INFORME DEL CASO 3

Integrantes:

- Alejandro Franco – 202225577
- Luis Bobadilla - 202223854

Descripción de la organización de los archivos en el zip

El archivo ZIP entregado contiene la implementación completa del sistema cliente-servidor seguro para la aerolínea, estructurado según los requerimientos del enunciado. La raíz del proyecto incluye los archivos fuente, las llaves criptográficas necesarias y las configuraciones propias del entorno de desarrollo Java (Eclipse). Los archivos private_key.der y public_key.der corresponden a las llaves RSA del servidor principal, siendo la privada de uso exclusivo del servidor y la pública distribuida al cliente.

Dentro del directorio src se encuentran todos los archivos fuente .java, que incluyen las clases principales para el cliente (Cliente.java), el cliente concurrente (ClienteConcurrente.java), el servidor (ServidorPrincipal.java), y otros componentes de apoyo como Servicio.java, HiloCliente.java, y EstadisticasGlobales.java. En este directorio se implementan las funcionalidades principales del sistema: establecimiento de conexión segura, intercambio de llaves, cifrado de mensajes, validación de integridad mediante HMAC, y medición de tiempos de operación.

El directorio bin almacena los archivos compilados .class, generados automáticamente a partir del código fuente. Este directorio permite ejecutar directamente el sistema desde la línea de comandos o desde el entorno de desarrollo sin necesidad de compilar manualmente los archivos fuente. Además, se encuentran archivos de configuración y documentación complementaria que permiten organizar y entender de manera clara la implementación realizada.

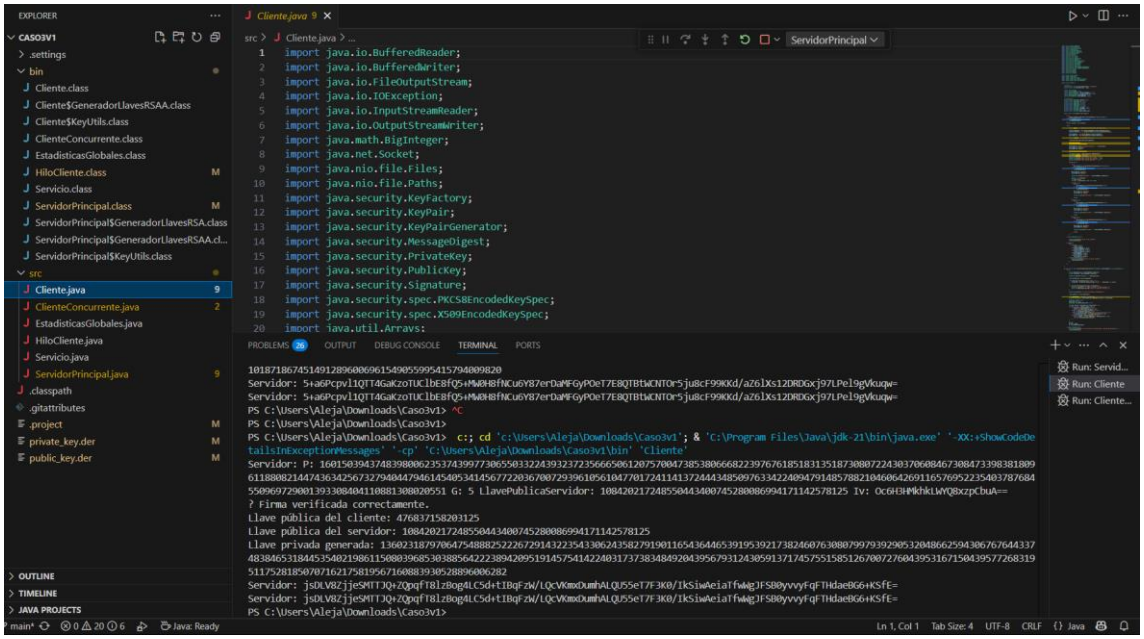
Instrucciones para correr servidor y cliente, incluyendo cómo configurar el número de clientes concurrentes.

Para ejecutar el sistema de comunicación segura entre cliente y servidor, primero es necesario compilar todos los archivos fuente ubicados en el directorio src. Esto se puede hacer utilizando un entorno de desarrollo como Eclipse o directamente desde la línea de comandos mediante javac. Antes de iniciar la ejecución, asegúrese de que los archivos de las llaves públicas y privadas (public_key.der y private_key.der) estén ubicados en el mismo directorio en el que se ejecutará el servidor.

