Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ

к лабораторной работе №3

на тему

**ПРОТОКОЛ KERBEROS**

Выполнил: студент гр.253504

Фроленко К.Ю.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

[1  Формулировака задачи 3](#_Toc189566878)

[2 Ход работы 4](#_Toc189566879)

[Заключение 5](#_Toc189566880)

1 ФОРМУЛИРОВКА ЗАДАЧИ

В данной работе требуется разработать программное средство, реализующее протокол *Kerberos* для обеспечения безопасной аутентификации в клиент-серверной архитектуре. Программа должна демонстрировать ключевые этапы обмена данными между клиентом и сервером, который объединяет функции сервера аутентификации (*AS*), сервера выдачи билетов (*TGS*) и сервисного сервера. Основной целью является показ принципов работы *Kerberos*, где клиент запрашивает аутентификацию у *AS*, получает зашифрованный *Ticket Granting Ticket* (*TGT*), затем с помощью *TGT* обращается к *TGS* для получения билета для доступа к конкретному сервису, и, наконец, представляется этот билет сервисному серверу для подтверждения своей личности.

Реализация должна включать обмен зашифрованными сообщениями, используя упрощённые алгоритмы симметричного шифрования для демонстрации процессов шифрования и дешифрования данных. Программа должна обеспечивать возможность запуска сервера как отдельного сетевого процесса, который принимает запросы от клиента через сокеты, обрабатывая их в соответствии с этапами протокола. Клиентская часть, в свою очередь, должна последовательно выполнять запросы к различным компонентам сервера, выводя полученные сеансовые ключи и подтверждение аутентификации, а также отображать отладочную информацию для контроля хода выполнения операций.

Таким образом, итоговое решение должно не только демонстрировать корректное выполнение механизма *Kerberos*, но и предоставлять удобный способ отслеживания процесса обмена зашифрованными данными между клиентом и сервером, обеспечивая понимание основных принципов аутентификации в распределённых системах.

2 ХОД РАБОТЫ

При запуске серверного модуля в терминале выводятся отладочные сообщения, демонстрирующие выполнение ключевых этапов протокола *Kerberos*. Сервер, объединяющий функции аутентификации (*AS*), выдачи билетов (*TGS*) и подтверждения доступа сервису, генерирует сеансовые ключи, формирует *Ticket Granting Ticket* (*TGT*) и сервисный билет, а также выводит информацию о каждом из этих этапов.

Запуск клиентской части инициирует последовательную обработку запросов. Вначале клиент отправляет запрос к серверу *AS*, указывая свой идентификатор (например, «*client1*»). Сервер *AS*, используя предопределённый общий секрет, генерирует сеансовый ключ для связи с *TGS* и формирует *TGT*, который шифруется и возвращается клиенту. Клиент расшифровывает полученное сообщение с помощью своего секретного ключа, извлекает сеансовый ключ для дальнейшего обмена с сервером *TGS* и сохраняет полученный *TGT*.

Затем клиент обращается к серверу *TGS*, передавая полученный *TGT* и идентификатор требуемого сервиса (например, «*fileserver*»). Сервер *TGS* дешифрует *TGT*, извлекает идентификатор клиента и исходный сеансовый ключ, генерирует новый сеансовый ключ для связи с сервисным сервером и формирует зашифрованный сервисный билет. Ответ от *TGS* шифруется с использованием сеансового ключа, полученного на предыдущем этапе, и возвращается клиенту. Клиент, расшифровывая сообщение, получает новый сеансовый ключ для сервиса и сервисный билет.

Заключительным этапом является отправка клиентом запроса к сервисному серверу с использованием полученного сервисного билета и нового сеансового ключа. Сервисный сервер, расшифровывая билет, проверяет подлинность клиента и возвращает подтверждение, содержащее идентификатор клиента. Этот результат свидетельствует о корректном выполнении всех этапов обмена зашифрованными сообщениями согласно протоколу *Kerberos*. На рисунке 1 представлен пример работы программы.

