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

Розрахункова графічна робота

з дисципліни

«Безпека програмного забезпечення»

Виконав:

студент групи ІП-05

Гапій Денис Едуардович

Номер залікової: 0504

Перевірив:

доц. Волокита А. М.

Київ 2023

Тема: «Системи безпеки програм і даних.»

Завдання:

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).

Виконання:

Лістинг server.js:

```
const tls = require('tls');

const fs = require('fs');

const crypto = require('crypto');

const options = {

    key: fs.readFileSync('server-key.pem'),

    cert: fs.readFileSync('server-cert.pem'),
```

```
};
const server = tls.createServer(options, (cleartextStream) => {
    console.log('Server: Client connected');
    cleartextStream.on('data', (data) => {
        console.log('Server received data:', data.toString());
        if (data.toString().includes('hello')) {
            const serverHelloMessage = 'Hello from the server!';
            cleartextStream.write(serverHelloMessage);

            // Generate a premaster secret for demonstration purposes (replace with your logic)
            const premasterSecret = 'ThisIsAPremasterSecret';
            // Encrypt the premaster secret using the client's public key with PKCS#1 v1.5 padding
            const encryptedPremasterSecret = crypto.publicEncrypt(
                {
                    key: fs.readFileSync('client-cert.pem'),  // Client's public key
                    padding: crypto.constants.RSA_PKCS1_PADDING,
                },
                Buffer.from(premasterSecret, 'utf8')  // Ensure proper encoding
            );
            // Send the encrypted premaster secret to the client
            cleartextStream.write(encryptedPremasterSecret);
        }
    });
    cleartextStream.on('end', () => {
        console.log('Server: Client disconnected');
    });
```

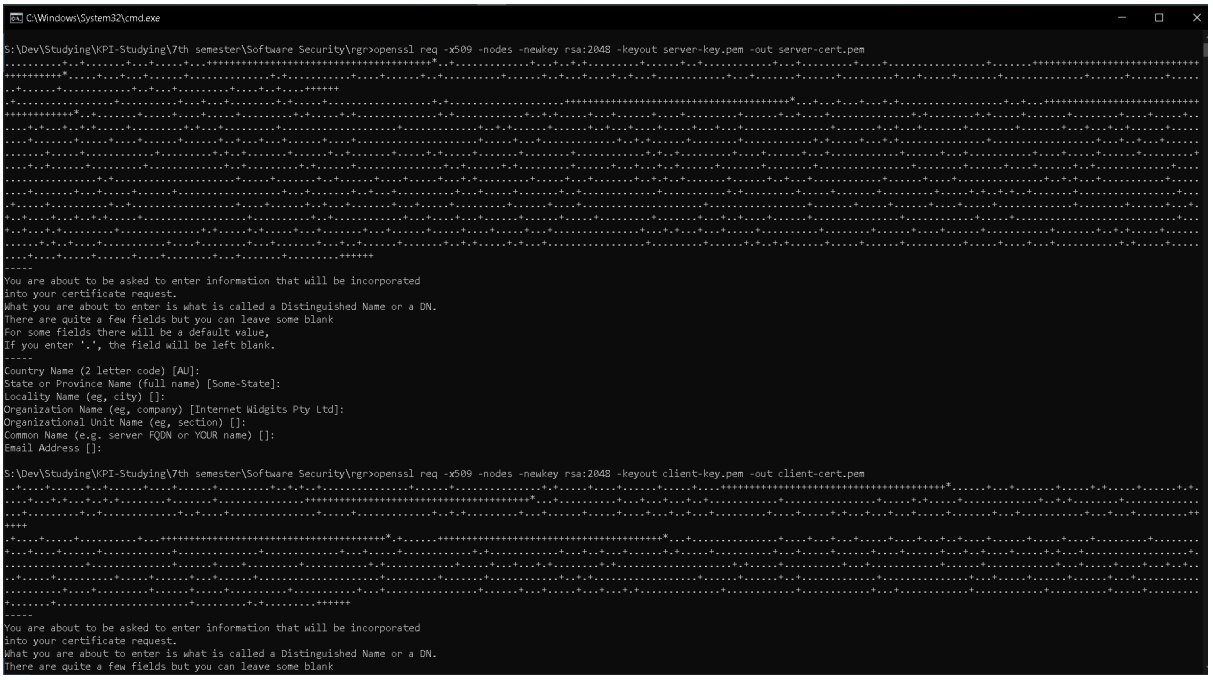
```
});
server.listen(8080, () => {
    console.log('Server listening on port 8080');
});
```

