МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**
**(МОСКОВСКИЙ ПОЛИТЕХ)**

Факультет информационных технологий
Кафедра «Инфокогнитивные технологии»

# Лабораторная работа 7

По дисциплине «Защита информации»
Направление подготовки 09.03.03 «Прикладная информатика»
Профиль «Корпоративные информационные системы»

**Выполнил:**

студент группы 201-361

Погудин Александр

Москва 2023

Цель работы: написать простое клиент-серверное приложение, в котором сервер выступает в качестве удостоверяющего центра, а клиенты могут обмениваться подписанными документами (с возможностью проверки подписей). Используя шифр: RSA.

## Введение

**Клиент-серверное приложение** - это приложение, состоящее из двух основных компонентов: сервера и клиента. Сервер представляет собой компьютер, который предоставляет какую-то услугу (например, базу данных, файлы, веб-страницы и т.д.). Клиент, в свою очередь, представляет собой компьютер, который запрашивает эту услугу через сеть (обычно Интернет). Таким образом, сервер обеспечивает доступ к каким-то ресурсам или услугам, а клиенты используют эти ресурсы или услуги, подключаясь к серверу по определенному протоколу. Взаимодействие между клиентами и сервером обычно происходит по сети в реальном времени. Примеры клиент-серверных приложений: - Интернет-магазин, где сервер предоставляет каталог товаров, а клиенты могут добавлять товары в корзину и оформлять заказы. - Электронная почта, где сервер предоставляет возможность отправлять и получать электронные письма, а клиенты используют почтовые программы для взаимодействия с сервером. - Онлайн-игры, где сервер обеспечивает игровое поле, а клиенты через Интернет играют в игру и обмениваются данными с сервером.

## Программы

```
# Создание приватного ключа
command = "openssl genpkey -algorithm RSA -out privatekey.pem -pkeyopt rsa_keygen_bits:1024"
result = subprocess.run(command, shell=True, capture_output=True)
print("--- private key created ---")

def main():
    # Создаем сокет и слушаем порт
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 5000))
    server_socket.listen(10)

    # Подключение клиентов
    print('Server is listening...')
    connection1, address1 = server_socket.accept()
    print(f"New client connected: {address1}")

    # Получаем зашифрованный документ от клиента
    # Расшифровываем документ
    file = open("message_server.enc", "wb")
    print("receiving data from server")
    file_data = connection1.recv(4096)
    file.write(file_data)
    file.close()
    print("file downloaded")

    # расшифровываем документ
    command = "openssl pkeyutl -decrypt -inkey privatekey.pem -in message_server.enc -out message_server.dec"
    result = subprocess.run(command, shell=True, capture_output=True)
    print("--- message deciphered ---")

    # вычисляем цифровой подписи
    command = "openssl dgst -sha256 -sign privatekey.pem -out signature.bin message_server.dec"
    result = subprocess.run(command, shell=True, capture_output=True)
    print("--- message signed ---")

    connection1.send("Message signed".encode())

    print('Server is listening...')
    connection2, address2 = server_socket.accept()
    print(f"New client connected: {address2}")


    file = open("message_server.enc", "rb")
    while True:
        file_data = file.read(4096)
        connection2.send(file_data)
        if not file_data:
            break
    print("file sended")


if __name__ == "__main__":
    main()
```

Программа представляет собой серверное приложение, которое создает приватный ключ шифрования, прослушивает порт 5000 для подключения клиентов, принимает зашифрованный документ от клиента, расшифровывает его с помощью приватного ключа, вычисляет цифровую подпись, подписывает расшифрованный документ, отправляет клиенту сообщение о

том, что документ подписан, прослушивает порт еще раз для подключения второго клиента и отправляет ему зашифрованный документ.

Для создания приватного ключа используется команда "openssl genpkey -algorithm RSA -out privatekey.pem -pkeyopt rsa_keygen_bits:1024", которая генерирует приватный ключ алгоритма RSA и сохраняет его в файл privatekey.pem.

После создания сокета и прослушивания порта, сервер принимает подключение первого клиента, который отправляет зашифрованный документ. Документ сохраняется на сервере, затем он расшифровывается с помощью приватного ключа командой "openssl pkeyutl -decrypt -inkey privatekey.pem -in message_server.enc -out message_server.dec".