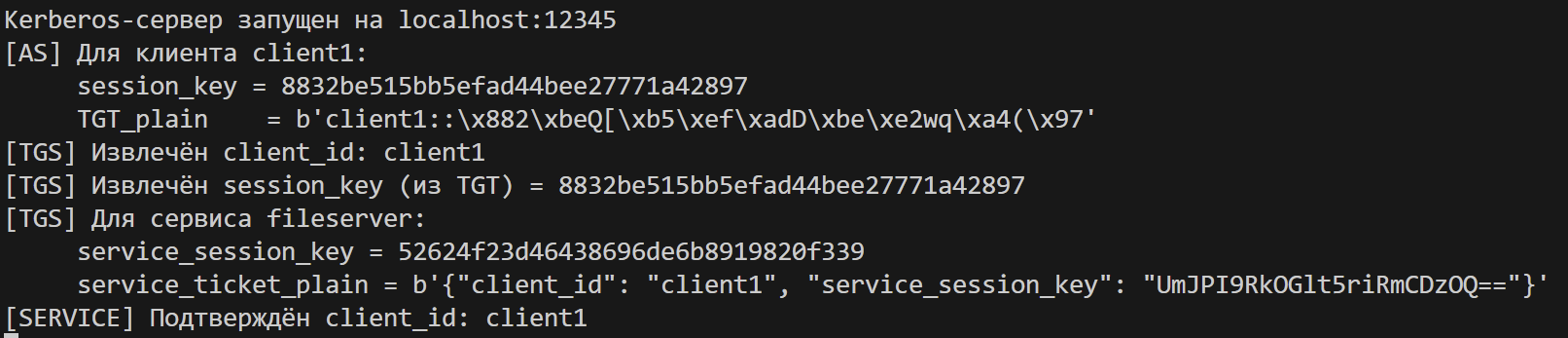


Рисунок 1 – Пример работы программы

# ЗАКЛЮЧЕНИЕ

В ходе данной работы было разработано программное средство, реализующее протокол *Kerberos* для обеспечения безопасной аутентификации в клиент-серверной архитектуре. Программа продемонстрировала ключевые этапы обмена зашифрованными сообщениями между клиентом и сервером, который объединяет функции сервера аутентификации (*AS*), сервера выдачи билетов (*TGS*) и сервисного сервера.

На начальном этапе была определена архитектура системы, выбран упрощённый алгоритм симметричного шифрования, позволяющий моделировать процессы шифрования и дешифрования данных на каждом из этапов аутентификации. Реализация обеспечила корректный обмен информацией: клиент отправляет запрос к серверу *AS*, получает зашифрованный *Ticket Granting Ticket* (*TGT*) и сеансовый ключ, затем с помощью *TGT* обращается к серверу *TGS* для получения сервисного билета, и, наконец, представляет билет сервисному серверу для проверки подлинности.

Представленный пример работы программы подтверждает, что система корректно формирует сеансовые ключи и билеты, а также выполняет проверку аутентификации клиента. Отладочные сообщения, выводимые сервером, позволяют наглядно проследить процесс формирования *TGT* и сервисного билета, что свидетельствует о надёжности и эффективности реализации протокола *Kerberos*.

Таким образом, поставленная задача была успешно решена. Разработанное средство демонстрирует основные принципы работы *Kerberos* и может служить основой для дальнейших исследований и развития механизмов безопасной аутентификации в распределённых системах.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг программного кода**

Листинг А.1 – файл *client.py*

import socket

import json

import base64

from server import simple\_encrypt, simple\_decrypt

USER\_ID = "client1"

USER\_KEY = b"client1secret"

