Josué Vega Valbuena (202213040)
Juan Leonardo Rangel Barrera (202215757)

DOCUMENTACION CASO 3

Organización Del proyecto

Para esta entrega se implementaron un total de 5 clases las cuales están organizadas jerárquicamente de la siguiente manera:

En primer lugar, tenemos la clase ***"Principal.java",*** esta clase es la que se encarga de inicializar los diferentes procesos y delegados de los servidores y clientes. En esta clase tenemos dos métodos principales. GenerateDelegates que es una función que crea una estructura de datos Map, en las cuales sus llaves corresponden a Objetos de tipo Server y sus valores corresponden a Objetos de tipo cliente. Esto se hace para tener una traza de cuantos procesos iniciaremos y como los correremos todos de forma organizada. De este último punto se encarga el método initDelegates, el cual se basa en el recorrido del mapa descrito anteriormente y por cada llave valor. Inicializar los diferentes objetos que se encuentran acá, así se puede tener certeza de que estamos conectando UN SOLO cliente a un solo servidor, en diferentes puertos.

En otra parte tenemos la clase ***Server.java,*** que simula lo que sería el servidor que implementamos para esta entrega. Esta clase extiende de la super clase Thread, lo que la convierte en un proceso del sistema. Esta clase se encarga de las comunicaciones y a las funciones del servidor como tal para las especificaciones del caso. En este caso tenemos diferentes métodos con el objetivo de mejorar y condensar un poco más el código que va a ejecutar este proceso en su método principal, el método run() que iniciaría cuando a la instancia de esta clase se le llame al método start().

Como comunicador a esta clase tenemos la clase ***Cliente.java***, que es el otro proceso que genera comunicaciones con el servidor con el objetivo de cumplir con ciertos requerimientos que especificamos. Esta clase de la misma forma que la clase Server, tiene sus propios métodos para poder ejecutar de forma más fácil el método principal de este proceso, el método run ().

Además, tenemos las clases ***Utilidades AES*** y ***Utilidades RSA***, que guardan en su interior los diferentes métodos para cifrado y descifrado de los mensajes que se enviarán entre el servidor y el cliente. Especificamente en la clase Utilidades AES, tenemos los métodos de cifrado SIMETRICO, con el objetivo de separar funciones y hacer más claro cómo funcionan estos procesos, tenemos específicamente el método encriptar y desencriptar los cuales funcionan con llaves simétricas como lo hace el cifrado SIMETRICO.
Por otro lado, en la clase UtilidadesRSA tenemos los métodos para cifrado asimétrico y adicionalmente el método para la generación de un código HASH, en este caso estamos especificando los métodos encriptar y desencriptar para el cifrado ASIMETRICO entre el servidor y el cliente. Adicionalmente, encontramos la función en

la que, a partir de un mensaje, devolvemos su código HASH, para asegurar integridad para el mensaje.

¿Como Configurarlo?

Para poder correr el programa, tenemos la clase Principal.java, esta clase es la que se encarga de coordinar los diferentes servidores y clientres delegados para que corran.

En este caso tenemos la clase MAIN:

```
Run | Debug
public static void main(String[] args) throws Exception {

    Principal principal = new Principal();

    Scanner scan = new Scanner(System.in);

    System.out.println(x:"");
    System.out.print(s:"Por favor ingrese el numero de Servidores y Clientes que desea correr: ");
    System.out.println(x:"");

    String numeroDelegados = scan.nextLine();

    principal.generateDelegates(Integer.parseInt(numeroDelegados));

    principal.initDelegates();

    scan.close();
```

Esta clase es la que se encarga de poder correr nuestro programa de la forma correcta.

En primer Lugar creamos un objeto de la clase Principal con el objetivo de usarla para generar los delegados de servidores y clientes.

Despues de esto especificamos y le preguntamos al usuario cuanto Servidores y cuantos Clientes quiere delegar para que puedan hacer el proceso que esta descrito en la entrega.

