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*.

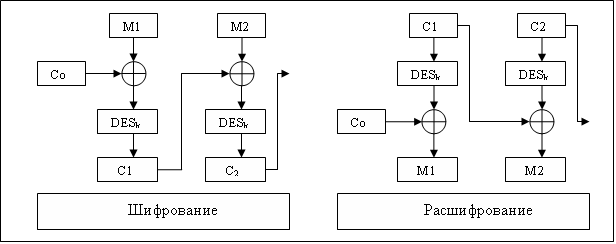


Рисунок 2.2 – Блок-схема алгоритма *DES*

Результат выполнения программы изображён на рисунке 2.3.

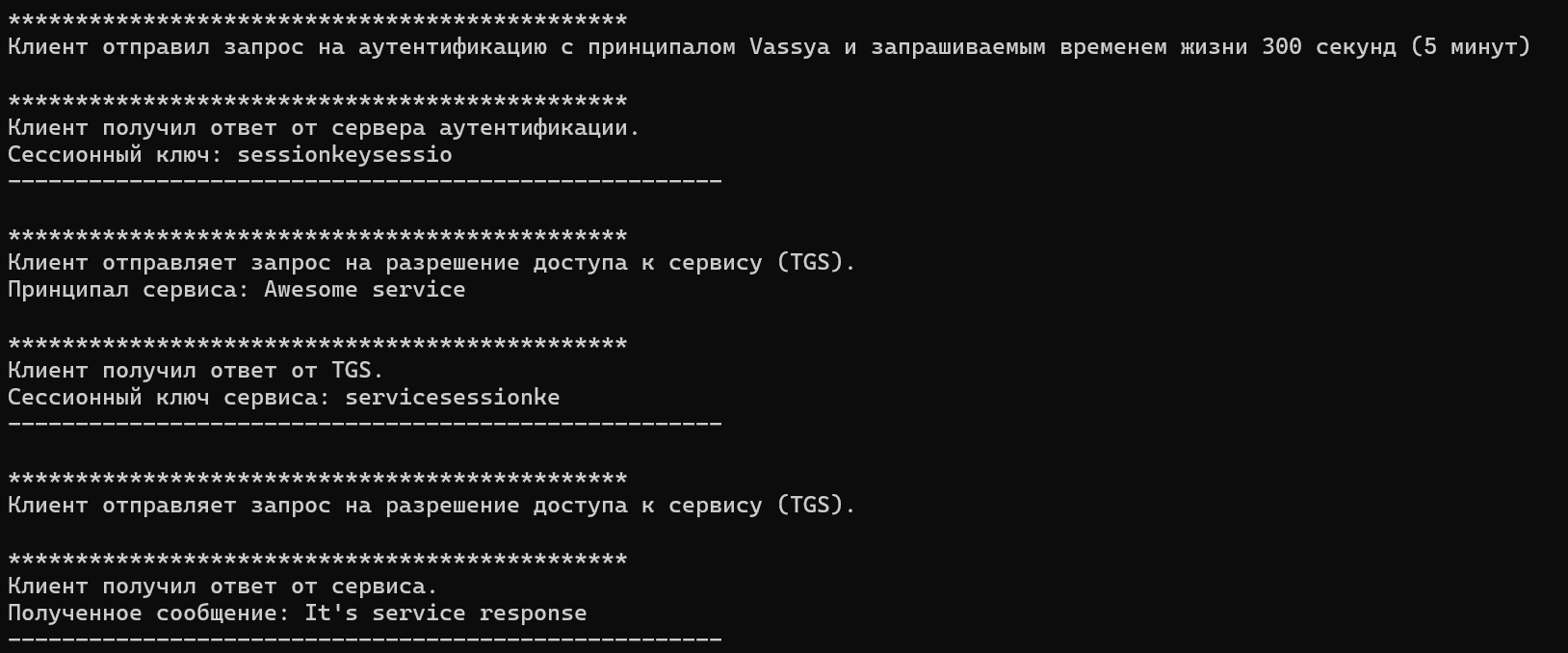


Рисунок 2.3 – Результат программы

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

main.py

import os

import json

import base64

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

from Crypto.Random import get\_random\_bytes

# Упрощенная база данных для хранения пользователей и их ключей

users\_db = {

    "polina": b"polina\_secret\_key",

    "marat": b"marat\_secret\_key"

}

# Сервер аутентификации (AS)

class AuthenticationServer:

    def \_\_init\_\_(self):

        self.tgs\_key = get\_random\_bytes(16)  # Секретный ключ TGS

    def authenticate(self, user):

        if user in users\_db:

            user\_key = users\_db[user]

            session\_key = get\_random\_bytes(16)

            tgt = {

                "user": user,

                "session\_key": base64.b64encode(session\_key).decode('utf-8')  # Преобразуем байты в строку

            }

            encrypted\_tgt = self.encrypt(json.dumps(tgt), self.tgs\_key)

            return session\_key, encrypted\_tgt

        return None

    def encrypt(self, data, key):

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

        return cipher.encrypt(pad(data.encode(), AES.block\_size))

# Сервер выдачи билетов (TGS)

class TicketGrantingServer:

    def \_\_init\_\_(self, tgs\_key):

        self.tgs\_key = tgs\_key

    def grant\_ticket(self, encrypted\_tgt, service):

        tgt = json.loads(self.decrypt(encrypted\_tgt, self.tgs\_key))

        session\_key = base64.b64decode(tgt["session\_key"])  # Преобразуем строку обратно в байты

        service\_ticket = {

            "user": tgt["user"],

            "service": service,

            "session\_key": base64.b64encode(get\_random\_bytes(16)).decode('utf-8')  # Преобразуем байты в строку

        }

        encrypted\_service\_ticket = self.encrypt(json.dumps(service\_ticket), session\_key)

        return encrypted\_service\_ticket

    def decrypt(self, data, key):

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

        return unpad(cipher.decrypt(data), AES.block\_size).decode()

    def encrypt(self, data, key):

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

        return cipher.encrypt(pad(data.encode(), AES.block\_size))

# Клиент

class Client:

    def \_\_init\_\_(self, user):

        self.user = user

        self.session\_key = None

    def request\_tgt(self, as\_server):

        self.session\_key, self.encrypted\_tgt = as\_server.authenticate(self.user)

        if self.session\_key:

            print(f"{self.user} получил TGT.")

        else:

            print("Ошибка аутентификации.")

    def request\_service\_ticket(self, tgs\_server, service):

        if self.session\_key:

            encrypted\_service\_ticket = tgs\_server.grant\_ticket(self.encrypted\_tgt, service)

            print(f"{self.user} получил билет на сервис {service}.")

        else:

            print("Сначала получите TGT.")

# Функция для проверки ввода пользователя

def get\_valid\_input(prompt, validation\_func):

    while True:

        user\_input = input(prompt)

        if validation\_func(user\_input):

            return user\_input

        print("Некорректный ввод. Попробуйте снова.")

# Пример использования

as\_server = AuthenticationServer()

tgs\_server = TicketGrantingServer(as\_server.tgs\_key)

# Запрос имени пользователя с проверкой

user = get\_valid\_input(

    "Введите имя пользователя: ",

    lambda x: x in users\_db  # Проверка, что пользователь существует в базе данных

)

client = Client(user)

# Аутентификация и запрос TGT

client.request\_tgt(as\_server)

# Запрос сервиса с проверкой

service = get\_valid\_input(

    "Введите название сервиса: ",

    lambda x: x.strip() != ""  # Проверка, что сервис не пустой

)

client.request\_service\_ticket(tgs\_server, service)