Integrantes

Juliana Sofía Ahumada Arcos	201921471
Karen Tatiana Vera Hernández	202113341
Juan David Arango Pulido	201911373

Caso No. 3

Informe Final

Tabla de Contenido

Generación de Llaves

Para generar las llaves necesarias, se calcula una llave maestra mediante el algoritmo de Diffie – Hellman.

En primer lugar, se obtiene el valor de p y g utilizando openssl. Se ejecutó el comando openssl dhparam -text 1024 y se obtuvieron los siguientes parámetros p y g:

```
.....................+.......+.....................................+.........................
.....................+.++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++
    DH Parameters: (1024 bit)
    P:
        00:d4:9a:7a:d8:53:f4:84:57:0e:18:11:cc:99:d2:
        85:d3:db:6b:ea:6e:f8:ec:f6:d2:45:05:85:90:d8:
        ea:a7:86:1a:51:2a:d0:5b:54:16:03:3a:f2:37:97:
        0e:32:d4:ac:b3:b2:71:b1:00:9d:96:f4:23:7c:35:
        78:1a:54:f7:ef:d6:6f:7c:06:a1:25:c2:10:23:a2:
        70:21:39:08:83:61:32:c9:d4:11:51:63:4e:45:c9:
        57:01:8a:23:3a:59:19:c5:ba:fd:9e:be:33:51:f8:
        4e:5f:56:23:b3:c8:4a:a9:20:04:39:9e:81:37:ac:
        8d:0d:2f:2a:7c:9a:38:bb:57
    G:    2 (0x2)
    recommended-private-length: 175 bits
-----BEGIN DH PARAMETERS-----
MIGLAoGBANSaethT9IRXDhgRzJnShdPba+pu+Oz20kUFhZDY6qeGGlEq0FtUFgM6
8jeXDjLUrLOycbEAnZb0I3w1eBpU9+/Wb3wGoSXCECOicCE5CINhMsnUEVFjTkXJ
VwGKIzpZGcW6/Z6+M1H4Tl9WI7PISqkgBDmegTesjQ0vKnyaOLtXAgECAgIArw==
-----END DH PARAMETERS-----

$
```

Ilustración 1. Parámetros Diffie – Hellman obtenidos en openssl.

Teniendo los parámetros obtenidos por openssl, se realiza una nueva clase llamada DiffieHellman, en donde se realiza el cálculo de y ($g^x \ mod \ p$) y el cálculo de z ($y^x \ mod \ p$):

```
public class DiffieHellman {
    private BigInteger p = new
BigInteger("D49A7AD853F484570E1811CC99D285D3DB6BEA6EF8ECF6D245058590D8EAA7861A512
AD05B5416033AF237970E32D4ACB3B271B1009D96F4237C35781A54F7EFD66F7C06A125C21023A270
213908836132C9D41151634E45C957018A233A5919C5BAFD9EBE3351F84E5F5623B3C84AA92004399
E8137AC8D0D2F2A7C9A38BB57", 16);
    private BigInteger g = new BigInteger("2");

    public BigInteger getP(){
        return p;
    }

    public BigInteger getG(){
        return g;
    }

    public BigInteger calcularmodp(BigInteger x){
        return g.modPow(x, p);
    }

    public BigInteger calcularz(BigInteger y, BigInteger x) {
        return y.modPow(x, p);
    }
}
```

Ilustración 2. Clase DiffieHellman para calcular llave maestra.

Ahora, teniendo los métodos implementados en la clase DiffieHellman, el cliente y el servidor genera un número x aleatorio (secreto) que será usado para calcular el y de la siguiente manera:

```
public static BigInteger[] generary(){
    DiffieHellman diffieHellman = new DiffieHellman();

    BigInteger p = diffieHellman.getP();

    BigInteger max = p.subtract(BigInteger.ONE);
    Random random = new Random();
    BigInteger x = new BigInteger(max.bitLength(), random);
    while (x.compareTo(max) >= 0) {
        x = new BigInteger(max.bitLength(), random);
    }

    BigInteger y = diffieHellman.calcularmodp(x);
    BigInteger[] valores = new BigInteger[2];
    valores[0] = y;
    valores[1] = x;

    return valores;
}
```

Ilustración 3. Método para generar valores x y y.