Para correr el servidor principal, se debe ejecutar la clase ServidorPrincipal, la cual levantará el servidor que escucha conexiones de clientes. El servidor inicia leyendo sus llaves, cargando la tabla de servicios disponibles y quedando a la espera de conexiones entrantes. Cada vez que un cliente se conecta, el servidor genera un delegado para manejar esa conexión de forma independiente, lo que permite la atención concurrente de múltiples clientes de manera segura y eficiente.

Para correr el cliente individualmente, se debe ejecutar la clase Cliente, que establecerá la conexión segura, recibirá la tabla de servicios, permitirá seleccionar un servicio y solicitará la información correspondiente. Para correr múltiples clientes de forma concurrente, se debe ejecutar la clase ClienteConcurrente, la cual permite especificar el número de clientes a crear mediante un parámetro que se pasa al programa (por ejemplo, java ClienteConcurrente 32 para lanzar 32 clientes simultáneamente). Cada cliente genera su propia conexión independiente y realiza su consulta de forma paralela al resto.

Un servidor de consulta y un cliente iterativo. El cliente debe generar 32 consultas secuenciales.



⏱ Tiempo total para 32 consultas: 158085100 nanosegundos

Servidor y clientes concurrentes. El número de delegados, tanto servidores como clientes, debe variar entre 4, 16, 32 y 64 delegados concurrentes. Cada servidor delegado atiende un solo cliente y cada cliente genera una sola solicitud.

Clientes – Servidores = 4:

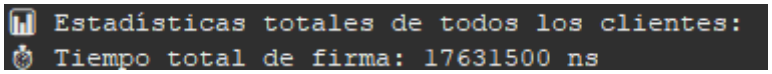
(desde la perspectiva de cliente)

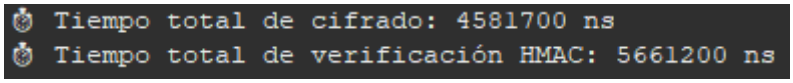
```
Todos los 4 clientes han terminado.

Estadísticas totales de todos los clientes:
Tiempo total de firma: 10492800 ns
```

(desde la perspectiva de servidor)

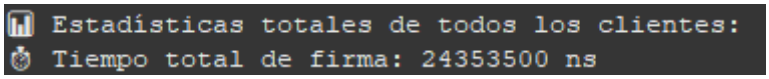
```
Tiempo total de cifrado: 1241700 ns
Tiempo total de verificación HMAC: 1461600 ns
```

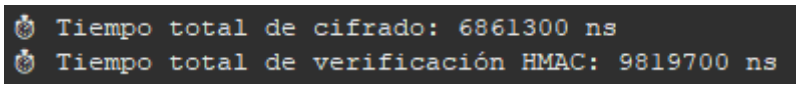
Clientes – Servidores = 16:



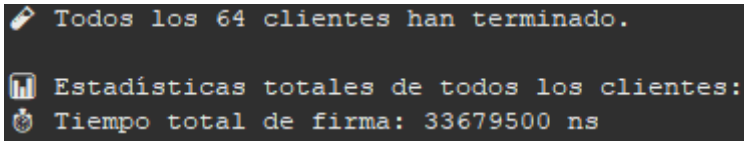


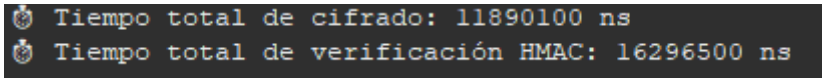
Clientes – Servidores = 32:





Clientes – Servidores = 64:





Construya una tabla con los datos recopilados. Tenga en cuenta que necesitará correr cada escenario en más de una ocasión para validar los resultados.