HOST, PORT = "localhost", 12345

def send\_request(action, data):

req = {"action": action, "data": data}

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect((HOST, PORT))

s.sendall(json.dumps(req).encode())

resp = s.recv(4096)

s.close()

return json.loads(resp.decode())

def client\_flow():

as\_req = {"client\_id": USER\_ID}

as\_resp = send\_request("AS", as\_req)

if "error" in as\_resp:

print("AS error:", as\_resp["error"])

return

as\_data\_encrypted = base64.b64decode(as\_resp["response"].encode())

message\_plain = simple\_decrypt(USER\_KEY, as\_data\_encrypted)

as\_msg = json.loads(message\_plain.decode())

session\_key = base64.b64decode(as\_msg["session\_key"].encode())

tgt\_encrypted = as\_msg["tgt"]

print("Получен сеансовый ключ для TGS:", session\_key.hex())

service\_id = "fileserver"

authenticator = "dummy"

tgs\_req = {

"tgt": tgt\_encrypted,

"service\_id": service\_id,

"authenticator": base64.b64encode(authenticator.encode()).decode(),

}

tgs\_resp = send\_request("TGS", tgs\_req)

if "error" in tgs\_resp:

print("TGS error:", tgs\_resp["error"])

return

tgs\_data\_encrypted = base64.b64decode(tgs\_resp["response"].encode())

tgs\_message\_plain = simple\_decrypt(session\_key, tgs\_data\_encrypted)

tgs\_msg = json.loads(tgs\_message\_plain.decode())

service\_session\_key = base64.b64decode(tgs\_msg["service\_session\_key"].encode())

service\_ticket = tgs\_msg["service\_ticket"]

print("Получен сеансовый ключ для сервиса:", service\_session\_key.hex())

service\_authenticator = "dummy\_service"

svc\_req = {

"service\_ticket": service\_ticket,

"authenticator": base64.b64encode(service\_authenticator.encode()).decode(),

"service\_session\_key": base64.b64encode(service\_session\_key).decode(),

}

svc\_resp = send\_request("SERVICE", svc\_req)

if "error" in svc\_resp:

print("SERVICE error:", svc\_resp["error"])

return

print("Сервер подтвердил клиента:", svc\_resp["client\_id"])

if \_\_name\_\_ == "\_\_main\_\_":

client\_flow()

Листинг А.2 – файл *server.py*

import os

import json

import base64

import socketserver

def simple\_encrypt(key: bytes, message: bytes) -> bytes:

rep = (key \* ((len(message) // len(key)) + 1))[:len(message)]

return bytes(a ^ b for a, b in zip(message, rep))

def simple\_decrypt(key: bytes, ciphertext: bytes) -> bytes:

return simple\_encrypt(key, ciphertext)

USER\_SECRETS = {"client1": b"client1secret"}

AS\_TGS\_KEY = b"as\_tgs\_shared"

TGS\_SERVICE\_KEY = b"tgs\_service\_shared"

def generate\_session\_key() -> bytes:

return os.urandom(16)

def as\_process\_authentication\_request(client\_id: str) -> bytes:

if client\_id not in USER\_SECRETS:

raise Exception("Unknown client")

session\_key = generate\_session\_key()

tgt\_plain = client\_id.encode() + b"::" + session\_key

tgt\_encrypted = simple\_encrypt(AS\_TGS\_KEY, tgt\_plain)

msg = {

"session\_key": base64.b64encode(session\_key).decode(),

"tgt": base64.b64encode(tgt\_encrypted).decode(),

}

message\_plain = json.dumps(msg).encode()

client\_key = USER\_SECRETS[client\_id]

message\_encrypted = simple\_encrypt(client\_key, message\_plain)

print(f"[AS] Для клиента {client\_id}:")

print(f" session\_key = {session\_key.hex()}")

print(f" TGT\_plain = {tgt\_plain}")

return message\_encrypted

def tgs\_process\_service\_request(

tgt\_encrypted\_b64: str, service\_id: str, authenticator\_encrypted\_b64: str

) -> bytes:

tgt\_encrypted = base64.b64decode(tgt\_encrypted\_b64.encode())

