Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Розграхункова-графічна роботи (Розширена)

з дисципліни

«Безпека програмного забезпечення»

Виконав:
студент групи ІМ-11
Царик Микола Миколайович

Перевірив:
Микита Євгенович Меленчуков

Київ 2025

Завдання

Безпека ПЗ.

Завдання на РГР. Курс Безпека ПЗ 2024.

Базовий варіант. Реалізувати імітацію TLS \ SSL “рукостискання” (**10 балів**):

1. Ініціювання клієнтом:
   - Клієнт ініціює рукостискання, надсилаючи повідомлення "привіт" (рандомно згенероване) на сервер.
2. Відповідь сервера:
   - Сервер відповідає повідомленням "привіт сервера" (рандомно згенероване), що містить також SSL-сертифікат (.509). (Або для базового варіанту - сервер генерує public \ private ключі та відправляє public клієнту).
3. Автентифікація:
   - Клієнт перевіряє SSL-сертифікат сервера в центрі сертифікації для підтвердження ідентичності сервера. (У випадку базового варіанту - пропустіть)
4. Обмін секретними рядками:
   - Клієнт надсилає секрет premaster, який шифрується відкритим ключем сервера.
   - Сервер розшифровує секрет premaster.
5. Генерація ключів сеансу:
   - Клієнт і сервер генерують ключі сеансу з клієнтського та серверного випадкових рядків і секрету premaster.
6. Готовність клієнта та сервера:
   - Клієнт та сервер надсилають повідомлення "готовий", зашифроване сеансовим ключем.
7. Завершення рукостискання:
   - Здійснюється безпечне симетричне шифрування, і рукостискання завершується.
   - Зв'язок продовжується за допомогою ключів сеансу.

Реалізувати передачу даних по захищеному каналу (наприклад, повідомлення чату, текстовий файл).

Розширений варіант (завдання узгоджується з викладачем) (**10 балів**):

- Розгорніть сервер для перевірки root-сертифіката.
- Змоделюйте розподілену топологію (3 і більше нод), де хендшейк може бути між будь якою парою.

Увага! Для отримання оцінки 100 за курс потрібно виконати розширений варіант.

Оформити звіт з РГР. Титульний лист, стисла теоретична частина і практична реалізація кожного розділу, список літератури (обов’язково посилання на власний гітхаб чи аналогічну систему).

Теоретична частина

1. TLS/SSL протокол

TLS (Transport Layer Security) — це криптографічний протокол, який забезпечує захист даних під час їх передачі через мережу за допомогою шифрування, цілісності та автентифікації. TLS є еволюцією SSL і широко використовується для захисту вебтрафіку, електронної пошти та інших мережевих протоколів.

2. Центр сертифікації (Certificate Authority, CA)

Центр сертифікації — це довірений суб'єкт, який видає цифрові сертифікати для підтвердження справжності ключів, що використовуються в криптографічних протоколах. CA підписує сертифікати за допомогою свого приватного ключа, гарантує автентичність суб'єкта та цілісність його даних.

3. Цифровий сертифікат

Цифровий сертифікат — це документ у цифровому форматі, який підтверджує власника публічного ключа та містить його дані (ім'я, домен, організацію тощо). Сертифікат підписується Центром сертифікації для підтвердження його автентичності.

4. Certificate Signing Request (CSR)

CSR — це запит на підписання сертифіката, який містить публічний ключ та інформацію про суб'єкта (організація, домен, країна тощо). CSR підписується приватним ключем суб'єкта перед відправленням до CA.

5. Асиметричне шифрування

Це метод шифрування, де використовуються дві різні частини ключа: публічний ключ для шифрування даних і приватний ключ для їх розшифрування. RSA є прикладом алгоритму асиметричного шифрування.

6. Симетричне шифрування (AES)

Симетричне шифрування використовує один і той самий ключ для шифрування та розшифрування даних. AES (Advanced Encryption Standard) — це сучасний алгоритм блочного шифрування, який підтримує ключі розміром 128, 192 і 256 біт.