Cuando el cliente ingresa este número, llamamos a la función generateDelegates de la clase Principal:

```
public void generateDelegates(Integer numberOfDelegates) throws NoS
    mapDelegates = new HashMap<>();
    for(int i = 0; i < numberOfDelegates ; i++){
        Server servidor = new Server(1234+i);
        mapDelegates.put(servidor, new Cliente(servidor,1234+i));

    }
}
```

Este método tiene como objetivo principal, poder guardar las diferentes instancias que estamos creando de Server y Cliente en un mapa, esto con el objetivo de poder luego correr estos Objetos como procesos sin ninguna interferencia entre estos.

Luego como observamos en la clase Main, llamamos el metodo de initDelegates:

```
public void initDelegates() throws InterruptedException{

    for (Map.Entry<Server, Cliente> entry : mapDelegates.entrySet()) {
        entry.getKey().start();
        Cliente.sleep(millis:1000);
        entry.getValue().start();
        Server.sleep(millis:2000);
        Cliente.sleep(millis:2000);
        System.out.println(x:"");
    }
```

Este método recorre todo el mapa, antes creado y corremos las instancias de Servidor y Cliente que tener respectivamente en la llave y en el valor de un espacio de la estructura de datos. Cuando empezamos a correr estos procesos creamos una pequeña espera entre el inicio del proceso del servidor y el inicio del proceso del cliente, dado que esperamos a que el servidor inicie el proceso cree el puerto para que el cliente pueda conectarse sin ningún problema.

De esta forma se corren los servidores y clientes concurrentes, para que generen el proceso que esperamos.

Preguntas:

1. En el protocolo descrito el cliente conoce la llave publica del servidor (K_w+). ¿Cuál es el método comúnmente usado para obtener estas llaves públicas para comunicarse con servidores web?
En la comunicación en internet se usa comúnmente los certificados digitales, los cuales son unos ficheros que contienen la clave publica del servidor y ciertos datos adicionales que permiten la verificación de la identidad del servidor (“Certificado digital”, 2024). Estos certificados son firmados electrónicamente por Entidades Certificadoras. Por ejemplo, en HTTPS, el servidor presenta su certificado al cliente, que luego verifica la autenticidad del certificado contra una lista preestablecida de las entidades certificadoras. Este proceso nos asegura que el servidor es legítimo y que la comunicación no ha sido intervenida.

2. ¿Por qué es necesario cifrar G y P con la llave privada?
Hay dos aspectos fundamentales por lo que es necesario hacer el cifrado de G y P con la llave privada. El primero es la integridad; al cifrar con la llave privada, se garantiza que los datos enviados no han sido alterados en tránsito. Cuando el cliente verifica la firma con la llave pública del servidor, también se está comprobando que G y P no han sido modificados desde que fueron cifrados por el servidor. Cualquier alteración en los datos haría que la verificación de la firma fallara. El segundo aspecto es la autenticación; Al cifrar G y P con su llave privada, el servidor asegura

que solo él puede haber generado ese mensaje específico. Cuando el cliente recibe y descifra estos valores usando la llave pública del servidor, puede estar seguro de que realmente provienen del servidor con el que se va a comunicar. (“Notas de clase del curso Infraestructura de Computacional”, 2022)

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).

Supongamos que el servidor que aloja Banner es comprometido y las claves privadas de cifrado son robadas. Estas claves podrían ser usadas para descifrar comunicaciones previamente interceptadas y almacenadas por un atacante.

Si Banner no utiliza un protocolo que asegure Forward Secrecy, entonces todas las sesiones cifradas anteriores que fueron interceptadas podrían ser descifradas utilizando la clave privada comprometida. Esto podría exponer información sensible como las calificaciones de los estudiantes, sus horarios de clases, información personal, etc.

