Caso 3

Cristian Camilo Rey Vargas - 202116752

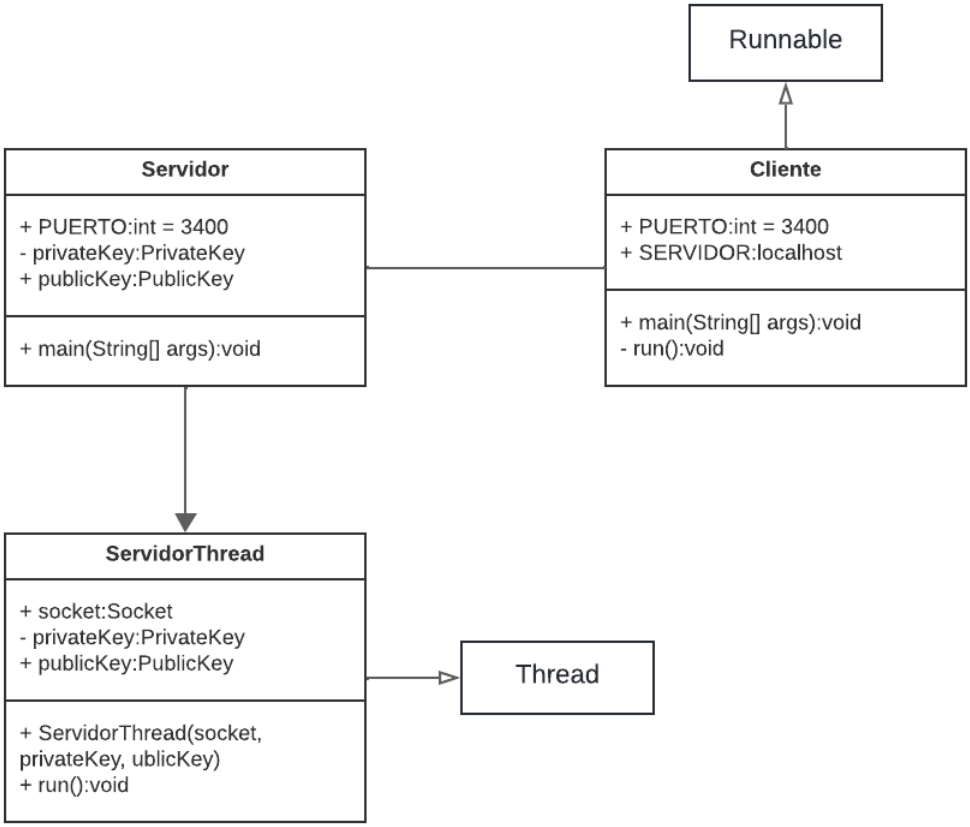
Juan Andres Carrasquilla Gutierrez – 202110183

Santiago Gómez - 202120215

Indicaciones de Ejecución

Para ejecutar el programa, se debe primero ejecutar la clase Servidor, y luego la clase Cliente. Justo después, en la consola de Cliente se le pedirá el número de clientes que quiere ejecutar.

Diagrama de clases



Explicación de las clases y su relacionamiento.

Clase Cliente

Paso 1: Inicio Seguro y Envío del Reto

```
SecureRandom random = new SecureRandom();
BigInteger reto = new BigInteger(numBits:1024, random);
byte[] retoBytes = reto.toByteArray();
out.writeObject(retoBytes);
```

El cliente genera un reto seguro utilizando un número aleatorio grande. Este reto es enviado al servidor para comenzar el proceso de autenticación y garantizar que la conexión es iniciada por un cliente legítimo.

Paso 4: Verificación del Reto

```
byte[] firmaServidor = (byte[]) in.readObject();
Signature signature = Signature.getInstance(algorithm:"SHA256withRSA");
signature.initVerify(publicKey);
signature.update(retoBytes);
String verify = (signature.verify(firmaServidor) ? "OK" : "ERROR");
//System.out.println(verify);
out.writeObject(verify);
verify = "";
```

El cliente recibe y verifica la firma del reto enviado por el servidor utilizando la clave pública del servidor. Esto asegura que el mensaje no ha sido alterado y que el servidor es legítimo.