7. Режим шифрування CFB (Cipher Feedback)
CFB — це режим роботи симетричного шифрування, який дозволяє обробляти дані потоково (байт за байтом), забезпечуючи захист без необхідності додавання заповнення (padding).

8. Вектор ініціалізації (IV)
IV — це випадкові дані, що використовуються в алгоритмах шифрування для забезпечення випадковості результату навіть при однакових вхідних даних і ключах. IV додає криптографічну стійкість.

9. HMAC (Hash-based Message Authentication Code)
HMAC — це механізм для перевірки цілісності та автентичності повідомлень. Він базується на використанні хеш-функції (наприклад, SHA-256) та секретного ключа.

10. HKDF (HMAC-based Key Derivation Function)
HKDF — це функція генерації ключів, яка дозволяє створювати сильні симетричні ключі на основі вихідного матеріалу (наприклад, premaster_secret).

11. Premaster Secret
Premaster Secret — це секретний ключ, який обмінюється між клієнтом і сервером за допомогою асиметричного шифрування. Він використовується для генерації сеансового симетричного ключа.

12. Сеансовий ключ
Сеансовий ключ генерується для кожного сеансу з використанням Premaster Secret, випадкових значень клієнта та сервера. Він використовується для симетричного шифрування даних упродовж сеансу.

13. Цифровий підпис
Цифровий підпис — це зашифрований хеш повідомлення, створений за допомогою приватного ключа. Він гарантує автентичність відправника та цілісність даних.

14. Верифікація сертифіката
Верифікація сертифіката — це процес перевірки підпису сертифіката за допомогою публічного ключа CA. Успішна перевірка підтверджує, що сертифікат виданий довіреним центром і не був змінений.

Хід виконання

1. Ініціалізація клієнтом

Клієнт ініціює TLS-рукостискання, надсилаючи повідомлення "привіт клієнта" (випадково згенероване значення) на сервер.

Код клієнта:

```
# Клієнт надсилає серверу випадковий рядок ("привіт клієнта").
hello_client = os.urandom(16).hex()
client.sendall(hello_client.encode())
print(f"Відправлено серверу: {hello_client}")
```

Код сервера:

```
# Отримання "hello_client" від клієнта
data = conn.recv(1024).decode()
hello_client = data
print(f"Отримано від клієнта: {hello_client}")
```

2. Відповідь сервера

Сервер відповідає клієнту повідомленням "привіт сервера", яке містить також публічний ключ сервера та сертифікат, виданий центром сертифікації.

Код сервера:

```
# Формування даних для клієнта
hello_server = os.urandom(16).hex()
response = {
    "hello_server": hello_server,
    "public_key": public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    ).decode(),
    "certificate":
server_certificate.public_bytes(serialization.Encoding.PEM).decode()
}

conn.sendall(json.dumps(response).encode())
print("Відправлено клієнту: hello_server + публічний ключ +
сертифікат")
```

Код клієнта:

```
# Отримання привіт, публічний ключ, сертифікат від серверу
data = client.recv(4096).decode()
response = json.loads(data)

hello_server = response["hello_server"]
server_public_key_pem = response["public_key"]
server_cert_pem = response["certificate"]
print(f"Отримано від сервера: hello_server = {hello_server}")

server_public_key = serialization.load_pem_public_key(
    server_public_key_pem.encode()
)
print("Публічний ключ сервера завантажено.")

server_certificate =
load_pem_x509_certificate(server_cert_pem.encode())
print("Сертифікат сервера завантажено.")
```

3. Автентифікація

Клієнт перевіряє отриманий сертифікат у центрі сертифікації для підтвердження ідентичності сервера.