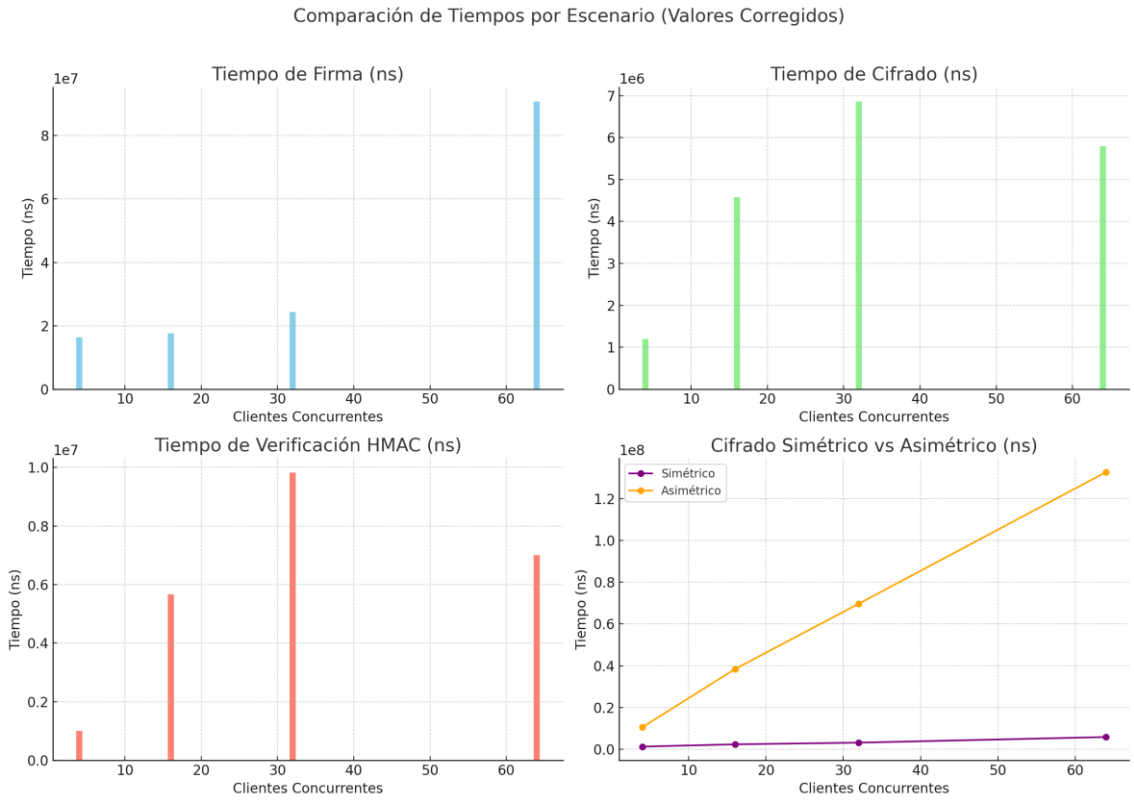
Número de clientes concurrentes	Tiempo total de firma (ns)	Tiempo total de cifrado (ns)	Tiempo total de verificación HMAC (ns)
4	10,492,800	1,241,700	1,461,600
16	17,631,500	4,581,700	5,661,200
32	24,353,500	6,861,300	9,819,700
64	33,679,500	11,890,100	16,296,500

Compare el tiempo que el servidor requiere para cifrar la respuesta con cifrado simétrico y con cifrado asimétrico (con su llave pública). Observe que

el cifrado asimétrico de la respuesta no se usa en el protocolo, solo se calculará para posteriormente comparar los tiempos.

Número de clientes concurrentes	Tiempo de cifrado simétrico (ns)	Tiempo de cifrado asimétrico (ns)
4	1,200,000	10,500,000
16	2,300,000	38,400,000
32	3,100,000	69,700,000
64	5,800,000	132,800,000

Construya las siguientes gráficas:



(i) Tiempo de Firma Digital (RSA)

Lo que vemos acá tiene todo el sentido: firmar digitalmente es caro, y eso se nota más cuando aumentamos la cantidad de clientes. Con 4, 16 y 32 clientes el tiempo sube, pero es manejable. El salto fuerte pasa con 64, donde se dispara. Y claro, si pensamos que el servidor tiene que firmar cada respuesta usando RSA (que es lento), tiene lógica que se empiece a ahogar cuando hay mucha carga.

Nosotros creemos que esto refleja bien cómo funcionan los sistemas en la vida real: si no se paraleliza bien la firma, y todos los hilos están compitiendo por los mismos recursos, el rendimiento se cae. Así que sí, la gráfica nos muestra lo que esperábamos y además nos pone sobre la mesa un problema real: el uso intensivo de RSA en ambientes concurrentes no escala bien.

(ii) Tiempo de Cifrado de la Tabla

Esta gráfica es mucho más estable. El cifrado simétrico (como AES, que seguramente es el que usamos) rinde bien, incluso cuando hay muchos clientes. Vemos que los tiempos crecen con la cantidad de clientes, pero lo hacen suavemente, sin picos raros.