Далее, вычисляется цифровая подпись командой "openssl dgst -sha256 -sign privatekey.pem -out signature.bin message_server.dec", которая использует приватный ключ для вычисления подписи в файл signature.bin.

После этого сервер отправляет клиенту сообщение о том, что документ подписан, прослушивает порт еще раз для подключения второго клиента и отправляет ему зашифрованный документ командой "connection2.send(file_data)".

Код уже содержит ряд комментариев, которые поясняют, что делает каждая строка.

```python
# Соединяемся с сервером
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 5000))

# Создание публичного ключа
command = "openssl rsa -pubout -in privatekey.pem -out publickey_user1.pem"
result = subprocess.run(command, shell=True, capture_output=True)
print("--- public key created ---")

# Шифруем документ
command = "openssl rsautl -encrypt -inkey publickey_user1.pem -pubin -in message_user1.txt -out message_user1.enc"
result = subprocess.run(command, shell=True, capture_output=True)
print("--- message encrypted ---")

# Отправляем зашифрованный документ серверу
file = open("message_user1.enc", "rb")
file_data = file.read(4096)
s.send(file_data)
print("file sended")

data = s.recv(1024).decode()
print('Ответ от сервера:', data)

# Верификация цифровой подписи файла
command = "openssl dgst -sha256 -verify publickey_user1.pem -signature signature.bin message_user1.txt"
result = subprocess.run(command, shell=True, capture_output=True)
print(f"--- verification {'passed' if result.returncode == 0 else 'failed'} ---")
```

Программа представляет собой клиентское приложение, которое соединяется с сервером, создает публичный ключ шифрования на основе приватного ключа, шифрует документ с помощью публичного ключа, отправляет зашифрованный документ серверу, принимает от сервера сообщение, что документ подписан, и верифицирует цифровую подпись файла.

Для соединения с сервером используется сокет, который создается с помощью `socket.socket`. Сокет подключается к серверу на адрес `localhost` и порту `5000` с помощью метода `s.connect(('localhost', 5000))`.

Затем, создается публичный ключ на основе приватного ключа с помощью команды `openssl rsa -pubout -in privatekey.pem -out publickey_user1.pem`. Она берет приватный ключ из файла `privatekey.pem` и создает на его базе публичный ключ, который сохраняется в файл `publickey_user1.pem`.

После этого, документ `message_user1.txt` шифруется с помощью публичного ключа с помощью команды `openssl rsautl -encrypt -inkey publickey_user1.pem -pubin -in message_user1.txt -out message_user1.enc`. Зашифрованный документ сохраняется в файл `message_user1.enc`.

Зашифрованный документ отправляется серверу с помощью метода `s.send(file_data)`, где `file_data` является байтовыми данными зашифрованного документа.

После этого, клиент ожидает ответ от сервера и принимает его с помощью метода `s.recv(1024).decode()`. Ответ сервера сообщает клиенту, что файл успешно подписан.

Затем, верифицируется цифровая подпись файла с помощью команды `openssl dgst -sha256 -verify publickey_user1.pem -signature signature.bin message_user1.txt`. Эта команда использует публичный ключ для проверки цифровой подписи файла `signature.bin` на основе исходного файла `message_user1.txt`. Результат верификации выводится в консоль с помощью строк `f"--- verification {'passed' if result.returncode == 0 else 'failed'} ---"`. Если верификация прошла успешно, то в консоль будет выведено "--- verification passed ---", в противном случае "--- verification failed ---".

```
# Соединяемся с сервером
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 5000))

# Создание публичного ключа
command = "openssl rsa -pubout -in privatekey.pem -out publickey_user2.pem"
result = subprocess.run(command, shell=True, capture_output=True)
print("--- public key created ---")

# Получаем зашифрованный документ от клиента

file = open("message_user2.enc", "wb")
print("receiving data from server")
while True:
    file_data = s.recv(4096)
    file.write(file_data)
    if not file_data:
        break
file.close()
print("file downloaded")

# Расшифровываем документ
command = "openssl pkeyutl -decrypt -inkey privatekey.pem -in message_server.enc -out message_user2.dec"
result = subprocess.run(command, shell=True, capture_output=True)
print("--- message deciphered ---")

# Верификация цифровой подписи файла
command = "openssl dgst -sha256 -verify publickey_user2.pem -signature signature.bin message_user2.dec"
result = subprocess.run(command, shell=True, capture_output=True)
print(f"--- verification {'passed' if result.returncode == 0 else 'failed'} ---")
```