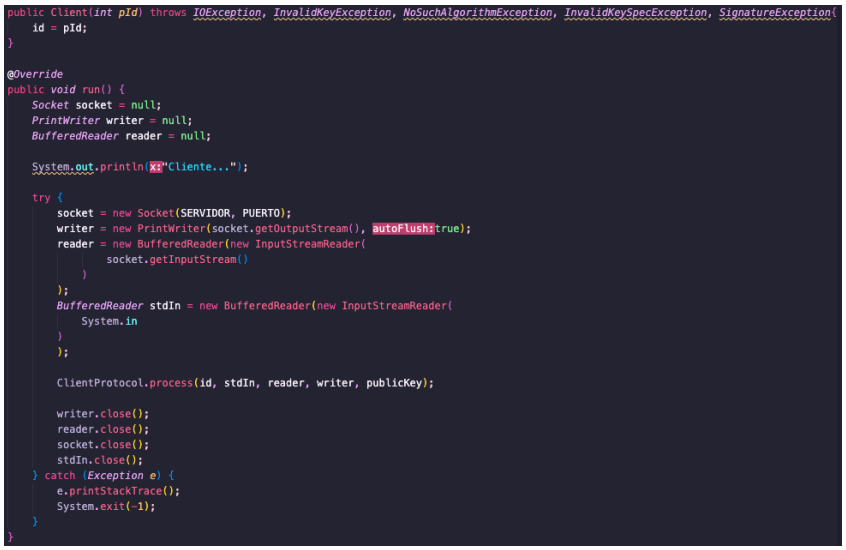
Por último, el cliente y el servidor intercambian los valores y propios y generan la llave maestra z utilizando el y del otro.

Organización de archivos dentro del código

App.java
Clase principal para correr el código.

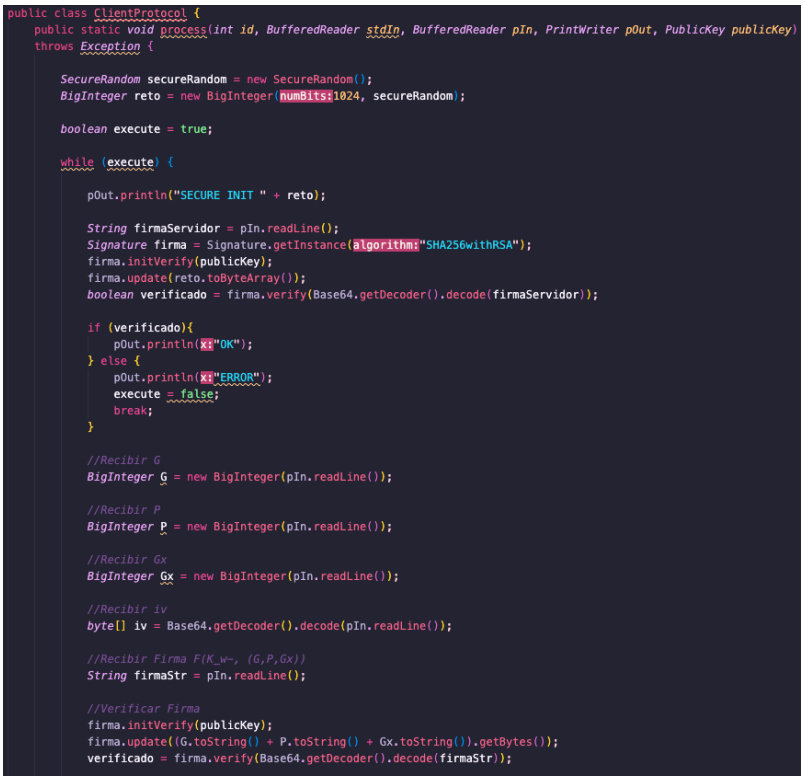
```
public class App {
    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(System.in);
        System.out.print(s:"Por favor, ingresa la cantidad de delegados deseados: ");
        int delegados = scanner.nextInt();

        Server server = new Server();
        server.start();

        Client.publicKey = Server.publicKey;

        for (int i = 0; i < delegados; i++){
            Client client = new Client(pId:0);
            client.start();
        }
        scanner.close();
    }
}
```