Paso 5 y 9: Verificación de la Firma del DH y Respuesta del Servidor

```
signature.update(g.toByteArray());
signature.update(p.toByteArray());
signature.update(gxmodp.toByteArray());
verify = (signature.verify(firmaDH) ? "OK" : "ERROR");
//System.out.println(verify);
out.writeObject(verify);
```

El cliente verifica la firma de los parámetros de Diffie-Hellman (DH) proporcionados por el servidor. Esto incluye el generador g, el primo p, y gxmodp. La verificación de la firma asegura que los parámetros no han sido alterados.

Paso 8: Verificación de Claves

```
DHParameterSpec dhSpecClient = new DHParameterSpec(p, g);
KeyPairGenerator clientKeyPairGen = KeyPairGenerator.getInstance(algorithm:"DH");
clientKeyPairGen.initialize(dhSpecClient);
KeyPair clientKeyPair = clientKeyPairGen.generateKeyPair();
BigInteger gymodp = ((DHPublicKey) clientKeyPair.getPublic()).getY();
out.writeObject(gymodp);
```

El cliente genera su parte del intercambio de claves DH (g^y mod p) y la envía al servidor. Esto forma parte del establecimiento de una clave compartida de manera segura.

Paso 11b: Generación de Clave Simétrica y MAC

```
KeyAgreement keyAgreement = KeyAgreement.getInstance("DH");
keyAgreement.init(clientKeyPair.getPrivate());
DHPublicKeySpec dhPubKeySpec = new DHPublicKeySpec(gxmodp, p, g);
PublicKey serverPublicKey = KeyFactory.getInstance(algorithm:"DH").generatePublic(dhPubKeySpec);
keyAgreement.doPhase(serverPublicKey, true);
byte[] clientSecret = keyAgreement.generateSecret();
SecretKey clientAesKey = new SecretKeySpec(clientSecret, 0, 16, "AES");
```

El cliente utiliza la parte pública del servidor para completar el intercambio DH y generar una clave compartida. Esta clave se utiliza para cifrar la comunicación subsiguiente.

Paso 13, 14, 17, 18: Envío y Verificación de Credenciales y Consultas

```
String credentials = "usuario:contrasena";
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
SecretKeySpec keySpec = new SecretKeySpec(aesKey.getEncoded(), "AES");
IvParameterSpec ivSpec = new IvParameterSpec(iv);
cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
byte[] encryptedCredentials = cipher.doFinal(credentials.getBytes());
out.writeObject(encryptedCredentials);
System.out.println(x:"Credenciales enviadas exitosamente.");
```

El cliente cifra y envía las credenciales al servidor utilizando AES en modo CBC. Las credenciales incluyen el usuario y la contraseña, asegurando que solo el servidor legítimo pueda descifrarlas.

Paso 21: Verificación Final de la Respuesta

```
byte[] encryptedResponse = (byte[]) in.readObject();
byte[] hmacResponse = (byte[]) in.readObject();
cipher.init(Cipher.DECRYPT_MODE, aesKey, new IvParameterSpec(iv));
byte[] decryptedResponse = cipher.doFinal(encryptedResponse);
System.out.println("Respuesta recibida y descifrada: " + new String(decryptedResponse));

mac.init(hmacSha256Key);
byte[] calculatedHmac = mac.doFinal(decryptedResponse);

if (MessageDigest.isEqual(hmacResponse, calculatedHmac)) {
    System.out.println(x:"HMAC verificado con éxito.");
} else {
    System.out.println(x:"Error de verificación HMAC.");
}
```

El cliente recibe y verifica la respuesta cifrada y su HMAC del servidor. Esto confirma la integridad y la autenticidad de la respuesta, completando el ciclo de solicitud y respuesta seguras.

En resumen, en la clase cliente se hace:

Conexión y Autenticación Inicial: El cliente inicia la comunicación con el servidor, recibiendo la clave pública del servidor para verificar la autenticidad de este mediante la firma de un reto.

Establecimiento de la Clave Segura: Utiliza el protocolo Diffie-Hellman para generar una clave compartida de forma segura. Esto permite que tanto el cliente como el servidor tengan una clave secreta común sin haberla transmitido por la red.