Код клієнта:

```
# Підключення до CA для перевірки сертифіката
ca_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ca_conn.connect(('localhost', 65431))
ca_conn.sendall(server_cert_pem.encode())  # Надсилання сертифіката до
CA
response = ca_conn.recv(1024).decode()
ca_conn.close()

if response != "cert_valid":
    print("Сертифікат недійсний!")
    client.close()
    exit()
print("Сертифікат сервера дійсний.")
```

Код центру сертифікації:

```
elif b"-----BEGIN CERTIFICATE-----" in data:
    # Обробка сертифіката від клієнта
    print("Отримано сертифікат для перевірки.")
    certificate = load_pem_x509_certificate(data)
    if verify_certificate(certificate):
        conn.sendall(b"cert_valid")
        print("Сертифікат дійсний.")
```

```
    else:
        conn.sendall(b"cert_invalid")
        print("Сертифікат не дійсний.")
```

4. Обмін секретними рядками

Клієнт надсилає зашифрований premaster_secret, який сервер розшифровує своїм приватним ключем.

Код клієнта:

```
# Генерація premaster secret
premaster_secret = os.urandom(32)
encrypted_premaster = server_public_key.encrypt(
    premaster_secret,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)
client.sendall(encrypted_premaster)
print("Зашифрований premaster secret надіслано серверу.")
```

Код сервера:

```
# Отримання зашифрованого premaster secret
encrypted_premaster = conn.recv(1024)
premaster_secret = private_key.decrypt(
    encrypted_premaster,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)
print(f"Розшифровано premaster secret: {premaster_secret.hex()}")
```

5. Генерація ключів сеансу

Клієнт і сервер генерують симетричні ключі сеансу на основі hello_client, hello_server та premaster_secret.

Код клієнта та сервера (однаковий):

```
# Генерація симетричного ключа
key_material = hello_client.encode() + hello_server.encode() +
premaster_secret
session_key = HKDF(
```

```
    algorithm=hashes.SHA256(),
    length=32,
    salt=None,
    info=b"handshake data",
).derive(key_material)
print(f"Згенеровано симетричний ключ: {session_key.hex()}")
```

6. Готовність клієнта та сервера

Клієнт і сервер обмінюються зашифрованими повідомленнями "готовий".

Код сервера:

```
# Обмін повідомленням "готовий"
iv = os.urandom(16)
cipher = Cipher(algorithms.AES(session_key), modes.CFB(iv))
encryptor = cipher.encryptor()
ready_message = "готовий".encode()
encrypted_ready = iv + encryptor.update(ready_message) +
encryptor.finalize()
conn.sendall(encrypted_ready)
print("Відправлено зашифроване повідомлення 'готовий' клієнту")
```

Код клієнта:

```
# Отримання та розшифрування повідомлення "готовий"
ciphertext = client.recv(1024)
iv = ciphertext[:16]
encrypted_message = ciphertext[16:]

cipher = Cipher(algorithms.AES(session_key), modes.CFB(iv))
decryptor = cipher.decryptor()
message = decryptor.update(encrypted_message) + decryptor.finalize()
print(f"Отримано розшифроване повідомлення від сервера:
{message.decode()}")

# Надсилання повідомлення "готовий" серверу
ready_message = "готовий".encode()
iv = os.urandom(16)
cipher = Cipher(algorithms.AES(session_key), modes.CFB(iv))
encryptor = cipher.encryptor()
encrypted_ready = iv + encryptor.update(ready_message) +
encryptor.finalize()
client.sendall(encrypted_ready)
print("Повідомлення 'готовий' надіслано серверу.")
```

7. Завершення рукостискання та встановлення захищеного з'єднання

Після обміну повідомленнями "готовий" клієнт і сервер починають використовувати симетричний ключ для обміну даними.

