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 №3

на тему

**ИДЕНТИФИКАЦИЯ И АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ. ПРОТОКОЛ KERBEROS**

Выполнил:

студент гр. 253504

Носкович П.Н.

Проверил:

Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

[Введенеие 3](#_Toc157436533)

[1 Краткие теоретические сведения 4](#_Toc157436534)

[2 Результат выполнения программы 7](#_Toc157436535)

[Приложение А 10](#_Toc157436537)

# ВВЕДЕНИЕ

Цель данной лабораторной работы заключается в изучении теоретических сведений по работе протокола *Kerberos* и алгоритма *DES* и разработки программы, реализующее протокол распределения ключей *Kerberos*, включая процедуру, реализующую алгоритм *DES*.

# 1 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 имеющий возможности тонкой настройки под конкретные применения, существует в нескольких версиях. Мы рассмотрим упрощенный механизм аутентификации, реализованный с помощью протокола *Kerberos* версии 5.

В начале имеются три участника протокола *Kerberos*: клиент, сервис, центр распределения ключей.

Каждый из участников обладает своим долговременным секретом (ключом). Кроме того, центр распределения ключей обладает секретами всех участников.

Клиент отправляет серверу аутентификации запрос, содержащий: принципал клиента и срок жизни билета.

Сервер аутентификации по полученному принципалу находит в базе *Kerberos* секрет клиента. Кроме того, для дальнейшего общения с *KDC* сервер аутентификации случайным образом генерирует сессионный ключ. В итоге в ответ клиенту отправляются два сообщения. Первое сообщение зашифровано с использованием секрета клиента и содержит: сессионный ключ для *KDC*, метка времени, срок жизни *TGT*. Второе сообщение (*TGT*) зашифровано уже с использованием секрета *KDC* и включает в себя те же самые данные, что и первое сообщение, но вместе с принципалом клиента.

Клиент, приняв ответ, может расшифровать только первое сообщение. Таким образом он получает сессионный ключ для дальнейшего общения с *KDC. TGT* также сохраняется у клиента в зашифрованном виде.

Теперь, пройдя аутентификацию, клиент желает получить доступ к какому-то сервису. Для этого он отправляет серверу выдачи разрешений запрос, содержащий: принципал сервиса, аутентификатор, состоящий из принципала клиента и метки времени, зашифрованных с использованием извлеченного ранее сессионного ключа для общения с *KDC*, сохраненный *TGT*.

Приняв запрос, сервер выдачи разрешений прежде всего выполняет проверку полученных данных. Сначала с использованием секрета *KDC* сервер расшифровывает *TGT* и по метке времени со сроком действия убеждается, что *TGT* не протух.

Далее сервер извлекает сессионный ключ для *KDC*. Несмотря на то, что указанный ключ был создан в *KDC*, нужды хранить его в базе Kerberos нет. Действительно, *TGT* не может быть изменен кем-либо кроме KDC, поэтому полученным из него данным можно доверять.

В случае успешного завершения проверок сервер выдачи разрешений отправляет клиенту ответ, содержащий два сообщения. Первое сообщение зашифровано с использованием сессионного ключа для *KDC* и содержит: сессионный ключ для общения с сервисом, метка времени, срок жизни *TGS* билета, принципал сервиса. Второе сообщение (*TGS* билет) зашифровано с использованием секрета сервиса и включает в себя те же самые данные, что и первое сообщение, а также принципал клиента. Клиент, приняв ответ, может расшифровать только первое сообщение. Таким образом он получает сессионный ключ для дальнейшего общения с сервисом. *TGS* билет сохраняется у клиента в зашифрованном виде.

Клиент отправляет сервису запрос на получение доступа, содержащий: аутентификатор, состоящий из принципала клиента и метки времени, зашифрованных с использованием извлеченного ранее сессионного ключа для общения с сервисом, сохраненный *TGS* билет, флаг взаимной аутентификации. Приняв запрос, сервис прежде всего выполняет проверку полученных данных. Сначала с использованием своего секрета сервис расшифровывает *TGS* и по метке времени со сроком действия убеждается, что *TGS* не протух. Далее сервис извлекает сессионный ключ.

*TGS* билет не может быть изменен кем-либо кроме того, кто знает секрет сервиса, а это *KDC* и сам сервис. Сервис доверяет *KDC*, таким образом извлеченным из *TGS* билета данным сервис также может доверять.

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.

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.

Затем 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*. 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.*

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.