Si Banner implementa Diffie-Hellman, cada sesión entre el servidor y el cliente (por ejemplo, un estudiante o administrador accediendo al sistema) utilizará una clave de sesión única que no puede ser derivada de las claves a largo plazo (como la clave privada del servidor). Esto significa que, incluso si la clave privada del servidor es comprometida en el futuro, las sesiones pasadas no pueden ser descifradas, ya que las claves de sesión no pueden ser reconstruidas. (“Notas de clase del curso Infraestructura de Computacional”, 2022)

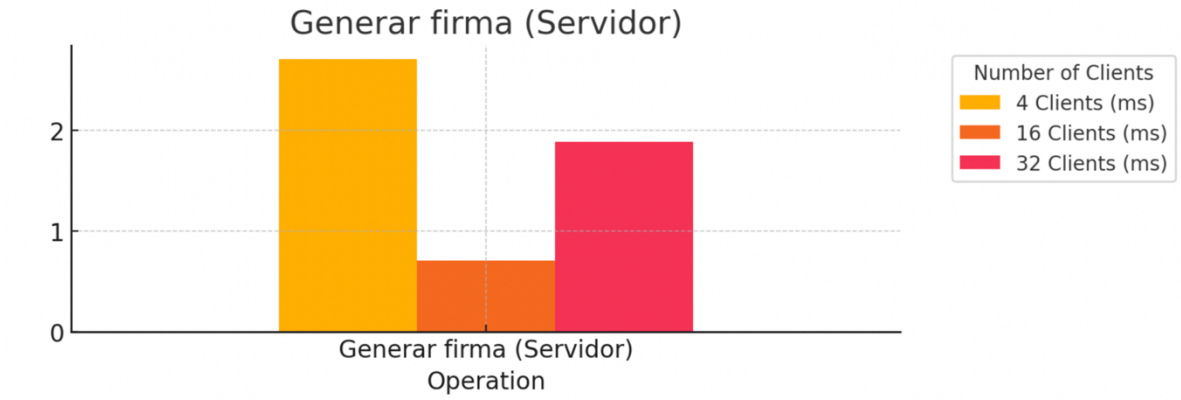
Entonces… ¿Por qué se debería tener implementado Foward Secrety?
Primero, para la universidad, donde la privacidad y la seguridad de los datos académicos son fundamentales, garantizar que las comunicaciones pasadas no puedan ser descifradas incluso si las claves de cifrado son robadas en el futuro es una cuestión de mucha importancia. Foward nos proporciona protección contra robo y manipulación no autorizada de información.

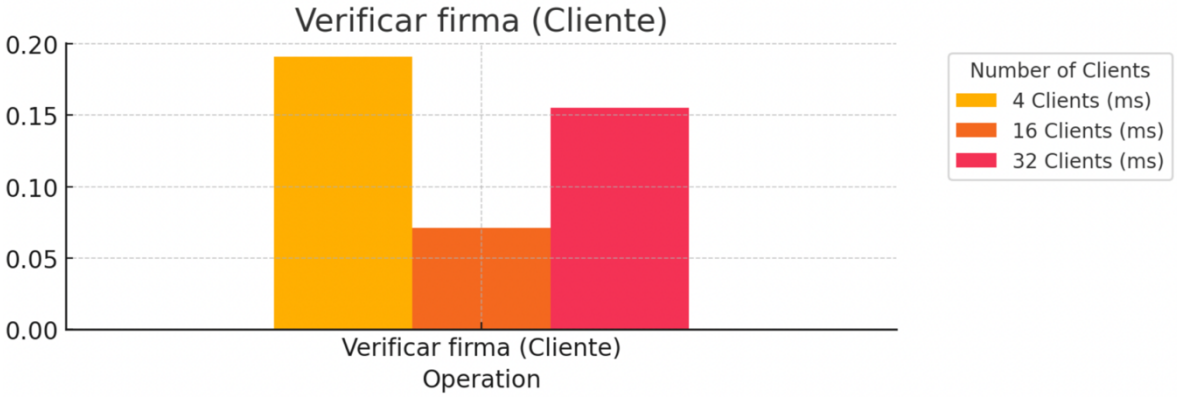
Segundo, como los datos académicos serán almacenados por mucho tiempo. La Forward Secrecy reduce el riesgo a largo plazo, pues se garantiza que los registros almacenados permanezcan seguros incluso si las claves actuales se ven comprometidas en el futuro.

