INFRAESTRUCTURA COMPUTACIONAL – ISIS2203

INFORME CASO 3: CANALES SEGUROS

Angie Ximena López Cruz

Ax.lopez@uniandes.edu.co

202312848

Juan David Torres Albarracín

jd.torresa1@uniandes.edu.co

202317608

Bogotá – Colombia

28 de abril de 2025

0 Descripción de la organización de los archivos

Los archivos de la solución siguen este ordenamiento:

```
InfraCompCaso3/
├── docs/                          # Carpeta para documentación
│   └──InformeCaso3Infracomp.pdf          # Informe que documenta la solución y los experimentos
├── src/                           # Carpeta principal del código fuente
│   ├── ServidorPrincipal.java      # Clase principal del servidor
│   ├── ServidorDelegado.java       # Clase para manejar conexiones de clientes
│   ├── Cliente.java                # Clase principal del cliente
│   ├── ClienteDelegado.java        # Clase para manejar clientes concurrentes
│   └── ClientePruebaRendimiento.java # Clase para pruebas de rendimiento
└── README.md                      # Archivo de descripción del proyecto
```

Todos los archivos Java realizados en la implementación del esquema Cliente-Servidor se pueden encontrar dentro de la carpeta 'root' InfraCompCaso3. Esto facilitará la ejecución de la solución. A continuación, se presenta cada una de las clases implementadas y una breve descripción de su propósito:

GeneradorLlaves.java
Este archivo contiene una clase que genera un par de llaves RSA (pública y privada) y las guarda en archivos (server_public.key y server_private.key). Es una utilidad independiente que se ejecuta antes de iniciar el servidor para garantizar que las llaves necesarias estén disponibles.

ServidorPrincipal.java
Este es el archivo principal del servidor. Contiene la lógica para inicializar la tabla de servicios, cargar o generar las llaves RSA, y manejar las conexiones entrantes de los clientes. Utiliza un pool de hilos para delegar las conexiones a instancias de ServidorDelegado.

ServidorDelegado.java
Este archivo implementa la lógica de comunicación segura entre el servidor y los clientes. Maneja el protocolo de autenticación, el intercambio de claves Diffie-Hellman, y la transmisión de datos cifrados y autenticados (usando AES y HMAC). Cada conexión de cliente es manejada por una instancia de esta clase.

Cliente.java
Este archivo implementa la lógica del cliente principal. Permite al usuario interactuar con el servidor de forma manual (modo interactivo) o realizar múltiples consultas automáticas

(modo automático). Maneja la autenticación, el intercambio de claves, y la comunicación segura con el servidor.

ClienteDelegado.java
Este archivo implementa un cliente delegado que se utiliza en pruebas concurrentes. Cada instancia de esta clase representa un cliente que realiza una consulta al servidor de forma independiente. Es utilizado por ClientePruebaRendimiento para simular múltiples clientes concurrentes.

ClientePruebaRendimiento.java
Este archivo se utiliza para realizar pruebas de rendimiento. Permite ejecutar clientes en dos modos: iterativo (un cliente realiza múltiples consultas secuenciales) o concurrente (varios clientes realizan consultas simultáneamente). Es útil para evaluar el desempeño del sistema bajo diferentes cargas.

1 Instrucciones para correr el servidor y el cliente.

Prerrequisitos.
En un primer lugar, java debe estar presente en el path del equipo dentro del cual se quiera ejecutar la solución. Posteriormente, se deberá abrir una consola en la carpeta 'root' del proyecto. Para distinguirla de otras consolas que deberán inicializarse más adelante, esta se llamará 'Consola A'.

1. Compilar todos los archivos:
Ejecutar en consola A el comando:

Consola A>> javac *.java

2. Generar las llaves RSA (solo se necesita una vez):

Consola A>> java GeneradorLlaves

3. Iniciar el servidor principal:

Consola A>> java ServidorPrincipal

4. Ejecutar cliente en modo interactivo:

Para esto, será necesario abrir una nueva consola (Consola B) desde la cual se ejecutará el Cliente. Además, será necesario que en la Consola A ya se tenga corriendo el Servidor Principal.

