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
2 Ход работы 4
Заключение 5

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.

```
Kerberos-сервер запущен на localhost:12345
[AS] Для клиента client1:
     session_key = 8832be515bb5efad44bee27771a42897
     TGT_plain    = b'client1::\x882\xbeQ[\xb5\xef\xadD\xbe\xe2wq\xa4(\x97'
[TGS] Извлечён client_id: client1
[TGS] Извлечён session_key (из TGT) = 8832be515bb5efad44bee27771a42897
[TGS] Для сервиса fileserver:
     service_session_key = 52624f23d46438696de6b8919820f339
     service_ticket_plain = b'{"client_id": "client1", "service_session_key": "UmJPI9RkOGlt5riRmCDzOQ=="}'
[SERVICE] Подтверждён client_id: client1
```

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()
```