Код клієнта:

```
# Надсилання повідомлення серверу через захищене з'єднання
final_message = "Це повідомлення від клієнта!".encode()
iv = os.urandom(16)
cipher = Cipher(algorithms.AES(session_key), modes.CFB(iv))
encryptor = cipher.encryptor()
encrypted_message = iv + encryptor.update(final_message) +
encryptor.finalize()
client.sendall(encrypted_message)
print("Повідомлення надіслано серверу.")
```

Код сервера:

```
# Отримання зашифрованого повідомлення від клієнта
ciphertext = conn.recv(1024)
iv = ciphertext[:16]
encrypted_message = ciphertext[16:]

cipher = Cipher(algorithms.AES(session_key), modes.CFB(iv))
decryptor = cipher.decryptor()
final_message = decryptor.update(encrypted_message) +
decryptor.finalize()
print(f"Розшифроване повідомлення від клієнта:
{final_message.decode()}")
```

8. Верифікація сертифіката сервера (розширена функціональність)

У розширеному завданні додано використання центру сертифікації (CA), який видає та верифікує сертифікати сервера. Це дозволяє гарантувати автентичність сервера.

Код сервера:

```
# Створення CSR для отримання сертифіката
csr = CertificateSigningRequestBuilder().subject_name(
    x509.Name([
        x509.NameAttribute(NameOID.COUNTRY_NAME, "UA"),
        x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, "Kyiv"),
        x509.NameAttribute(NameOID.LOCALITY_NAME, "Kyiv"),
        x509.NameAttribute(NameOID.ORGANIZATION_NAME, "Server
```

```
Organization"),
        x509.NameAttribute(NameOID.COMMON_NAME, "localhost"),
    ])
).sign(private_key, hashes.SHA256())

# Підключення до CA
ca_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ca_conn.connect(('localhost', 65431))
print("Підключено до центру сертифікації.")

# Відправка CSR до CA
csr_pem = csr.public_bytes(serialization.Encoding.PEM)
ca_conn.sendall(csr_pem)
print("CSR надіслано до центру сертифікації.")

# Отримання підписаного сертифіката
cert_pem = ca_conn.recv(4096)
server_certificate = load_pem_x509_certificate(cert_pem)
print("Сертифікат отримано від центру сертифікації.")
ca_conn.close()
```

Код клієнта:

```
# Верифікація сертифіката через CA
ca_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ca_conn.connect(('localhost', 65431))
ca_conn.sendall(server_cert_pem.encode())  # Надсилання сертифіката до
CA
response = ca_conn.recv(1024).decode()
ca_conn.close()

if response != "cert_valid":
    print("Сертифікат недійсний!")
    client.close()
    exit()
print("Сертифікат сервера дійсний.")
```

Код центру сертифікації:

```
elif b"-----BEGIN CERTIFICATE REQUEST-----" in data:
    # Обробка CSR від сервера
    print("Отримано CSR.")
    csr = load_pem_x509_csr(data)
    signed_cert = sign_csr(csr)
    conn.sendall(signed_cert.public_bytes(serialization.Encoding.PEM))
    print("Підписаний сертифікат відправлений.")
elif b"-----BEGIN CERTIFICATE-----" in data:
    # Обробка сертифіката від клієнта
    print("Отримано сертифікат для перевірки.")
    certificate = load_pem_x509_certificate(data)
    if verify_certificate(certificate):
```

```
        conn.sendall(b"cert_valid")
        print("Сертифікат дійсний.")
    else:
        conn.sendall(b"cert_invalid")
        print("Сертифікат не дійсний.")
```

9. Захищений обмін даними після рукостискання

Клієнт і сервер використовують симетричний ключ для захищеного обміну повідомленнями.