Consola B>>java Cliente

El modo interactivo despliega varias opciones que se pueden escoger para simular los servicios que ofrece la aerolínea.

5. Ejecutar cliente en modo iterativo (n consultas secuenciales):

Para esto, será necesario abrir una nueva consola (Consola B) desde la cual se ejecutará el Cliente. Además, será necesario que en la Consola A ya se tenga corriendo el Servidor Principal.

Consola B>>java ClientePruebaRendimiento iterativo <n>

Donde <n> deberá ser reemplazado por el número de consultas secuenciales que se quiere realizar con dicho Cliente. Por ejemplo, para realizar 32 consultas secuenciales:

Consola B>>java ClientePruebaRendimiento iterativo 32

Ejecutar pruebas de rendimiento concurrentes:

Para esto, será necesario abrir una nueva consola (Consola B) desde la cual se ejecutará el Cliente. Además, será necesario que en la Consola A ya se tenga corriendo el Servidor Principal.

Consola B>>java ClientePruebaRendimiento concurrente <n>

Donde <n> deberá ser reemplazado por el número de clientes concurrentes con los que se realizarán consultas. N deberá ser menor o igual a 64 (en el enunciado se dice que se harán pruebas con máximo 64 clientes concurrentes).

Ejemplo con cuatro clientes concurrentes:

Consola B>>java ClientePruebaRendimiento concurrente 4

2 Análisis

Se desarrolló un sistema cliente-servidor seguro que establece comunicación cifrada mediante RSA, Diffie-Hellman, AES y HMAC. El objetivo es medir el desempeño del servidor en diferentes escenarios de carga concurrente.

Revisar en el archivo de excel donde está por cada escenario los resultados de cada una de las 3 pruebas

Escenario 1: Un servidor de consulta y un cliente iterativo con 32 consultas secuenciales. En este escenario se realizaron 3 pruebas para validar sus resultados y tener un promedio total de los tiempos.

Promedios totales:

Promedio Tiempo Firmar(ms)	Promedio Tiempo Cifrado AES (ms)	Promedio Tiempo cifrado RSA (ms)	Promedio Tiempo verificación (ms)
0,31860833	0,16855	0,076125	0,11181146

Escenario 2: Servidor y clientes concurrentes, donde el número de delegados, tanto de servidores como de clientes es de 4. Cada servidor delegado atiende un solo cliente y cada cliente genera una sola solicitud.

Para este escenario se realizaron 3 pruebas para validar los resultados y tener un promedio general de los tiempos.

Promedios totales:

Promedio Tiempo Firmar (ms)	**PromedioTiempo Cifrado AES (ms)**	**Promedio Tiempo Cifrado RSA (ms)**	**Promedio Tiempo Verificación (ms)**
0,35141667	0,096825	0,05269167	0,11440833

Escenario 3: Servidor y clientes concurrentes, donde el número de delegados, tanto de servidores como de clientes es de 16. Cada servidor delegado atiende un solo cliente y cada cliente genera una sola solicitud.

Para este escenario se realizaron 3 pruebas para validar los resultados y tener un promedio general de los tiempos.

Promedios totales:

Firmado DH (ms)	**Cifrado AES (ms)**	**Cifrado RSA (ms)**	**Verificación HMAC (ms)**
0,46944792	0,0938875	0,06871042	0,07559375

Escenario 4: Servidor y clientes concurrentes, donde el número de delegados, tanto de servidores como de clientes es de 32. Cada servidor delegado atiende un solo cliente y cada cliente genera una sola solicitud.

Para este escenario se realizaron 3 pruebas para validar los resultados y tener un promedio general de los tiempos.