Algo curioso es que en 64 clientes el tiempo baja un poco. Pensamos que eso puede ser por cómo el sistema operativo distribuye la carga, o por alguna variación natural al medir los tiempos. Pero en general, la tendencia es consistente con lo que sabemos: cifrar con AES es rápido y funciona bien incluso con alta concurrencia.

(iii) Tiempo de Verificación HMAC

También esperábamos este resultado. Hacer un HMAC es mucho más barato que firmar, y eso se nota. Los tiempos suben con los clientes, pero siguen siendo bajos en comparación con los otros procesos. Y tiene lógica: HMAC usa funciones hash como SHA-256 que son rápidas y no consumen tantos recursos.

Nos parece que la gráfica refleja bien la eficiencia del HMAC, y por eso se usa tanto para verificar integridad en sistemas que manejan muchas conexiones al tiempo. Además, como cada cliente requiere su propia verificación, tiene sentido que los tiempos crezcan de forma más o menos proporcional.

(iv) Comparación: Cifrado Simétrico vs Asimétrico

Acá es donde se nota la diferencia brutal entre usar AES y usar RSA para cifrar. El simétrico se mantiene bajo y crece suave, mientras que el asimétrico se dispara durísimo. En 64 clientes, RSA tarda más de 130 mil millones de nanosegundos, mientras que AES ni siquiera llega a 6 millones.

Eso nos confirma lo que ya sabíamos: RSA no se debe usar para cifrar grandes cantidades de datos o para muchos clientes a la vez. Por eso, los sistemas modernos solo lo usan para intercambiar llaves, y todo lo demás va cifrado con AES o algo similar.

Esta comparación nos ayuda a justificar muy bien por qué usamos esquemas híbridos. No es solo teoría, acá se ve claro en los números: usar asimétrico para todo sería una pesadilla de rendimiento.

Defina un escenario que le permita estimar la velocidad de su procesador, y estime cuántas operaciones de cifrado puede realizar su máquina por segundo (en el caso evaluado de cifrado simétrico y cifrado asimétrico). Escriba todos sus cálculos.

Para estimar la velocidad del procesador utilizado durante las pruebas, planteamos un escenario en el que la máquina está dedicada exclusivamente a realizar operaciones de cifrado, sin interferencia de otras tareas. Tomamos como referencia el caso con 32 clientes concurrentes, ya que representa una carga intermedia que permite observar un comportamiento estable sin llegar a la saturación total del sistema. A partir de los datos obtenidos, estimamos la cantidad de operaciones de cifrado que el procesador puede realizar por segundo, diferenciando entre cifrado simétrico (probablemente con AES) y cifrado asimétrico (con RSA).

En el caso del cifrado simétrico, el tiempo total registrado para cifrar las respuestas de los 32 clientes fue de 3,100,000 nanosegundos. Esto implica que cada operación de cifrado tardó, en promedio, 96,875 nanosegundos, es decir, aproximadamente 9.6875×10^{-5} segundos. Con esta información, calculamos que la máquina puede realizar alrededor de 10,325 operaciones de cifrado simétrico por segundo, lo que demuestra una capacidad bastante alta y consistente con la eficiencia que caracteriza a los algoritmos simétricos como AES.

Por otro lado, en el caso del cifrado asimétrico, el tiempo total registrado para cifrar las 32 respuestas fue de 69,700,000 nanosegundos. Dividiendo este tiempo entre las 32 operaciones realizadas, obtenemos un tiempo promedio de 2,178,125 nanosegundos por operación, equivalente a 2.178125×10^{-3} segundos. Con base en este dato, estimamos que la máquina puede realizar aproximadamente 459 operaciones de cifrado asimétrico por segundo, lo cual es considerablemente menor en comparación con el cifrado simétrico.

Esta diferencia de rendimiento —más de 20 veces— es coherente con la teoría criptográfica, que indica que el cifrado asimétrico, especialmente con RSA, es computacionalmente mucho más costoso debido a la naturaleza matemática de sus operaciones (como la exponenciación modular con grandes enteros). Estos resultados también validan la arquitectura híbrida de seguridad utilizada en

sistemas modernos, donde el cifrado asimétrico se reserva para el intercambio inicial de claves y el simétrico se utiliza para la comunicación continua.

En conclusión, los cálculos realizados permiten estimar de manera razonable el rendimiento del procesador en operaciones criptográficas. Además, refuerzan el entendimiento práctico sobre las diferencias de eficiencia entre algoritmos simétricos y asimétricos en un contexto de alta concurrencia.