tgt\_plain = simple\_decrypt(AS\_TGS\_KEY, tgt\_encrypted)

try:

client\_id\_bytes, session\_key = tgt\_plain.split(b"::", 1)

except Exception as e:

raise Exception("Invalid TGT format") from e

client\_id = client\_id\_bytes.decode()

print(f"[TGS] Извлечён client\_id: {client\_id}")

print(f"[TGS] Извлечён session\_key (из TGT) = {session\_key.hex()}")

service\_session\_key = generate\_session\_key()

service\_ticket\_plain = json.dumps({

"client\_id": client\_id,

"service\_session\_key": base64.b64encode(service\_session\_key).decode()

}).encode()

service\_ticket\_encrypted = simple\_encrypt(TGS\_SERVICE\_KEY, service\_ticket\_plain)

msg = {

"service\_session\_key": base64.b64encode(service\_session\_key).decode(),

"service\_ticket": base64.b64encode(service\_ticket\_encrypted).decode(),

}

message\_plain = json.dumps(msg).encode()

message\_encrypted = simple\_encrypt(session\_key, message\_plain)

print(f"[TGS] Для сервиса {service\_id}:")

print(f" service\_session\_key = {service\_session\_key.hex()}")

print(f" service\_ticket\_plain = {service\_ticket\_plain}")

return message\_encrypted

def service\_process\_client\_request(

service\_ticket\_encrypted\_b64: str, authenticator\_encrypted\_b64: str, service\_session\_key\_b64: str

) -> str:

service\_ticket\_encrypted = base64.b64decode(service\_ticket\_encrypted\_b64.encode())

service\_session\_key = base64.b64decode(service\_session\_key\_b64.encode())

service\_ticket\_plain = simple\_decrypt(TGS\_SERVICE\_KEY, service\_ticket\_encrypted)

try:

data = json.loads(service\_ticket\_plain.decode())

except Exception as e:

raise Exception("Invalid service ticket format") from e

client\_id = data.get("client\_id")

ticket\_service\_session\_key\_b64 = data.get("service\_session\_key")

if not client\_id or not ticket\_service\_session\_key\_b64:

raise Exception("Invalid service ticket content")

ticket\_service\_session\_key = base64.b64decode(ticket\_service\_session\_key\_b64.encode())

if ticket\_service\_session\_key != service\_session\_key:

raise Exception("Invalid service session key!")

print(f"[SERVICE] Подтверждён client\_id: {client\_id}")

return client\_id

class KerberosRequestHandler(socketserver.BaseRequestHandler):

def handle(self):

data = self.request.recv(4096)

if not data:

return

req = json.loads(data.decode())

action = req.get("action")

payload = req.get("data")

resp = {}

try:

if action == "AS":

client\_id = payload["client\_id"]

result = as\_process\_authentication\_request(client\_id)

resp["response"] = base64.b64encode(result).decode()

elif action == "TGS":

tgt = payload["tgt"]

service\_id = payload["service\_id"]

authenticator = payload["authenticator"]

result = tgs\_process\_service\_request(tgt, service\_id, authenticator)

resp["response"] = base64.b64encode(result).decode()

elif action == "SERVICE":

service\_ticket = payload["service\_ticket"]

authenticator = payload["authenticator"]

service\_session\_key = payload["service\_session\_key"]

client\_id = service\_process\_client\_request(service\_ticket, authenticator, service\_session\_key)

resp["client\_id"] = client\_id

else:

resp["error"] = "Unknown action"

except Exception as e:

resp["error"] = str(e)

print(f"[ERROR] При обработке {action}: {e}")

self.request.sendall(json.dumps(resp).encode())

if \_\_name\_\_ == "\_\_main\_\_":

HOST, PORT = "localhost", 12345

server = socketserver.ThreadingTCPServer((HOST, PORT), KerberosRequestHandler)

print(f"Kerberos-сервер запущен на {HOST}:{PORT}")

server.serve\_forever()