Cifrado y Envío de Credenciales: El cliente cifra sus credenciales utilizando la clave derivada del intercambio Diffie-Hellman y las envía al servidor para su verificación.

Consulta Segura: Una vez autenticado, el cliente puede realizar consultas seguras al servidor. Estas consultas se cifran y se envían junto con un HMAC para garantizar su integridad y autenticidad.

Recepción y Verificación de Respuestas: El cliente recibe respuestas cifradas del servidor, las descifra y verifica su integridad mediante la comparación de HMACs, asegurando que la respuesta no ha sido alterada en tránsito.

Clase ServidorThread

Paso 2: Recepción del Reto y Cálculo de la Respuesta

```
byte[] retoBytes = (byte[]) in.readObject();
Signature signature = Signature.getInstance(algorithm:"SHA256withRSA");
signature.initSign(privateKey);
signature.update(retoBytes);
byte[] firma = signature.sign();
out.writeObject(firma);
```

El servidor recibe el reto del cliente, lo firma con su clave privada RSA, y envía la firma de vuelta al cliente. Esto asegura al cliente que está comunicándose con el servidor legítimo.

Paso 3, 6, 7: Generación y Envío de Parámetros DH

```
System.out.println(x:"OK recibido");
// Establecimiento de la conexión segura
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance(algorithm:"DH");
keyPairGen.initialize(keysize:1024);
KeyPair serverKeyPair = keyPairGen.generateKeyPair();
DHPublicKey serverPublicKey = (DHPublicKey) serverKeyPair.getPublic();
BigInteger g = serverPublicKey.getParams().getG();
BigInteger p = serverPublicKey.getParams().getP();
BigInteger gxmodp = serverPublicKey.getY();

SecureRandom random = new SecureRandom();
byte[] iv = new byte[16];
random.nextBytes(iv);

out.writeObject(g);
out.writeObject(p);
out.writeObject(gxmodp);
out.writeObject(iv);
```

El servidor genera un par de claves Diffie-Hellman y envía los parámetros g,p y g^x mod p al

cliente. Esto establece las bases para que ambos participantes generen una clave compartida de forma segura.

Paso 11a: Cálculo de la Clave Compartida

```
System.out.println(x:"OK recibido");
BigInteger gymodp = (BigInteger) in.readObject();
KeyAgreement keyAgreement = KeyAgreement.getInstance("DH");
keyAgreement.init(serverKeyPair.getPrivate());
DHPublicKeySpec dhPubKeySpec = new DHPublicKeySpec(gymodp, p, g);
PublicKey clientPublicKey = KeyFactory.getInstance(algorithm:"DH").generatePublic(dhPubKeySpec);
keyAgreement.doPhase(clientPublicKey, true);
byte[] serverSecret = keyAgreement.generateSecret();
SecretKey serverAesKey = new SecretKeySpec(serverSecret, 0, 16, "AES");
```

El servidor calcula la clave compartida utilizando la parte pública de Diffie-Hellman enviada por el cliente. Esto completa el intercambio de claves Diffie-Hellman.

Paso 12, 15, 16: Envío de 'CONTINUAR' y Verificación de Credenciales

```
out.writeObject(obj:"CONTINUAR");

// Paso 15 y 16: Recibir y verificar credenciales cifradas
byte[] encryptedCredentials = (byte[]) in.readObject();
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
SecretKeySpec keySpec = new SecretKeySpec(aesKey.getEncoded(), "AES");
IvParameterSpec ivSpec = new IvParameterSpec(iv);
cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
byte[] decryptedCredentials = cipher.doFinal(encryptedCredentials);
String credentials = new String(decryptedCredentials);
```