Код клієнта:

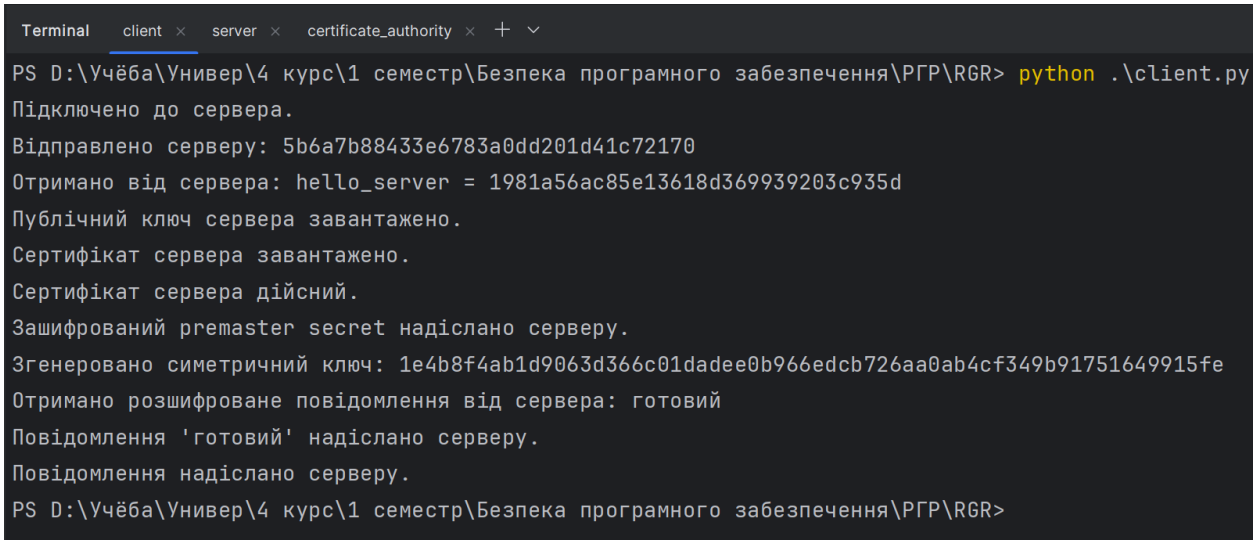
```
# Надсилання нового повідомлення серверу
secure_message = "Захищений обмін даними.".encode()
iv = os.urandom(16)
cipher = Cipher(algorithms.AES(session_key), modes.CFB(iv))
encryptor = cipher.encryptor()
encrypted_message = iv + encryptor.update(secure_message) +
encryptor.finalize()
client.sendall(encrypted_message)
print("Захищене повідомлення надіслано серверу.")
```

Код сервера:

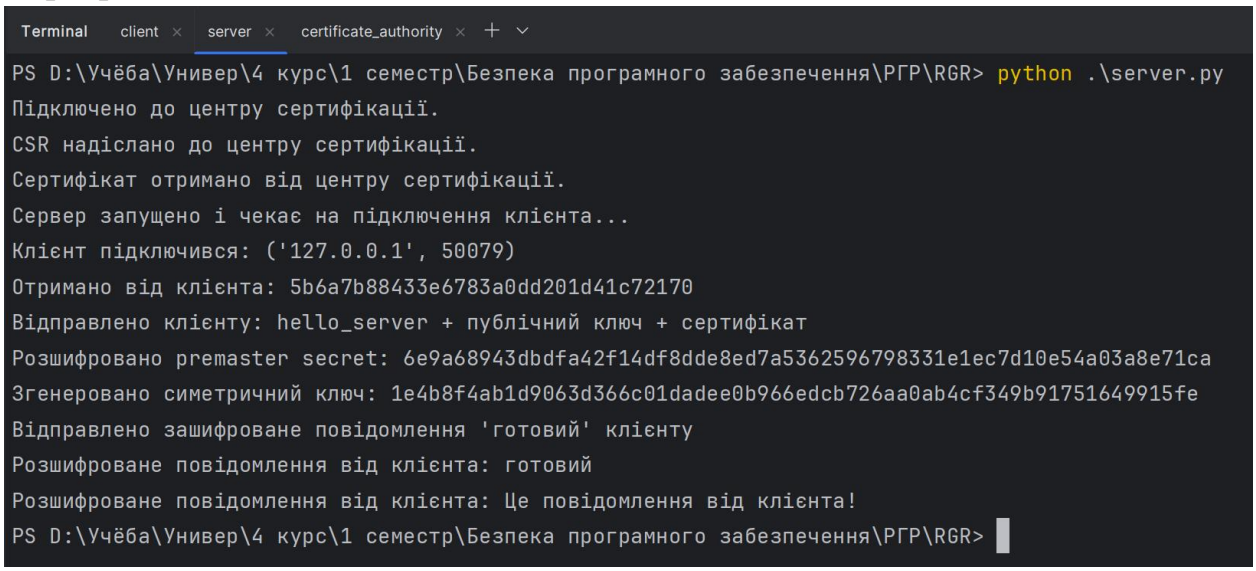
```
# Отримання нового зашифрованого повідомлення від клієнта
ciphertext = conn.recv(1024)
iv = ciphertext[:16]
encrypted_message = ciphertext[16:]

cipher = Cipher(algorithms.AES(session_key), modes.CFB(iv))
decryptor = cipher.decryptor()
secure_message = decryptor.update(encrypted_message) +
decryptor.finalize()
print(f"Отримано захищене повідомлення від клієнта:
{secure_message.decode()}")
```