4.

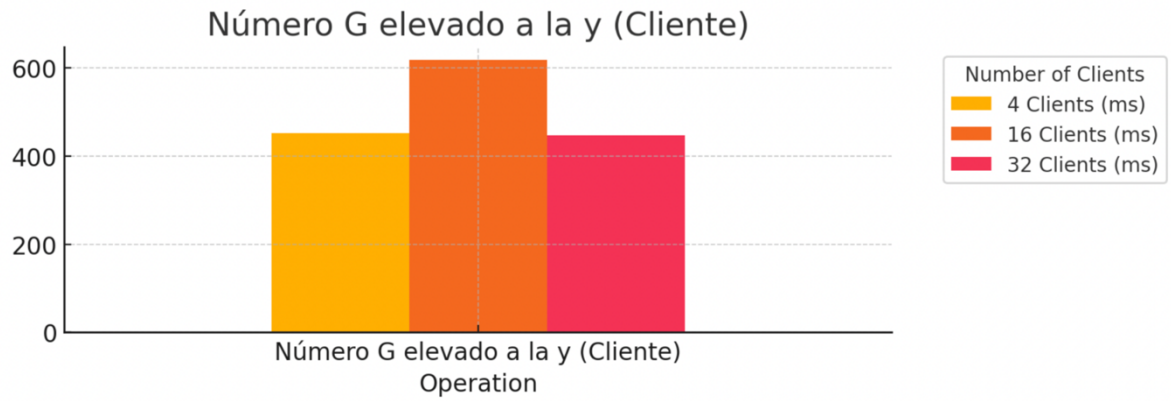
Josué Vega Valbuena (202213040)
Juan Leonardo Rangel Barrera (202215757)

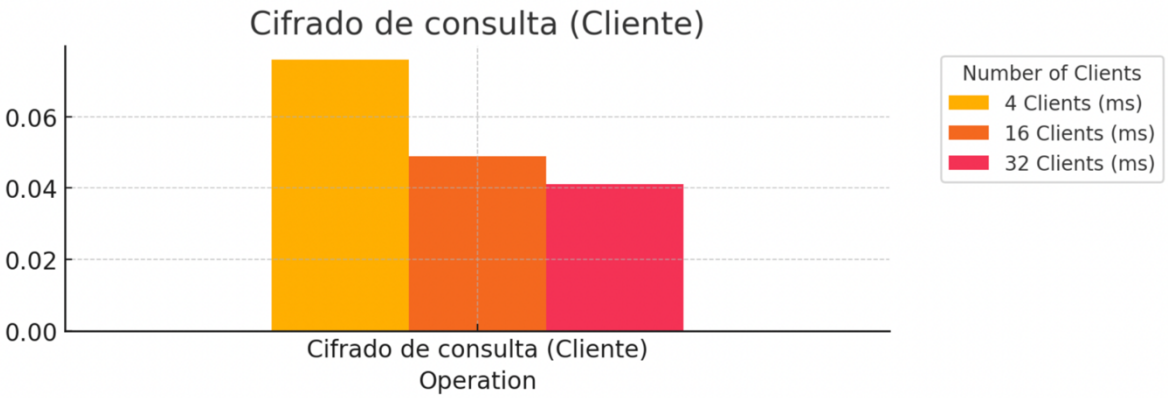
	4 Clientes (ms)	16 Clientes (ms)	32 Clientes (ms)
Generar firma (Servidor)	2715	0,703	1887
Verificar firma (Cliente)	0,191	0,071	0,155
Número G elevado a la y (Cliente)	452874	617961	447055
Cifrado de consulta (Cliente)	0,076	0,049	0,41
Generar número de autenticación (Cliente)	0,059	0,036	0,021
Descifrar consulta (Servidor)	0,068	0,055	0,047
Verificar (Servidor)	0,057	0,042	0,025