El servidor solicita al cliente que continúe con el envío de las credenciales cifradas, las cuales son luego descifradas y verificadas. Se envía una respuesta basada en la validación de estas credenciales.

```
byte[] consultaCifrada = (byte[]) in.readObject();
byte[] hmacConsultaRecibido = (byte[]) in.readObject();
cipher.init(Cipher.DECRYPT_MODE, aesKey, new IvParameterSpec(iv));
byte[] consultaDescifrada = cipher.doFinal(consultaCifrada);
System.out.println("Consulta recibida y descifrada: " + new String(consultaDescifrada));
Mac mac = Mac.getInstance("HmacSHA256");
mac.init(hmacSha256Key);
byte[] hmacCalculado = mac.doFinal(consultaDescifrada);
if (MessageDigest.isEqual(hmacConsultaRecibido, hmacCalculado)) {
    System.out.println(x:"HMAC verificado con éxito.");
} else {
    System.out.println(x:"Error de verificación HMAC.");

}
```

El servidor recibe y descifra la consulta del cliente. Verifica el HMAC para asegurar la integridad y autenticidad de la consulta. Responde al cliente con la información solicitada y un HMAC de la respuesta para verificación.

Paso 21: Verificación Final de la Respuesta del Cliente

```
String respuesta = "Saldo: $1000";
byte[] respuestaBytes = respuesta.getBytes();
cipher.init(Cipher.ENCRYPT_MODE, aesKey, new IvParameterSpec(iv));
byte[] respuestaCifrada = cipher.doFinal(respuestaBytes);
out.writeObject(respuestaCifrada);

byte[] hmacRespuesta = mac.doFinal(respuestaBytes);
out.writeObject(hmacRespuesta);
System.out.println(x:"Respuesta y HMAC enviados al cliente.");
```

El servidor envía la respuesta cifrada y su HMAC correspondiente al cliente. Esto asegura que la comunicación final entre el cliente y el servidor sea segura y verificable.

En la clase ServidorThread lo que se hizo en general fue:

Recepción y Respuesta al Reto: Al recibir una conexión, el servidor envía su clave pública y responde a un reto para probar su identidad al cliente.

Configuración de la Conexión Segura: Genera y envía los parámetros de Diffie-Hellman necesarios para el establecimiento de una clave compartida segura.

Autenticación de Credenciales: Recibe, descifra y verifica las credenciales del cliente. Responde con un mensaje de éxito o error según el resultado de la verificación.

Procesamiento de Consultas: Recibe consultas cifradas y sus respectivos HMACs del cliente, los verifica para garantizar su integridad y autenticidad, y procesa dichas consultas.

Envío de Respuestas Seguras: Envía las respuestas a las consultas del cliente de manera cifrada, acompañadas de un HMAC para permitir al cliente verificar su integridad.

Clase servidor

```java
public static void main(String[] args) {
    ServerSocket serverSocket = null;
    try {
        serverSocket = new ServerSocket(PUERTO);
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance(algorithm:"RSA");
        keyGen.initialize(keysize:2048);
        KeyPair pair = keyGen.generateKeyPair();
        privateKey = pair.getPrivate();
        publicKey = pair.getPublic();
        System.out.println("Servidor listo y escuchando en el puerto " + PUERTO);
        while (true) {
            Socket socket = serverSocket.accept();
            Thread thread = new Thread(new ServidorThread(socket, privateKey, publicKey));
            thread.start();
        }
    } catch (Exception e) {
        System.out.println("Error al iniciar el servidor: " + e.getMessage());
    } finally {
        try {
            if (serverSocket != null) serverSocket.close();
        } catch (IOException e) {
            System.out.println("Error al cerrar el servidor: " + e.getMessage());
        }
    }
```

Lo que se hace en esta clase es:

Inicialización del Servidor: La clase configura y pone en marcha un servidor en un puerto especificado (3400). Utiliza ServerSocket para escuchar solicitudes de conexión de los clientes.

Generación de Claves: Al inicio, genera un par de claves RSA que incluyen una clave privada y una pública. La clave pública es compartida con los clientes para permitir la verificación de la identidad del servidor y para actividades criptográficas como el cifrado y la firma digital.

Manejo de Conexiones: Por cada conexión entrante, el servidor acepta el socket del cliente y crea un nuevo hilo de ejecución (ServidorThread) que manejará la sesión con dicho cliente. Esto permite al servidor atender múltiples clientes de manera concurrente sin bloquear o interrumpir el servicio a otros clientes.