Лістинг client.js:

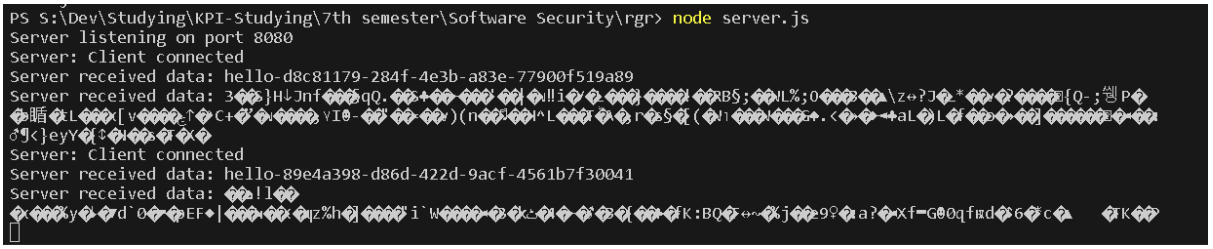
```
const tls = require('tls');
const fs = require('fs');
const crypto = require('crypto');
const uuid = require('uuid');
const options = {
    key: fs.readFileSync('client-key.pem'),
    cert: fs.readFileSync('client-cert.pem'),
};
process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0";
const socket = tls.connect(8080, 'localhost', options, () => {
    console.log('Client: Connected to server');
    const randomUuid = uuid.v4();
    // Send the "hello" message with the random UUID to the server
    socket.write(`hello-${randomUuid}`);
    // Generate a premaster secret (for demonstration purposes, using a simple
string)
    const premasterSecret = 'ThisIsAPremasterSecret';

    console.log('Client: Encrypting premaster secret:', premasterSecret);

    // Encrypt the premaster secret using the server's public key
    const encryptedPremasterSecret = crypto.publicEncrypt(
        {
```

```
            key: fs.readFileSync('server-cert.pem'),  // Server's public key
            padding: crypto.constants.RSA_PKCS1_PADDING,
        },
        Buffer.from(premasterSecret)
    );
    socket.write(encryptedPremasterSecret);
    socket.on('data', (data) => {
        console.log('Client received data:', data.toString());
    });
    socket.on('end', () => {
        console.log('Client: Connection closed');
    });
});
```

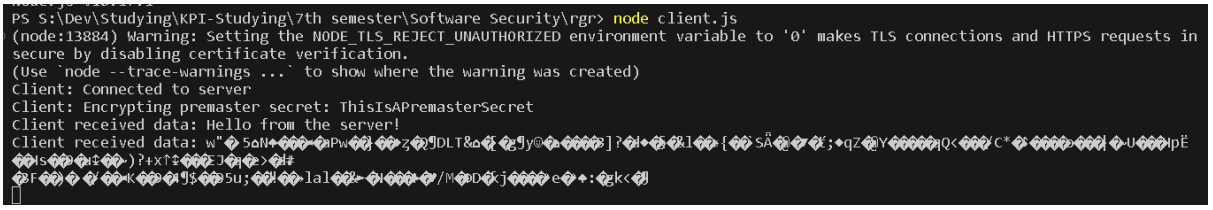
Генерація серверного / клієнтського SSL-сертифікатів за допомоги openssl:



