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

**ИДЕНТИФИКАЦИЯ И АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ. ПРОТОКОЛ KERBEROS**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Герчик А.В. |

Минск 2025

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc189743996)

[2 Реализация программного средства 4](#_Toc189743997)

[Заключение 6](#_Toc189743998)

[Список использованных источников 7](#_Toc189743999)

[Приложение А (обязательное) Исходный код программы 8](#_Toc189744000)

**1 ПОСТАНОВКА ЗАДАЧИ**

Постановка задачи для данной лабораторной работы заключается в разработке и реализации упрощенной модели протокола аутентификации *Kerberos* на языке программирования *Python*. Протокол *Kerberos* является одним из наиболее широко используемых механизмов аутентификации в распределенных системах, обеспечивающим безопасное взаимодействие между клиентами и серверами в сети [1]. Цель работы заключается в изучении принципов работы протокола, его основных компонентов и этапов, а также в практической реализации этих этапов с использованием криптографических методов.

В рамках работы необходимо разработать три основных компонента системы: сервер аутентификации (*Authentication* *Server*, *AS*), сервер выдачи билетов (*Ticket* *Granting* *Server*, *TGS*) и сервер приложений (*Application* *Server*, *AP*). Сервер аутентификации отвечает за начальную аутентификацию клиента и выдачу билета на получение билетов (*Ticket* *Granting* *Ticket*, *TGT*). Сервер выдачи билетов предоставляет клиенту билет для доступа к конкретному сервису (*Service* *Ticket*), а сервер приложений проверяет подлинность клиента на основе полученного билета и предоставляет доступ к запрашиваемому сервису.

Для реализации протокола необходимо использовать симметричное шифрование с использованием библиотеки «*cryptography*» [2], а также обеспечить генерацию и управление ключами, включая ключи клиентов, серверов и сессионные ключи. Важным аспектом работы является реализация механизмов проверки временных меток для предотвращения атак с использованием повторной передачи данных. Также необходимо обеспечить корректную сериализацию и десериализацию данных при передаче между компонентами системы, включая преобразование бинарных данных в строки для их передачи в формате *JSON*.

В результате выполнения работы должен быть создан программный код, демонстрирующий все этапы работы протокола *Kerberos*: регистрацию клиента и сервиса, аутентификацию клиента, получение *TGT*, запрос и выдачу сервисного билета, а также доступ к сервису с проверкой подлинности клиента. Код должен быть документирован, а в отчете должны быть описаны основные этапы работы, принятые решения и результаты тестирования системы.

**2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО СРЕДСТВА**

Реализация программного средства для моделирования протокола *Kerberos* включает в себя разработку трех основных компонентов: сервера аутентификации (*Authentication* *Server*, *AS*), сервера выдачи билетов (*Ticket* *Granting* *Server*, *TGS*) и сервера приложений (*Application* *Server*, *AP*). Каждый из этих компонентов выполняет определенные функции, обеспечивающие безопасную аутентификацию и авторизацию клиента в системе. В качестве языка программирования был выбран *Python*, а для реализации криптографических функций использовалась библиотека *cryptography*.

Сервер аутентификации отвечает за начальную аутентификацию клиента и выдачу билета на получение билетов (*Ticket* *Granting* *Ticket*, *TGT*). При регистрации клиента на сервере создается запись, содержащая идентификатор клиента и ключ, сгенерированный на основе пароля. Для аутентификации клиент отправляет запрос на сервер, который проверяет наличие клиента в базе данных и генерирует сессионный ключ. Сессионный ключ и *TGT* шифруются с использованием ключа клиента и ключа *TGS* соответственно. *TGT* содержит информацию о клиенте, сессионном ключе, временной метке и сроке действия. Сервер возвращает зашифрованный ответ клиенту, который может использовать *TGT* для дальнейшего взаимодействия с системой.

Сервер выдачи билетов предоставляет клиенту билет для доступа к конкретному сервису (*Service* *Ticket*). Для получения билета клиент отправляет *TGT* и аутентификатор, который содержит идентификатор клиента и временную метку, зашифрованные сессионным ключом. *TGS* расшифровывает *TGT* с использованием своего ключа, проверяет подлинность клиента и срок действия *TGT*. Если проверка прошла успешно, *TGS* генерирует новый сессионный ключ для взаимодействия клиента с сервером приложений и создает сервисный билет, который шифруется с использованием ключа сервера приложений. Сервисный билет и новый сессионный ключ возвращаются клиенту в зашифрованном виде.