Seguridad y Concurrencia: A través del uso de Thread, la clase asegura que cada conexión tenga un flujo independiente de ejecución, lo que es esencial para mantener la integridad y la seguridad en el procesamiento de las transacciones y datos entre el servidor y múltiples clientes.

Respuesta a las preguntas

1)

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

- El método más común para obtener llaves públicas es a través de la infraestructura de clave pública (PKI, por sus siglas en inglés), que utiliza certificados digitales emitidos

por autoridades certificadoras (CA) para asegurar y autenticar las comunicaciones digitales entre navegadores web y servidores web. Estos certificados asocian claves públicas con la identidad de sus propietarios verificados, facilitando así un entorno seguro para las transacciones en línea.

ii) ¿Por qué es necesario cifrar G y P con la llave privada?

- Cifrar los componentes como G (el generador) y P (el módulo) en un intercambio de claves Diffie-Hellman utilizando la clave privada es crucial para garantizar la autenticidad y la integridad de estos valores. Al firmar estos componentes con la clave privada, el receptor puede utilizar la clave pública correspondiente para verificar que los valores recibidos no han sido alterados y que provienen de una fuente legítima, asegurando así la fiabilidad del proceso de establecimiento de claves compartidas.

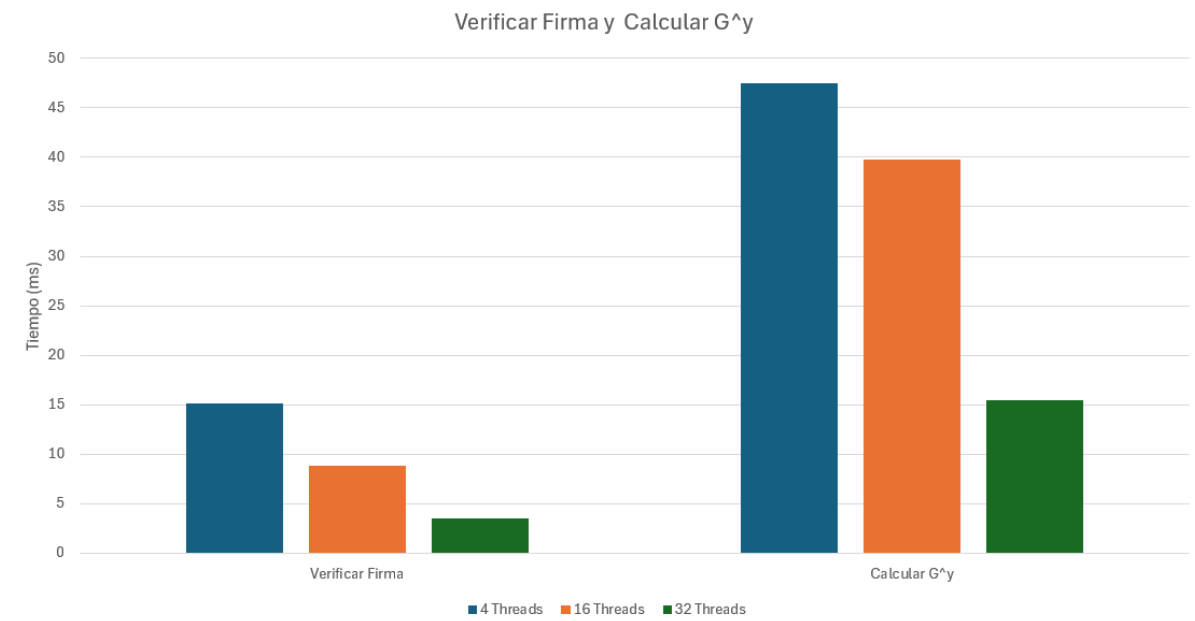
iii) El protocolo Diffie-Hellman garantiza "Forward Secrecy", presente un caso en el contexto del sistema Banner de la Universidad donde sería útil tener esta garantía, justifique su respuesta (por qué es útil en ese caso).