Результати

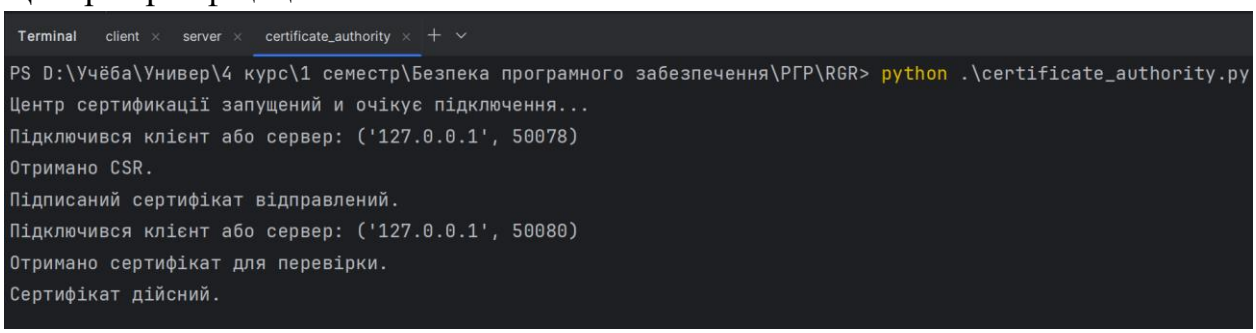
Клієнт:



Сервер:



Центр сертифіцації:



Висновок

У ході виконання розрахунково-графічної роботи було реалізовано протокол імітації TLS/SSL-рукостискання, який включав ініціалізацію рукостискання клієнтом, обмін повідомленнями між клієнтом і сервером, автентифікацію сервера за допомогою центру сертифікації (CA), обмін секретними даними (premaster secret) та генерацію симетричних ключів для подальшого захищеного обміну даними. Клієнт та сервер виконали успішну комунікацію, включаючи передачу "hello" повідомлень, сертифікатів та обмін ключами. Сертифікат сервера було перевірено через центр сертифікації, що гарантувало його дійсність та автентичність. Обидві сторони згенерували ідентичні ключі сесії, використовуючи рандомні значення та premaster secret, що дозволило здійснити захищений обмін даними з використанням алгоритму AES. Окрім базового функціоналу, було реалізовано розширений сценарій із використанням центру сертифікації для створення, підписання та перевірки сертифікатів, а також перевірки цілісності та автентичності переданих даних.

Список джерел:

1. YouTube. "TLS Handshake in Depth". Джерело.
2. Cryptography.io. "X.509 Certificate API". Джерело.
3. YouTube. "Public Key Infrastructure and Certificates". Джерело.
4. Python.org. "socket — Low-level networking interface". Джерело.

Посилання на репозиторій в GitHub:

https://github.com/TsNikolay/TLS-SSL-handshake-simulation