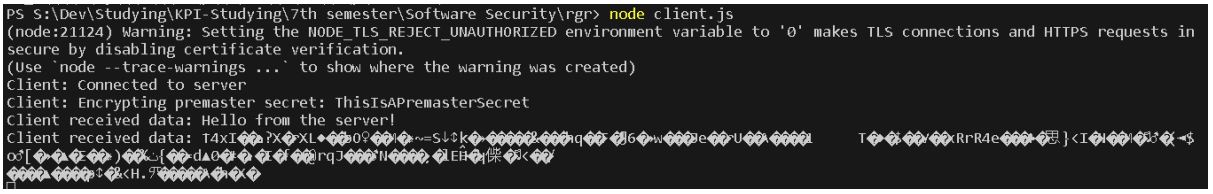
Результат запуску серверу:



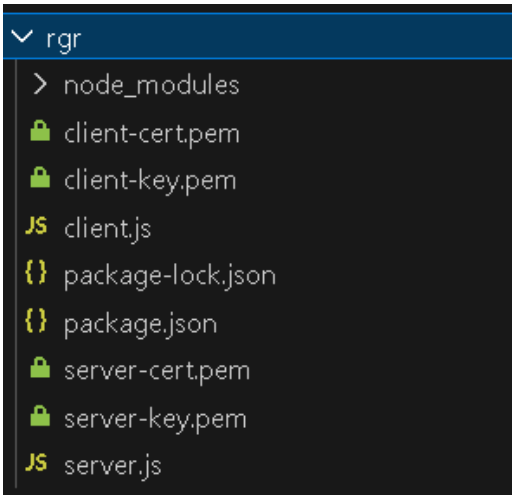
Результат запуску клієнту 1:



Результат запуску клієнту 2:



Структура проекту:



Теоретична частина:

1. Ініціювання клієнтом:

Теорія:

Рукостискання TLS/SSL розпочинається з ініціювання клієнтом підключення.

Реалізація:

У файлі client.js клієнт встановлює TLS-з'єднання із сервером за допомогою методу tls.connect.

2. Відповідь сервера:

Теорія:

Сервер відповідає на ініціювання клієнта з'єднанням.

Реалізація:

У файлі server.js сервер слухає вхідні підключення за допомогою tls.createServer.

3. Повідомлення "Привіт":

Теорія:

Клієнт та сервер обмінюються повідомленнями "привіт" у рамках процесу рукостискання.

Реалізація:

Клієнт ініціює рукостискання, надсилаючи повідомлення "привіт" із випадковим UUID.

Сервер відповідає повідомленням "привіт".

4. Обмін секретом premaster:

Теорія:

Клієнт генерує секрет premaster і шифрує його за допомогою публічного ключа сервера.

Сервер розшифровує секрет premaster за допомогою свого приватного ключа.

Реалізація:

Клієнт генерує секрет premaster і шифрує його у файлі client.js.

Сервер розшифровує секрет premaster у файлі server.js за допомогою приватного ключа сервера.

5. Генерація ключа сеансу:

Теорія:

Сервер та клієнт використовують розшифрований секрет premaster для створення ключа сеансу.

Реалізація:

У файлі server.js сервер використовує розшифрований секрет premaster для генерації ключа сеансу.

6. Обмін зашифрованими повідомленнями:

Теорія:

Після рукостискання клієнт та сервер можуть обмінюватися зашифрованими повідомленнями за допомогою отриманого ключа сеансу.

Реалізація:

У файлах client.js та server.js обмінюються зашифрованими повідомленнями, використовуючи отриманий ключ сеансу.

7. Завершення та захищена комунікація:

Теорія:

Рукостискання завершується, і подальша комунікація захищена за допомогою ключа сеансу.

Реалізація:

Після обміну повідомленнями "готовий" у файлах client.js та server.js комунікація продовжується, використовуючи отриманий ключ сеансу.

Ці кроки реалізації охоплюють основні аспекти базового процесу рукостискання TLS/SSL, включаючи обмін ключами, шифрування та захищену комунікацію.

Цей проект дозволяє отримати базове розуміння процесу рукостискання TLS/SSL. Відповідно до теоретичних принципів, клієнт та сервер успішно обмінюються ключами, встановлюють безпечне з'єднання та продовжують захищену комунікацію. Однак слід зауважити, що це лише спрощений приклад для освітніх цілей, і в реальних сценаріях слід використовувати більш продумані та безпечні практики, такі як використання сертифікатів від довірених центрів сертифікації та обробка помилок.