncrypter encrypterK\_C\_SS = null;

DatagramPacket packet = new DatagramPacket(buf, buf.length);

try {

socket.receive(packet);

System.out.println("C->SS");

} catch (IOException e) {

e.printStackTrace();

}

InetAddress address = packet.getAddress();

int port = packet.getPort();

packet = new DatagramPacket(buf, buf.length, address, port);

String received= new String(packet.getData(), 0, packet.getLength());

encryptedTGS = received.split("\n")[0].getBytes();

encryptedAut2 = received.split("\n")[1].getBytes();

System.out.println("SS get encrypted TGS,encrypted Aut2 from C");

DesEncrypter encrypterK\_TGS\_SS = new DesEncrypter(Constants.K\_TGS\_SS);

TGS = encrypterK\_TGS\_SS.decryptToStr(encryptedTGS).split("\n");

System.out.println("SS decrypt TGS:"+TGS);

try {

encrypterK\_C\_SS = new DesEncrypter(DesEncrypter.getSK(TGS[TGSComponents.K\_C\_SS.ordinal()]));

Aut2 = encrypterK\_C\_SS.decryptToStr(encryptedAut2).split("\n");

}

catch (Exception ex)

{

System.out.println("C used wrong K\_C\_SS");

System.exit(-1);

}

System.out.println("SS decrypt Aut2:"+Aut2);

LocalDateTime t3 = LocalDateTime.parse(TGS[TGSComponents.t3.ordinal()], Constants.formatter);

int p2 = Integer.parseInt(TGS[TGSComponents.p2.ordinal()]);

LocalDateTime t4 = LocalDateTime.parse(Aut2[1], Constants.formatter);

if(t4.isAfter(t3.plusMinutes(p2)))

{

System.out.println("TGS time limit is over");

System.exit(-1);

}

System.out.println("SS check TGS time limit");

System.out.println("SS->C");

String t4SS = t4.plusMinutes(1).format(Constants.formatter);

buf = encrypterK\_C\_SS.encrypt(t4SS);

packet = new DatagramPacket(buf, buf.length, address, port);

try {

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

System.out.println("SS send encrypted t4+1 to C");

}

socket.close();

}

}

**TicketGrantingServer.java:**

package com.company;

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.time.LocalDateTime;

import java.util.Base64;

import java.util.Objects;

public class TicketGrantingServer {

private DatagramSocket socket;

private boolean running;

private byte[] buf = new byte[256];

private byte[] encryptedTGT;

private byte[] encryptedAut1;

private String[] TGT;

private String[] Aut1;

private String TGS;//Ticket Granting Service - билет для доступа к SS {TGS} ={с,ss,t3,p2, KC\_SS }.

private String K\_C\_SS;//ключ для взаимодействия C и SS

private int p2 = 20;

public TicketGrantingServer()

{

try {

socket = new DatagramSocket(Constants.ticketGrantingServerPort);

} catch (SocketException e) {

e.printStackTrace();

}

}

public void run() {

running = true;

System.out.println("TGS running");

while (running) {

DatagramPacket packet = new DatagramPacket(buf, buf.length);

try {

socket.receive(packet);

} catch (IOException e) {

e.printStackTrace();

}

InetAddress address = packet.getAddress();

int port = packet.getPort();

String received= new String(packet.getData(), 0, packet.getLength());

encryptedTGT = received.split("\n")[0].getBytes();

encryptedAut1 = received.split("\n")[1].getBytes();

System.out.println("TGS get encrypted TGT,encrypted Aut1, ID from C");

DesEncrypter encrypterAS\_TGS = new DesEncrypter(Constants.K\_AS\_TGS);

TGT = encrypterAS\_TGS.decryptToStr(encryptedTGT).split("\n");

System.out.println("TGS decrypt TGT:"+TGT);

try {

DesEncrypter encrypterK\_C\_TGS = new DesEncrypter(DesEncrypter.getSK(TGT[TGTComponents.K\_C\_TGS.ordinal()]));

Aut1 = encrypterK\_C\_TGS.decryptToStr(encryptedAut1).split("\n");

}

catch (Exception ex)

{

System.out.println("C used wrong K\_C");

System.exit(-1);

}

System.out.println("TGS decrypt Aut1:"+Aut1);

LocalDateTime t1 = LocalDateTime.parse(TGT[TGTComponents.t1.ordinal()], Constants.formatter);

int p1 = Integer.parseInt(TGT[TGTComponents.p1.ordinal()]);

LocalDateTime t2 = LocalDateTime.parse(Aut1[1], Constants.formatter);

if(t2.isAfter(t1.plusMinutes(p1)))

{

System.out.println("TGT time limit is over");

System.exit(-1);

}

System.out.println("TGS check TGT time limit");

System.out.println("TGS->C");

K\_C\_SS = Base64.getEncoder().encodeToString(Objects.requireNonNull(DesEncrypter.generateSK()).getEncoded());

System.out.println("K\_C\_SS:"+ K\_C\_SS);

TGS = TGT[TGTComponents.c.ordinal()] + "\n";

TGS += Constants.serviceServerPort + "\n";

TGS += LocalDateTime.now().format(Constants.formatter) + "\n";

TGS += p2 + "\n";

TGS += K\_C\_SS;

System.out.println("TGS:"+ TGS);

DesEncrypter encrypterK\_TGS\_SS = new DesEncrypter(Constants.K\_TGS\_SS);

DesEncrypter encrypterK\_C\_TGS = new DesEncrypter(DesEncrypter.getSK(TGT[TGTComponents.K\_C\_TGS.ordinal()]));

buf = encrypterK\_C\_TGS.encrypt(encrypterK\_TGS\_SS.encrypt(TGS));

packet = new DatagramPacket(buf, buf.length, address, port);

try {

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

System.out.println("TGS encrypted by K\_TGS\_SS and K\_C\_TGS: "+new String(buf,0,buf.length));

buf = encrypterK\_C\_TGS.encrypt(K\_C\_SS);

packet = new DatagramPacket(buf, buf.length, address, port);

try {

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

System.out.println("K\_C\_SS encrypted by K\_C\_TGS: "+new String(buf,0,buf.length));

}

socket.close();

}

}