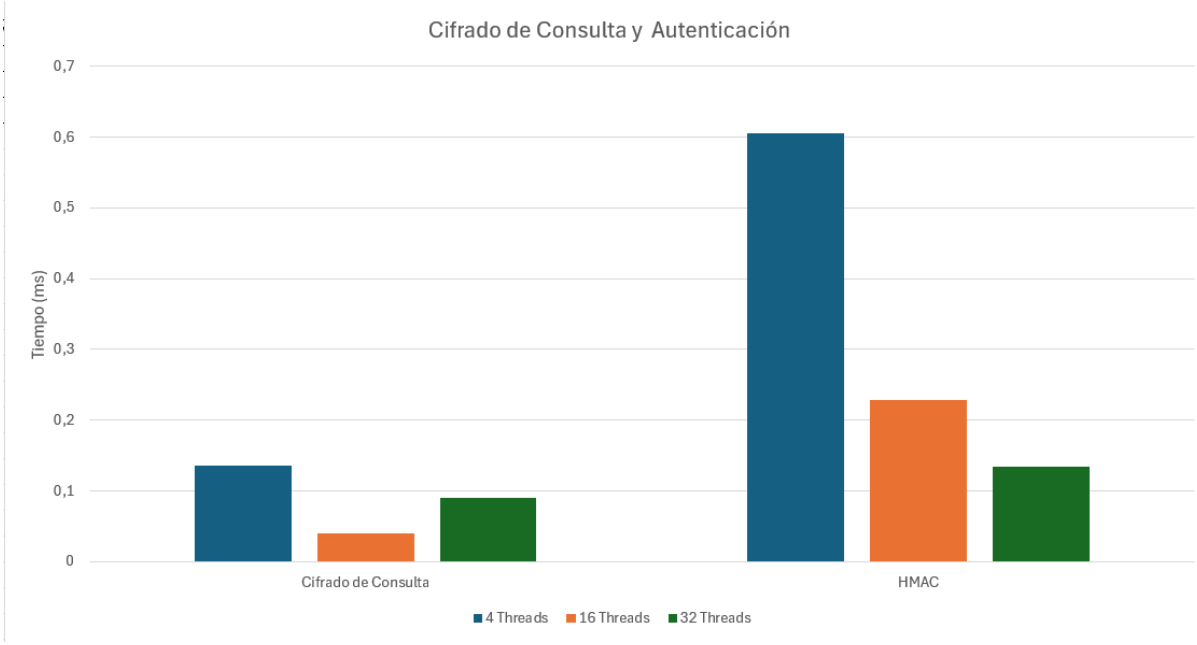
- El "Forward Secrecy", o secreto hacia adelante, garantizado por el protocolo Diffie-Hellman, asegura que las sesiones de comunicación pasadas permanezcan seguras incluso si la clave privada a largo plazo se ve comprometida en el futuro. En el contexto del sistema Banner de una universidad, donde los datos sensibles de los estudiantes, como calificaciones y registros personales, se manejan con frecuencia, el "Forward Secrecy" es crucial. Aun si las claves privadas se comprometen eventualmente, las sesiones de comunicación anteriores, que podrían haber transmitido o modificado información sensible, no pueden descifrarse, manteniendo la confidencialidad de los datos históricos.

ESCENARIOS DE PRUEBA

Tiempos del Cliente:

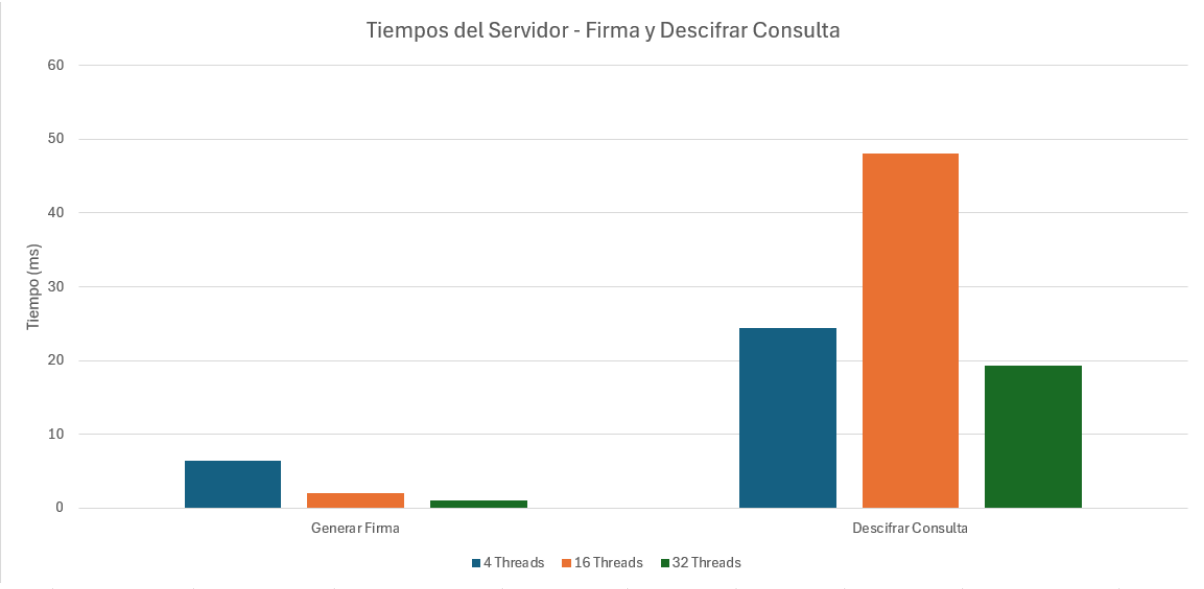
# Threads	Verificar Firma	Calcular G^y	Cifrado de Consulta	HMAC
4	15,14ms	47,45ms	0,1365ms	0,6053ms
16	8,798ms	39,74ms	0,0407ms	0,228ms
32	3,55ms	15,47ms	0,0907ms	0,135ms

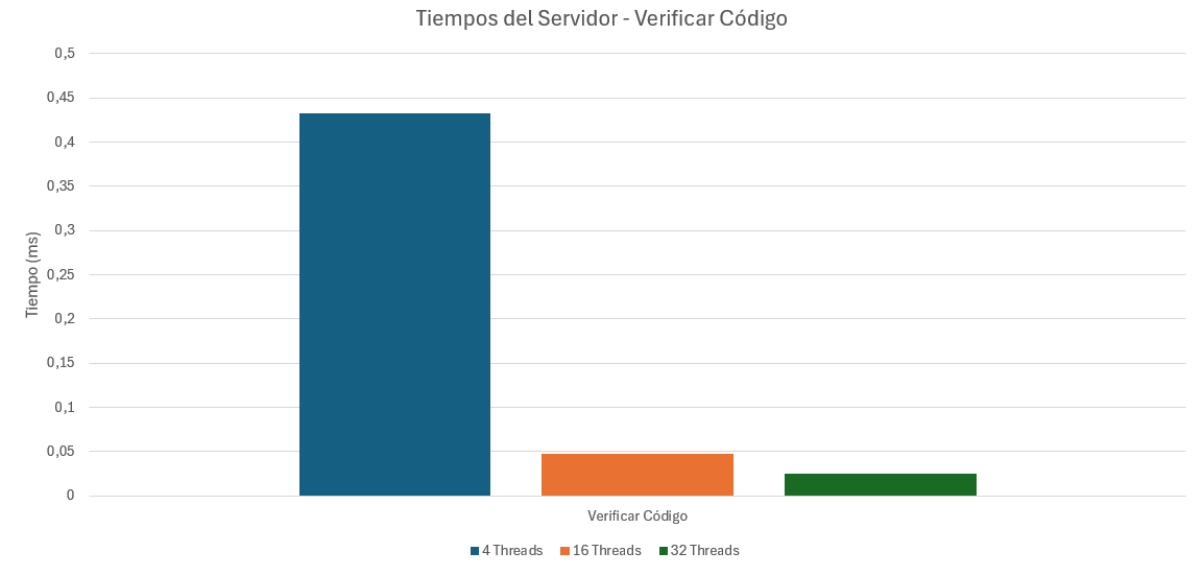




Tiempos del Servidor:

# Threads	Generar Firma	Descifrar Consulta	Verificar Código
4	6,47ms	24,43ms	0,432ms
16	1,99ms	48,1ms	0,048ms
32	1,02ms	19,25ms	0,025ms