Client.java
Clase cliente utilizada para comunicarse con la clase servidor.



```
public Client(int pId) throws IOException, InvalidKeyException, NoSuchAlgorithmException, InvalidKeySpecException, SignatureException{
    id = pId;
}

@Override
public void run() {
    Socket socket = null;
    PrintWriter writer = null;
    BufferedReader reader = null;

    System.out.println(x:"Cliente...");

    try {
        socket = new Socket(SERVIDOR, PUERTO);
        writer = new PrintWriter(socket.getOutputStream(), autoFlush:true);
        reader = new BufferedReader(new InputStreamReader(
                socket.getInputStream()
            )
        );
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(
            System.in
        )
        );

        ClientProtocol.process(id, stdIn, reader, writer, publicKey);

        writer.close();
        reader.close();
        socket.close();
        stdIn.close();
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(-1);
    }
}
```

ClientProtocol.java
Clase que implementa el protocolo Cliente – Servidor.



```
public class ClientProtocol {
    public static void process(int id, BufferedReader stdIn, BufferedReader pIn, PrintWriter pOut, PublicKey publicKey)
    throws Exception {

        SecureRandom secureRandom = new SecureRandom();
        BigInteger reto = new BigInteger(numBits:1024, secureRandom);

        boolean execute = true;

        while (execute) {

            pOut.println("SECURE INIT " + reto);

            String firmaServidor = pIn.readLine();
            Signature firma = Signature.getInstance(algorithm:"SHA256withRSA");
            firma.initVerify(publicKey);
            firma.update(reto.toByteArray());
            boolean verificado = firma.verify(Base64.getDecoder().decode(firmaServidor));

            if (verificado){
                pOut.println(x:"OK");
            } else {
                pOut.println(x:"ERROR");
                execute = false;
                break;
            }

            //Recibir G
            BigInteger G = new BigInteger(pIn.readLine());

            //Recibir P
            BigInteger P = new BigInteger(pIn.readLine());

            //Recibir Gx
            BigInteger Gx = new BigInteger(pIn.readLine());

            //Recibir iv
            byte[] iv = Base64.getDecoder().decode(pIn.readLine());

            //Recibir Firma F(K_w-, (G,P,Gx))
            String firmaStr = pIn.readLine();

            //Verificar Firma
            firma.initVerify(publicKey);
            firma.update((G.toString() + P.toString() + Gx.toString()).getBytes());
            verificado = firma.verify(Base64.getDecoder().decode(firmaStr));
```

DiffieHellman.java
Clase que calcula las funciones módulo.

```java
public class DiffieHellman {
    private BigInteger p = new
BigInteger("D49A7AD853F484570E1811CC99D285D3DB6BEA6EF8ECF6D245058590D8EAA7861A512
AD05B5416033AF237970E32D4ACB3B271B1009D96F4237C35781A54F7EFD66F7C06A125C21023A270
213908836132C9D41151634E45C957018A233A5919C5BAFD9EBE3351F84E5F5623B3C84AA92004399
E8137AC8D0D2F2A7C9A38BB57", 16);
    private BigInteger g = new BigInteger("2");

    public BigInteger getP(){
        return p;
    }

    public BigInteger getG(){
        return g;
    }

    public BigInteger calcularmodp(BigInteger x){
        return g.modPow(x, p);
    }

    public BigInteger calcularz(BigInteger y, BigInteger x) {
        return y.modPow(x, p);
    }
}
```