# 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*.

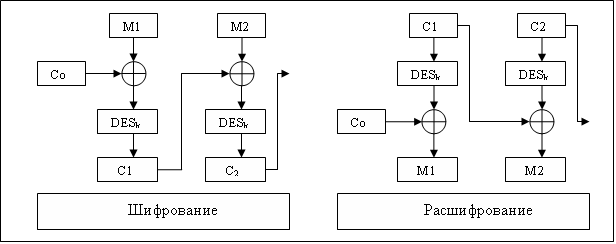


Рисунок 2.2 – Блок-схема алгоритма *DES*

Результат выполнения программы изображён на рисунке 2.3.

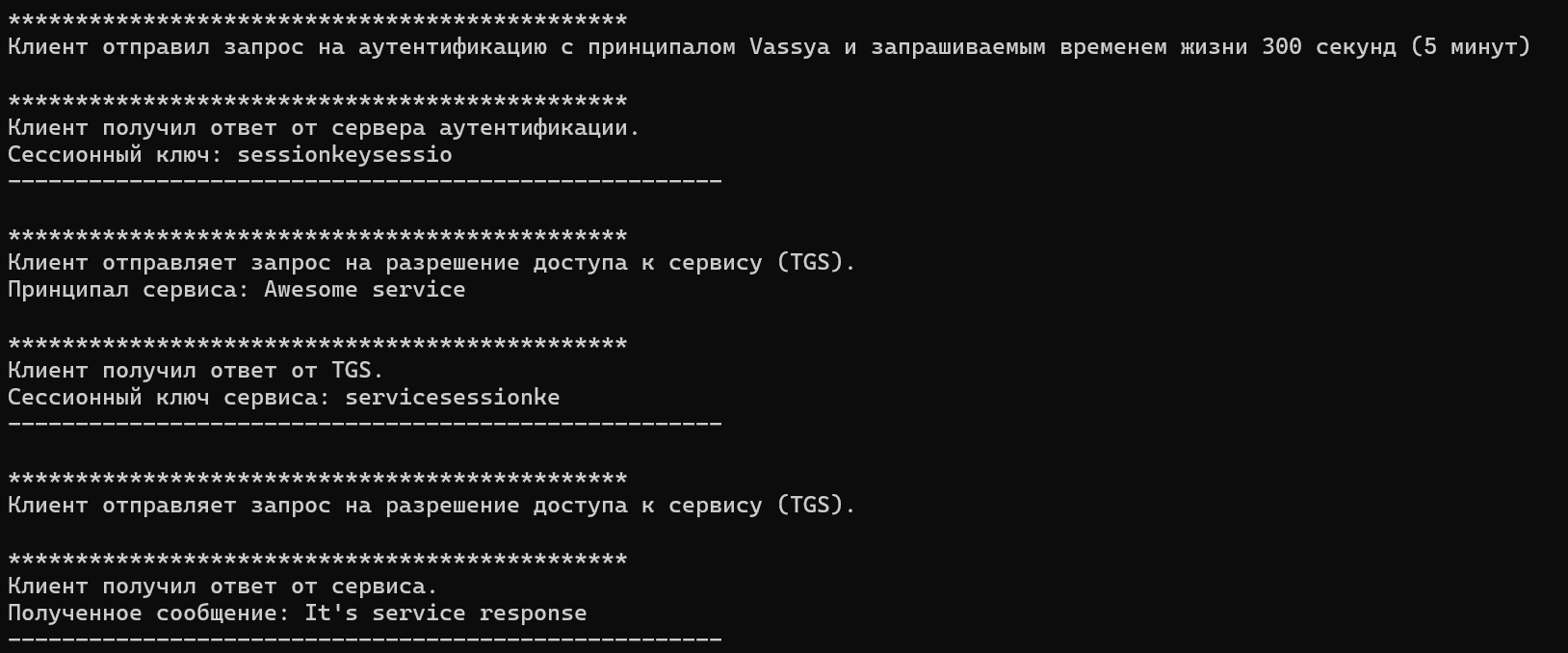


Рисунок 2.3 – Результат программы

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

main.py

import random

from datetime import datetime, timedelta

from Crypto.Cipher import DES

# Генерация случайного ключа DES (8 байт)

def generate\_des\_key():

    return random.getrandbits(64).to\_bytes(8, byteorder='big')

# Функция шифрования

def des\_encrypt(key, data):

    cipher = DES.new(key, DES.MODE\_ECB)

    return cipher.encrypt(data)