Сервер приложений проверяет подлинность клиента на основе полученного сервисного билета и предоставляет доступ к запрашиваемому сервису. Клиент отправляет сервисный билет и аутентификатор, зашифрованные новым сессионным ключом. Сервер приложений расшифровывает сервисный билет с использованием своего ключа, извлекает сессионный ключ и проверяет аутентификатор. Если проверка прошла успешно, сервер предоставляет доступ к сервису.

Для реализации криптографических функций использовалась библиотека *cryptography*, которая предоставляет удобный интерфейс для симметричного шифрования. Ключи генерируются с использованием функции *Fernet.generate\_key()*, а шифрование и дешифрование данных выполняются с помощью объектов *Fernet*. Для обеспечения безопасности ключи клиентов и серверов хранятся в зашифрованном виде, а сессионные ключи генерируются динамически для каждой сессии.

Для передачи данных между компонентами системы используется формат *JSON*. Поскольку *JSON* не поддерживает бинарные данные, все бинарные объекты (например, ключи и зашифрованные данные) преобразуются в строки с использованием кодировки *Base64*. Это позволяет корректно сериализовать и десериализовать данные при передаче между клиентом и серверами.

Тестирование программного средства проводилось на локальной машине с использованием заранее зарегистрированных клиентов и сервисов. Проверялась корректность работы всех этапов протокола *Kerberos*, включая аутентификацию клиента, получение *TGT*, запрос и выдачу сервисного билета, а также доступ к сервису. В процессе тестирования были выявлены и исправлены ошибки, связанные с сериализацией данных и проверкой временных меток.

В результате выполнения работы было разработано программное средство, которое успешно моделирует основные этапы протокола *Kerberos*. Программа демонстрирует работу всех компонентов системы и обеспечивает безопасную аутентификацию и авторизацию клиента в распределенной среде.

Код программного средства предоставлен в приложении А.

**ЗАКЛЮЧЕНИЕ**

В результате разработки программного средства для моделирования протокола *Kerberos* была успешно реализована функциональность трех ключевых компонентов: сервера аутентификации, сервера выдачи билетов и сервера приложений. Каждый из этих компонентов выполняет критически важные функции, обеспечивающие безопасную аутентификацию и авторизацию пользователей в распределенной среде. Сервер аутентификации отвечает за начальную проверку идентификации клиента, выдавая билет на получение билетов (*TGT*), который позволяет клиенту взаимодействовать с другими сервисами без повторной аутентификации. Это значительно упрощает процесс для пользователей и повышает общую безопасность системы.

Использование языка *Python* и библиотеки *cryptography* обеспечило надежную реализацию криптографических функций, таких как шифрование и дешифрование сессионных ключей и билетов. Библиотека предоставляет удобный интерфейс для управления ключами и шифрования, что позволяет сосредоточиться на логике реализации протокола, а не на низкоуровневых аспектах криптографии. Генерация ключей с использованием функции *Fernet.generate\_key()* и шифрование данных с помощью объектов *Fernet* гарантируют высокий уровень безопасности и защиту от несанкционированного доступа.

Все этапы аутентификации и авторизации были тщательно протестированы на локальной машине с заранее зарегистрированными клиентами и сервисами. В процессе тестирования проверялась корректность работы всех элементов протокола *Kerberos*, включая аутентификацию клиента, получение *TGT*, запрос и выдачу сервисного билета, а также доступ к различным сервисам. В результате тестирования были выявлены и исправлены ошибки, связанные с сериализацией данных и проверкой временных меток, что повысило устойчивость системы к возможным атакам и уязвимостям.

Разработка программного средства продемонстрировала полное соответствие основным принципам безопасности протокола *Kerberos*, включая использование временных меток и динамическую генерацию сессионных ключей для каждой сессии. Это обеспечивает защиту от атак повторного воспроизведения и гарантирует, что даже при компрометации одного из компонентов системы, другие остаются защищёнными.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Kerberos простыми словами [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/803163/.

[2] Cryptography documentation. [Электронный ресурс]. – Режим доступа: https://cryptography.io/.

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

import json

import time

import base64

import hashlib

from cryptography.fernet import Fernet

def generate\_key\_from\_password(password):

digest = hashlib.sha256(password.encode()).digest()

return base64.urlsafe\_b64encode(digest)

class AuthenticationServer:

def \_\_init\_\_(self, tgs\_secret\_key):

self.clients = {}

self.tgs\_secret\_key = tgs\_secret\_key

def register\_client(self, client\_id, password):

key = generate\_key\_from\_password(password)

self.clients[client\_id] = key

def request\_tgt(self, client\_id, tgs\_id):

if client\_id not in self.clients:

raise Exception("Client not registered")

client\_key = self.clients[client\_id]

session\_key = Fernet.generate\_key()

timestamp = int(time.time())