Server.java
Clase servidor utilizada para comunicarse con la clase ServerHandler.

```java
public Server() throws IOException {
    System.out.println(x:"Inicio del servidor prinicipal");

    try {
        serverSocket = new ServerSocket(PORT);
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(-1);
    }

    ServerHandler.setGP(G, P);

    try {
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(algorithm:"RSA");
        keyPairGenerator.initialize(keysize:1024);
        KeyPair keyPair = keyPairGenerator.generateKeyPair();

        privateKey = keyPair.getPrivate();
        publicKey = keyPair.getPublic();

        System.out.println(privateKey);
        System.out.println(Server.getPublicKey());

        ServerHandler.setKeys(privateKey, publicKey);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

@Override
public void run() {

    while (exceute) {
        Socket socket;
        try {
            socket = serverSocket.accept();

            ServerHandler serverHandler = new ServerHandler(socket, threadsNumber);
            threadsNumber++;

            serverHandler.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ServerHandler.java
Clase utilizada para comunicarse con la clase ServerProtocol.

ServerProtocol.java
Clase que implementa el protocolo Servidor – Cliente.

Instrucciones para correr el servidor y el cliente

Para correr el código, se debe compilar y ejecutar la clase App.java. Luego se ingresa el número de clientes delegados en la consola y se espera a la finalización de la ejecución.

Resultados

Preguntas

1. En el protocolo descrito el cliente conoce la llave pública del servidor (K_w+). ¿Cuál es el método comúnmente usado para obtener estas llaves públicas para comunicarse con servidores web?

El método comúnmente utilizado para obtener la llave pública de un servidor web es a través del intercambio de certificados SSL/TLS durante el proceso de handshake SSL/TLS al establecer una conexión segura. Cuando un cliente se conecta a un servidor web a través de HTTPS, el servidor envía su certificado digital al cliente durante la fase de inicio de sesión (handshake). Este certificado contiene la clave pública del servidor, así como otra información sobre la identidad del servidor, como el nombre de dominio y la autoridad de certificación que emitió el certificado.

El cliente verifica la autenticidad del certificado del servidor comprobando su firma digital utilizando las claves públicas de las autoridades de certificación de confianza almacenadas localmente. Si la firma es válida y el certificado es de una autoridad de certificación confiable, el cliente extrae la clave pública del servidor del certificado y la utiliza para cifrar los datos que se enviarán al servidor.

Este proceso asegura que el cliente esté comunicándose con el servidor correcto y que la comunicación esté cifrada de forma segura utilizando la clave pública del servidor para establecer una conexión segura.

2. ¿Por qué es necesario cifrar G y P con la llave privada?

Se cifra la concatenación de los valores de G, P y G^x con la clave privada del servidor para asegurar la integridad y la autenticidad de estos valores durante el intercambio entre el servidor y el cliente.

Integridad

Cifrar los parámetros G y P junto con G^x garantiza que estos valores no sean modificados durante la transmisión. Si un atacante intenta modificar estos valores en tránsito, la firma digital del servidor no coincidirá, lo que indicará una posible manipulación de los parámetros.

Autenticación
La firma digital de los parámetros del servidor utilizando su clave privada permite al cliente verificar la autenticidad del servidor. Si la firma se puede verificar correctamente utilizando la clave pública del servidor, el cliente puede confiar en que los parámetros provienen del servidor legítimo y no han sido alterados por un atacante.

3. El protocolo Diffie – Hellman garantiza “Forward Secrecy”, presente un caso en el contexto del sistema Banner de la Universidad donde sería útil tener esta garantía, justifique su respuesta (por qué es útil en ese caso).

"Forward Secrecy" es una propiedad de seguridad en la que las claves de encriptación no comprometen la seguridad de las comunicaciones pasadas. En el contexto del sistema Banner de la Universidad, donde se manejan datos sensibles de estudiantes, profesores, personal y administración, la garantia de Forward Secrecy sería extremadamente útil.
Por ejemplo, un escenario serio, en el que un atacante logra comprometer las claves privadas utilizadas para cifrar las comunicaciones entre el servidor Banner y los clientes (por ejemplo, los navegadores web de los usuarios). Sin Forward Secrecy, esto significaría que todas las comunicaciones anteriores que fueron cifradas utilizando esas claves privadas también estarían en riesgo de ser descifradas por el atacante.