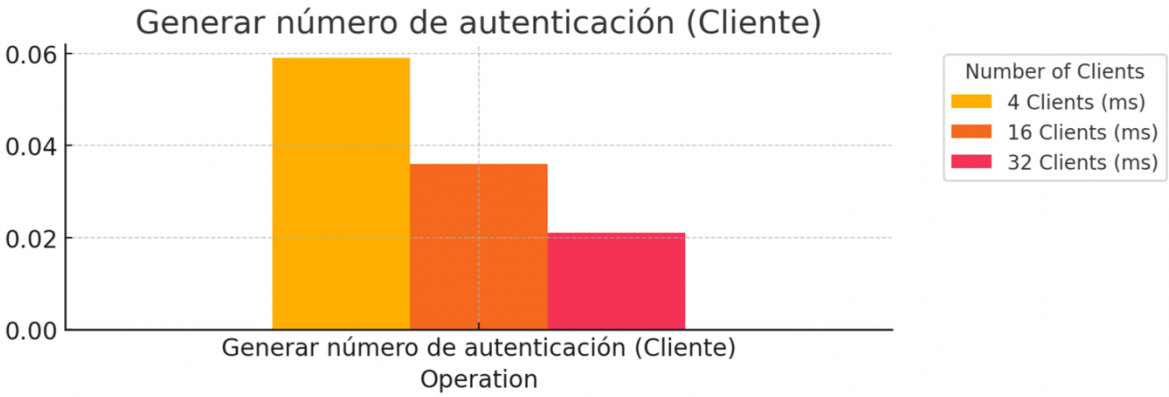


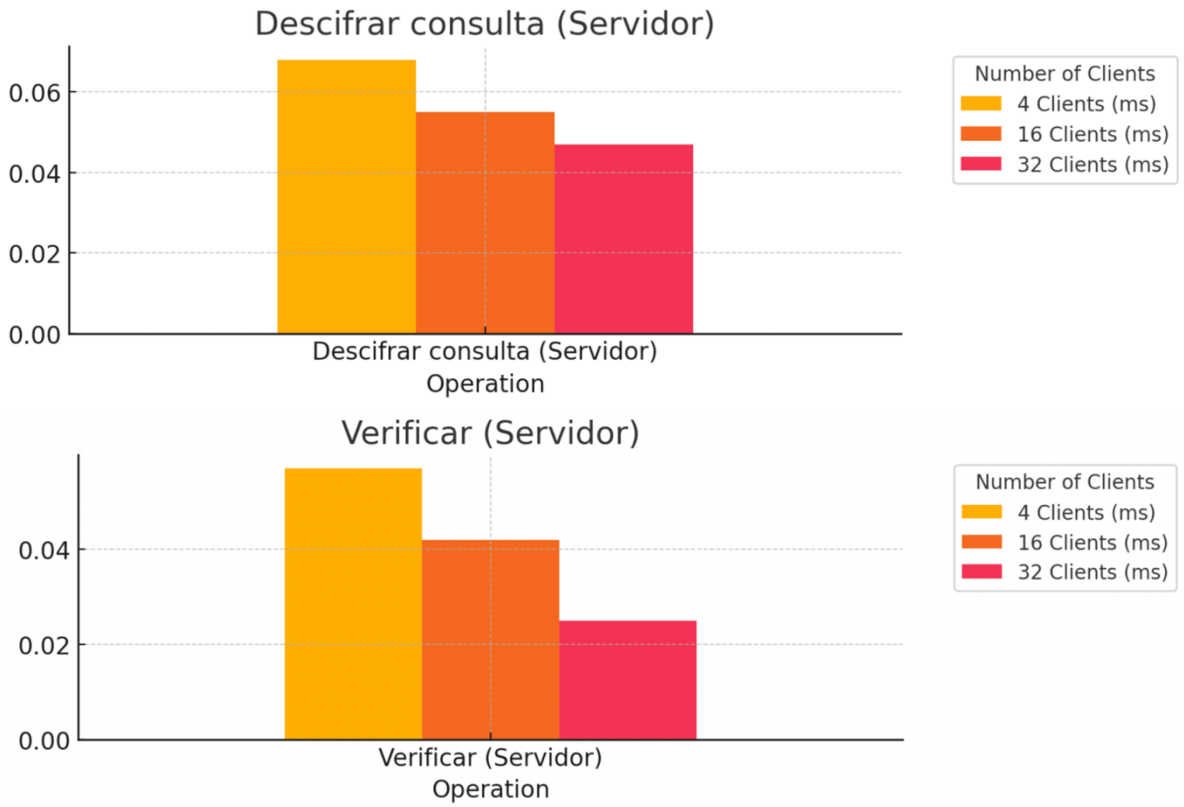


Josué Vega Valbuena (202213040)
Juan Leonardo Rangel Barrera (202215757)









5. Los resultados de los tiempos de operación para los distintos escenarios con 4 clientes sugieren que el sistema maneja variaciones significativas en el rendimiento según el tipo de operación realizada. Es por eso que operaciones intensivas como la generación de firmas y el cálculo del número G elevado a la y resultan con los tiempos más largos, lo cual nos dice que ahí hay un consumo considerable de recursos. Por ejemplo, la generación de firmas oscila alrededor de 3.027 ms, y el cálculo del número G alcanza los 377.888 ms, reflejando la complejidad y la demanda computacional de estas operaciones. Esto a diferencia de tareas como la verificación y el cifrado, las cuales muestran tiempos considerablemente menores, por debajo de 0.2 ms y cerca de 0.07 ms respectivamente, sugiriendo que estas áreas son menos exigentes o están mejor optimizadas. Sin embargo con todo eso hay una tendencia que no se esperaba, pues en ciertos momentos parece que esta implementación funciona mejor bajo más carga concurrente. De todas maneras consideramos que algunos datos no nos dicen mucho, como por ejemplo la generación de la firma, que tiene un comportamiento inesperado, pues, cuando se trabaja con 4 clientes tiene peor rendimiento que con 16, pero con 32 vuelve a haber una diferencia increíble. (Esas operaciones se realizaron múltiples veces y mantenía tal comportamiento).