# Функция дешифрования

def des\_decrypt(key, data):

    cipher = DES.new(key, DES.MODE\_ECB)

    return cipher.decrypt(data)

# Заполнение данных до кратного размера блока (64 бита)

def pad(data):

    pad\_len = 8 - (len(data) % 8)

    return data + bytes([pad\_len] \* pad\_len)

# Удаление заполнения

def unpad(data):

    pad\_len = data[-1]

    return data[:-pad\_len]

# Центр распределения ключей (KDC)

class KDC:

    def \_\_init\_\_(self):

        self.secrets = {}

    def register(self, principal, secret):

        self.secrets[principal] = secret

    def generate\_ticket(self, client, service, session\_key, lifetime):

        ticket\_data = {

            'session\_key': session\_key.hex(),

            'client': client,

            'service': service,

            'lifetime': lifetime.isoformat()

        }

        ticket\_str = str(ticket\_data).encode()

        service\_secret = self.secrets[service]

        return des\_encrypt(service\_secret, pad(ticket\_str))

    def authenticate(self, client, lifetime):

        client\_secret = self.secrets[client]

        session\_key = generate\_des\_key()

        tgt\_data = {

            'session\_key': session\_key.hex(),

            'client': client,

            'lifetime': lifetime.isoformat()

        }

        tgt\_str = str(tgt\_data).encode()

        tgt = des\_encrypt(self.secrets['kdc'], pad(tgt\_str))

        encrypted\_response = des\_encrypt(client\_secret, pad(session\_key + lifetime.isoformat().encode()))

        return encrypted\_response, tgt

# Инициализация KDC

kdc = KDC()

kdc.register('client', generate\_des\_key())

kdc.register('service', generate\_des\_key())

kdc.register('kdc', generate\_des\_key())

# Запрос ввода от пользователя

client\_input = input("Введите имя клиента (client): ").strip()

service\_input = input("Введите имя сервиса (service): ").strip()

lifetime\_input = input("Введите срок действия билета в часах: ").strip()

# Проверка ввода

if client\_input not in kdc.secrets:

    raise ValueError("Ошибка: Клиент не зарегистрирован в системе.")

if service\_input not in kdc.secrets:

    raise ValueError("Ошибка: Сервис не зарегистрирован в системе.")

if not lifetime\_input.isdigit() or int(lifetime\_input) <= 0:

    raise ValueError("Ошибка: Срок действия билета должен быть положительным числом.")

# Преобразование срока действия билета

lifetime = datetime.now() + timedelta(hours=int(lifetime\_input))

# Аутентификация клиента

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(f"Клиент отправил запрос на аутентификацию с принципалом {client\_input} и запрашиваемым временем жизни {int(lifetime\_input) \* 3600} секунд ({lifetime\_input} часов)")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

response, tgt = kdc.authenticate(client\_input, lifetime)

# Дешифруем ответ для извлечения session\_key

client\_secret = kdc.secrets[client\_input]

decrypted\_response = des\_decrypt(client\_secret, response)

session\_key = decrypted\_response[:8]

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Клиент получил ответ от сервера аутентификации.")

print(f"Сессионный ключ: {session\_key.hex()}\n")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

# Запрос к серверу авторизации

authenticator = {

    'client': client\_input,

    'timestamp': datetime.now().isoformat()

}

authenticator\_str = str(authenticator).encode()

encrypted\_authenticator = des\_encrypt(session\_key, pad(authenticator\_str))

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(f"Клиент отправляет запрос на разрешение доступа к сервису (TGS).\nПринципал сервиса: {service\_input}")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

# Сервер авторизации

service\_ticket = kdc.generate\_ticket(client\_input, service\_input, session\_key, lifetime)

decrypted\_ticket = des\_decrypt(kdc.secrets[service\_input], service\_ticket)

unpadded\_ticket = unpad(decrypted\_ticket)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Клиент получил ответ от TGS.")

print(f"Сессионный ключ сервиса: {session\_key.hex()}\n")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

# Клиент отправляет запрос сервису

service\_authenticator = des\_encrypt(session\_key, pad(authenticator\_str))

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(f"Клиент отправляет запрос на разрешение доступа к сервису (TGS).")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

# Проверка на стороне сервиса

decrypted\_service\_authenticator = des\_decrypt(session\_key, service\_authenticator)

unpadded\_authenticator = unpad(decrypted\_service\_authenticator)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Клиент получил ответ от сервиса.")

print("Полученное сообщение: It's service response\n")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")