# Преобразуем session\_key в строку для JSON

tgt\_data = {

'client\_id': client\_id,

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

'timestamp': timestamp,

'lifetime': 3600

}

fernet\_tgs = Fernet(self.tgs\_secret\_key)

tgt = fernet\_tgs.encrypt(json.dumps(tgt\_data).encode())

response\_data = {

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

'tgt': tgt.decode(),

'timestamp': timestamp,

'lifetime': 3600

}

fernet\_client = Fernet(client\_key)

return fernet\_client.encrypt(json.dumps(response\_data).encode())

class TicketGrantingServer:

def \_\_init\_\_(self, tgs\_secret\_key):

self.tgs\_secret\_key = tgs\_secret\_key

self.services = {}

def register\_service(self, service\_id, service\_key):

self.services[service\_id] = service\_key

def request\_service\_ticket(self, tgt, authenticator, service\_id):

fernet\_tgs = Fernet(self.tgs\_secret\_key)

decrypted\_tgt = json.loads(fernet\_tgs.decrypt(tgt).decode())

session\_key = decrypted\_tgt['session\_key'].encode()

client\_id = decrypted\_tgt['client\_id']

fernet\_session = Fernet(session\_key)

decrypted\_auth = json.loads(fernet\_session.decrypt(authenticator).decode())

if decrypted\_auth['client\_id'] != client\_id:

raise Exception("Client ID mismatch")

if time.time() - decrypted\_auth['timestamp'] > 30:

raise Exception("Authenticator expired")

if service\_id not in self.services:

raise Exception("Service not registered")

service\_key = self.services[service\_id]

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

ticket\_data = {

'client\_id': client\_id,

'service\_session\_key': service\_session\_key.decode(),

'timestamp': int(time.time()),

'lifetime': 3600

}

fernet\_service = Fernet(service\_key)

service\_ticket = fernet\_service.encrypt(json.dumps(ticket\_data).encode())

response\_data = {

'service\_session\_key': service\_session\_key.decode(),

'service\_ticket': service\_ticket.decode()

}

return fernet\_session.encrypt(json.dumps(response\_data).encode())

class ApplicationServer:

def \_\_init\_\_(self, service\_key):

self.service\_key = service\_key

def verify\_ticket(self, service\_ticket, authenticator):

fernet\_service = Fernet(self.service\_key)

ticket\_data = json.loads(fernet\_service.decrypt(service\_ticket).decode())

session\_key = ticket\_data['service\_session\_key'].encode()

client\_id = ticket\_data['client\_id']

fernet\_session = Fernet(session\_key)

auth\_data = json.loads(fernet\_session.decrypt(authenticator).decode())

if auth\_data['client\_id'] != client\_id:

raise Exception("Client ID mismatch")

if time.time() - auth\_data['timestamp'] > 30:

raise Exception("Authenticator expired")

return True

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

# Генерация ключей

tgs\_key = Fernet.generate\_key()

ap\_key = Fernet.generate\_key()

# Инициализация серверов

as\_server = AuthenticationServer(tgs\_key)

tgs\_server = TicketGrantingServer(tgs\_key)

ap\_server = ApplicationServer(ap\_key)

# Регистрация клиента и сервиса

as\_server.register\_client("alice", "password123")

tgs\_server.register\_service("file\_server", ap\_key)

# Клиентская часть

client = type('', (), {})()

client.id = "alice"

client.password = "password123"

# Аутентификация в AS

client\_key = generate\_key\_from\_password(client.password)

tgt\_response = as\_server.request\_tgt(client.id, "tgs")

response = Fernet(client\_key).decrypt(tgt\_response)

tgt\_data = json.loads(response.decode())

client.session\_key = tgt\_data['session\_key'].encode()

client.tgt = tgt\_data['tgt'].encode()

# Получение сервисного билета

authenticator = Fernet(client.session\_key).encrypt(

json.dumps({'client\_id': client.id, 'timestamp': int(time.time())}).encode()

)

service\_ticket\_response = tgs\_server.request\_service\_ticket(

client.tgt, authenticator, "file\_server"

)

service\_data = json.loads(Fernet(client.session\_key).decrypt(service\_ticket\_response).decode())

client.service\_session\_key = service\_data['service\_session\_key'].encode()

client.service\_ticket = service\_data['service\_ticket'].encode()

# Доступ к сервису

auth = Fernet(client.service\_session\_key).encrypt(

json.dumps({'client\_id': client.id, 'timestamp': int(time.time())}).encode()

)

print("Authentication successful:", ap\_server.verify\_ticket(client.service\_ticket, auth))