6. El procesador en el que se realizaron las pruebas es un ARM Apple M2 Pro de 10 núcleos (6 de rendimiento y 4 de eficiencia) a una velocidad de 3.5 GHz para los

núcleos de rendimiento y 2.2 GHz para los núcleos de eficiencia con 16 GB de ram LPDDR5.

Estimaciones:

	Promedio (ms)	Cantidad de operaciones
Cifrar consulta	0,178	5617,977528
Códigos de autenticación	0,038	26315,78947
Verificaciones de firma	0,041	24390,2439

Se tomo el promedio de cada operación en milisegundos. Para obtener la cantidad de operaciones en el segundo, a 1 segundo lo convertimos a milisegundos, quedando una formula 1000/promedio operación, con lo cual podemos estimar la cantidad de operaciones que puede hacer el procesador cada segundo.
Obtenemos que:
El M2 pro es capaz de realizar
Al menos, 5.618 cifrados de consultas, 26.315 generaciones de códigos de autenticación, y 24390 verificaciones de firma.

REFERENCIAS:

Castro, H., & Rueda, S. (2022). Notas de clase del curso Infraestructura de Computacional – ISIS 2203. Departamento de Ingeniería de Sistemas y Computación, Universidad de los Andes.

Certificado digital. (4 de mayor de 2024). En Wikipedia. https://es.wikipedia.org/wiki/Certificado_digital

Gitlan, D. (2024). ¿Qué es el PFS en ciberseguridad? Explicación del secreto perfecto. SSL Dragon. https://www.ssldragon.com/es/blog/perfect-forward-secrecy/