Análisis de Resultados

Verificación de Firma y Cálculo de G^y

La verificación de la firma y el cálculo de G^y muestran una mejora significativa en el rendimiento a medida que el número de hilos aumenta. Por ejemplo, el tiempo para verificar firmas disminuye de 15.14ms con 4 hilos a 3.55ms con 32 hilos, y el tiempo para calcular G^y baja de 47.45ms a 15.47ms. Esto indica que estas operaciones se benefician de la paralelización, debido a la capacidad de procesamiento distribuido que permite manejar múltiples tareas simultáneamente sin bloqueo.

Cifrado de Consulta y HMAC:

Los tiempos de cifrado de consultas y generación de HMAC también mejoran con más hilos, aunque el cambio no es tan grande como en las operaciones criptográficas de firma y generación de claves. Esto podría deberse a que estas operaciones son relativamente más sencillas y menos computacionalmente difíciles de ejecutar.

Generación de Firma, Descifrado de Consulta y Verificación de Código en el Servidor:

Similar a los resultados del cliente, el servidor muestra una reducción significativa en el tiempo de generación de firma y descifrado de consultas con el aumento de hilos. Esto debido a que las operaciones de criptografía y descifrado también se escalan bien con el aumento de paralelismo.

La verificación de código tiene tiempos notablemente bajos en comparación con otras operaciones, y muestra una mejora constante con el aumento de hilos.

Estos resultados resaltan la importancia de la concurrencia en aplicaciones que realizan operaciones criptográficas intensivas. Al aumentar la cantidad de subprocesos, puede aprovechar mejor los recursos de hardware, como múltiples núcleos de CPU, para procesar tareas en paralelo, reduciendo así el tiempo de espera y aumentando la eficiencia general del sistema.

Esta eficiencia es importante en un entorno de producción donde el tiempo de respuesta y la capacidad de manejar múltiples solicitudes simultáneamente pueden tener un impacto directo en la experiencia del usuario y la capacidad de escalamiento del sistema. Además, los tiempos de ejecución mejorados con más subprocesos ayudan a mitigar los cuellos de botella en las operaciones de seguridad, lo cual es esencial para mantener tanto el rendimiento como la integridad en aplicaciones seguras.

6. Identifique la velocidad de su procesador, y estime cuántas consultas puede cifrar su máquina, cuántos códigos de autenticación puede calcular y cuántas verificaciones de firma, por segundo. Escriba todos sus cálculos.

Con un procesador de 2.50GHz

$$Ciclos\ de\ CPU\ =\ Tiempo\ en\ ms\ \cdot 10^6 \cdot Frecuencia\ del\ CPU$$

$$Operaciones\ por\ segundo\ =\ \frac{2.5 \cdot 10^9 \frac{ciclos}{segundo}}{ciclos\ por\ operación}$$

1. Verificar firma:
- $Ciclos\ para\ verificar\ firma\ =\ 3.55 \cdot 2.5 \cdot 10^6 = 8.875.000\ ciclos$
- $Firmas\ por\ segundo\ =\ \frac{2.5 \cdot 10^9}{8{,}875{,}000} = 281.69\ \frac{Firmas}{segundo}$
2. Cifrado de Consulta:
- $Ciclos\ para\ cifrado\ de\ consultas = 0.0907 \cdot 2.5 \cdot 10^6 = 226750$
- $Cifrado\ de\ Consulta\ por\ segundo\ = \frac{2.5 \cdot 10^9}{226750} = 11026.35 \frac{Consultas}{segundo}$
3. Códigos por segundo:
- $Ciclos\ HMAC\ = 0.135 \cdot 2.5 \cdot 10^6 = 337500\ ciclos$
- $HMAC\ por\ segundo\ =\ \frac{2.5 \cdot 10^9}{337500} = 7407.41\ \frac{HMAC}{segundo}$

Para el cálculo se tomaron los valores obtenidos en los escenarios de prueba con 32 threads.

Referencias:

https://www.okta.com/identity-101/public-key-infrastructure/

https://security.stackexchange.com/questions/87564/how-does-ssl-tls-pki-work