Данная программа представляет собой клиентское приложение, которое соединяется с сервером, создает публичный ключ шифрования на основе приватного ключа, принимает зашифрованный документ от сервера, расшифровывает его с помощью приватного ключа, и верифицирует цифровую подпись файла.

Для соединения с сервером используется сокет, который создается с помощью `socket.socket`. Сокет подключается к серверу на адрес `localhost` и порту `5000` с помощью метода `s.connect(('localhost', 5000))`.

Затем, создается публичный ключ на основе приватного ключа с помощью команды `openssl rsa -pubout -in privatekey.pem -out publickey_user2.pem`. Она берет приватный ключ из файла `privatekey.pem` и создает на его базе публичный ключ, который сохраняется в файл `publickey_user2.pem`.

После этого, клиент ожидает получения зашифрованного файла от сервера с помощью `s.recv(4096)`. Зашифрованный документ сохраняется в файл `message_user2.enc`.

Затем, зашифрованный документ расшифровывается с помощью приватного ключа командой `openssl pkeyutl -decrypt -inkey privatekey.pem -in message_server.enc -out message_user2.dec`. Расшифрованный документ сохраняется в файл `message_user2.dec`.

Затем, верифицируется цифровая подпись файла с помощью команды `openssl dgst -sha256 -verify publickey_user2.pem -signature signature.bin message_user2.dec`. Эта команда использует публичный ключ для проверки цифровой подписи файла `signature.bin` на основе расшифрованного файла `message_user2.dec`. Результат верификации выводится в консоль с помощью строк `f"--- verification {'passed' if result.returncode == 0 else 'failed'} ---"`. Если верификация прошла успешно, то в консоль будет выведено "--- verification passed ---", в противном случае "--- verification failed ---".

## Результат работы программ

Сервер

```
--- private key created ---
Server is listening...
New client connected: ('127.0.0.1', 14467)
receiving data from server
file downloaded
--- message deciphered ---
--- message signed ---
Server is listening...
New client connected: ('127.0.0.1', 14470)
file sended
```

Клиент1 (Отправитель)

```
--- public key created ---
--- message encrypted ---
file sended
Ответ от сервера: Message signed
--- verification passed ---
```

Клиент1 (Получатель)

```
--- public key created ---
receiving data from server
file downloaded
--- message deciphered ---
--- verification passed ---
```

## Вывод

В данной системе три программы: сервер, клиент 1 и клиент 2.

Сервер генерирует приватный ключ и прослушивает порт для подключения клиентов. Он принимает зашифрованный документ от первого клиента, расшифровывает его с помощью приватного ключа и подписывает цифровой подписью с помощью того же приватного ключа. Затем сервер отсылает зашифрованный документ второму клиенту.

Клиент 1 создает публичный ключ, шифрует документ и отправляет его на сервер. Клиент 1 также верифицирует цифровую подпись от сервера с помощью своего публичного ключа.

Клиент 2 создает публичный ключ, принимает зашифрованный документ от сервера, расшифровывает его и верифицирует цифровую подпись от сервера с помощью своего публичного ключа.

Таким образом, эта система использует асимметричное шифрование RSA и цифровые подписи для обеспечения конфиденциальности, целостности и подлинности передаваемых данных между клиентами и сервером. Каждый клиент имеет свой пару ключей, один из которых приватный, а другой - публичный. Приватные ключи используются только для расшифровки сообщений и создания цифровых подписей, которые затем могут быть проверены с помощью соответствующих публичных ключей. Обмен публичными ключами позволяет каждому клиенту шифровать сообщения, которые могут быть безопасно переданы по незащищенной сети.