Sin embargo, si se implementa Forward Secrecy utilizando un protocolo como Diffie-Hellman, incluso si un atacante logra comprometer las claves de sesión utilizadas en una conexión específica, no podrá descifrar las comunicaciones pasadas que se cifraron con otras claves de sesión. Cada conexión utiliza una clave de sesión única generada mediante el intercambio de claves de Diffie-Hellman, lo que garantiza que las comunicaciones pasadas permanezcan seguras incluso si las claves de sesión actuales on comprometidas.

Por lo tanto, en el contexto del sistema Banner de la Universidad, donde la privacidad y la seguridad de los datos son fundamentales, la garantía de Forward Secrecy proporcionada por el protocolo Diffie-Hellman sería esencial para proteger las comunicaciones pasadas y futuras de los usuarios, incluso en caso de compromiso de las claves de sesión actuales. Esto ayudaría a mitigar el impacto de posibles violaciones de seguridad y a proteger la confidencialidad de la información de la universidad y sus miembros.

Tiempos de demora en diferentes escenarios del cliente

Para realizar la medición del tiempo, se tomaron los escenarios para 4, 16, 32 y 64 clientes delegados.

1. Verificación de firma

Clientes	4	16	32	64
Tiempo (seg)	0.005938	0.01643729	0.03236688	0.1040744

2. Cálculo de G^y

Clientes	4	16	32	64
Tiempo (seg)	0.006492694	0.027267716	0.013218075	0.10408775

3. Cifrado de la consulta

Clientes	4	16	32	64
Tiempo (seg)	0.000800057	0.00003808098	0.000477903	0.000373941

4. Generación del código de autenticación

Clientes	4	16	32	64
Tiempo (seg)	0.005520872	0.013979564	0.01808997	0.0371345

Tiempos de demora en diferentes escenarios del servidor

Para realizar la medición del tiempo, se tomaron los escenarios para 4, 16, 32 y 64 clientes delegados. Los resultados presentados aquí corresponden al promedio de los resultados obtenidos durante las pruebas del código.

1. Generación de firma

Clientes	4	16	32	64
Tiempo (seg)	0.038915	0.228693284	0.045007213	0.054613078

2. Descifrado de la consulta

Clientes	4	16	32	64
Tiempo (seg)	5.47695E-4	0.00002175998	0.000270681	0.000335

3. Verificación del código de autenticación

Clientes	4	16	32	64
Tiempo (seg)	0.001009489	0.0018735862	0.00384673	0.028879423

Gráficas y Tablas

A continuación, se presentan los resultados obtenidos:

Cliente	4	16	32	64
Verificación de firma	0.005938	0.01643729	0.03236688	0.1040744
Cálculo de G^y	0.006492694	0.027267716	0.013218075	0.10408775
Cifrado de la consulta	0.000800057	0.00003808098	0.000477903	0.000373941
Generación del código de autenticación	0.005520872	0.013979564	0.01808997	0.0371345

Servidor	4	16	32	64
Generación de firma	0.038915	0.228693284	0.045007213	0.054613078
Descifrado de la consulta	5.47695E-4	0.00002175998	0.000270681	0.000335
Verificación del código de autenticación	0.001009489	0.0018735862	0.00384673	0.028879423

Análisis de resultados

Como podemos observar, a medida que aumenta el numero de conexiones concurrentes al servidor, los tiempos de operación empiezan a aumentar significativamente del lado del servidor, pues debe compartir sus recursos para atender todas las peticiones. Del lado de los clientes, los tiempos de operación no parecen tener un aumento tan drástico como lo presenta el servidor, lo cual es lógico, pues en un caso del mundo real, estos son maquinas diferentes, que no les importa el número de clientes concurrentes.

Cálculos adicionales

Velocidad de procesador

Las pruebas fueron realizadas en un Ryzen 5 3500 U, el cual cuenta con una velocidad base de 2.1 GHz. Sin embargo, para estas pruebas se utilizó un mono núcleo bajo Overclock, que alcanzo velocidades de $3.7\ GHz$.

Esta velocidad se traduce a $3700000000\ ciclo/seg$.

Consultas que puede cifrar la máquina

Suponiendo el mejor tiempo promedio obtenido en las pruebas, de $0.000800057\ seg$, para el cifrado de las consultas, a la maquina le tomaría $2.960.210\ ciclos$ por cifrado. Lo que se traduciría a que la maquina puede hacer 1249 cifrados por segundo.

Referencias

Qué es el intercambio de claves Diffie-Hellman y cómo funciona. (2021, junio 24). Ciberseguridad. https://ciberseguridad.com/guias/recursos/intercambio-claves-diffie-hellman/

Tablado, F. (2021, junio 15). Algoritmo Diffie-Hellman. Descripción, funcionamiento y ejemplos. Grupo Atico34; Ático34 Protección de datos para empresas y autónomos. https://protecciondatos-lopd.com/empresas/algoritmo-diffie-hellman/

What is Perfect Forward Secrecy? Definition & FAQs. (2017, abril 22). Avi Networks. https://avinetworks.com/glossary/perfect-forward-secrecy/