Promedios totales:

Promedio Tiempo Firmar(ms)	Promedio Tiempo Cifrado AES (ms)	Promedio Tiempo cifrado RSA (ms)	Promedio Tiempo verificación (ms)
0,4287	0,06784063	0,07545313	0,07056146

Escenario 5: Servidor y clientes concurrentes, donde el número de delegados, tanto de servidores como de clientes es de 64. Cada servidor delegado atiende un solo cliente y cada cliente genera una sola solicitud.

Para este escenario se realizaron 2 pruebas para validar los resultados y tener un promedio general de los tiempos.

Promedios totales:

Promedio Tiempo Firmar(ms)	Promedio Tiempo Cifrado AES (ms)	Promedio Tiempo cifrado RSA (ms)	Promedio Tiempo verificación (ms)
0,38941016	0,04644375	0,06368438	0,02885156

Después de validar cada escenario obtuvimos estos promedios de todos los escenarios:

Escenario	**Tiempo de firmado DH (ms)**	**Tiempo cifrado simétrico AES (ms)**	**Tiempo cifrado asimétrico RSA (ms)**	**Tiempo verificación HMAC (ms)**
32 consultas secuenciales	0,31860833	0,16855	0,076125	0,11181146
4 concurrentes	0,35141667	0,096825	0,05269167	0,11440833

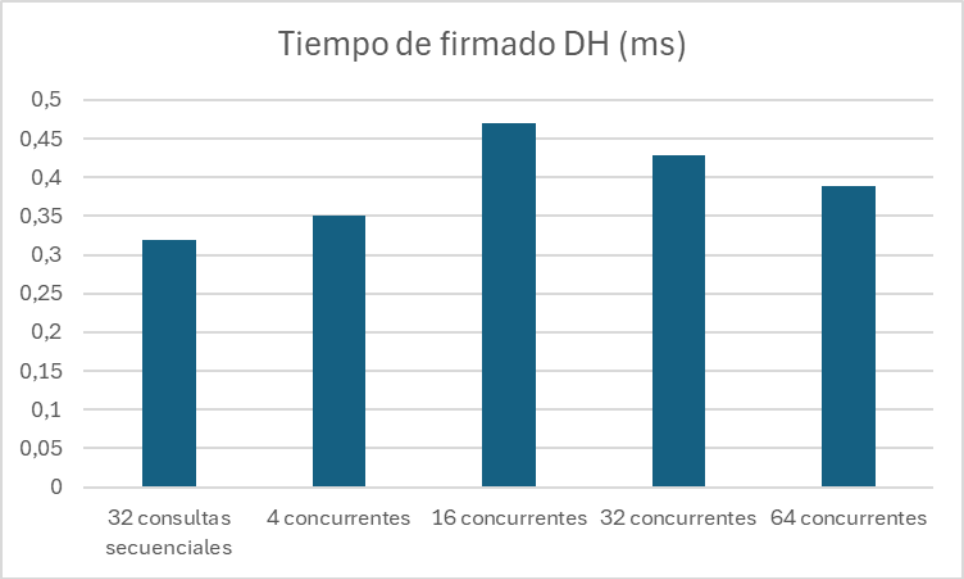
16 concurrentes	0,46944792	0,0938875	0,06871042	0,07559375
32 concurrentes	0,4287	0,06784063	0,07545313	0,07056146
64 concurrentes	0,38941016	0,04644375	0,06368438	0,02885156

En el experimento realizado, se observó que, para mensajes pequeños (alrededor de 62 bytes), el tiempo de cifrado simétrico con AES fue ligeramente superior al tiempo de cifrado asimétrico con RSA. Esta diferencia se debe a que el cifrado AES, particularmente en modo CBC con PKCS5Padding, implica una serie de operaciones adicionales como la inicialización del Cipher, la generación y uso del vector de inicialización (IV), y la gestión de bloques y relleno, que introducen un costo fijo en el proceso de cifrado, independientemente del tamaño del mensaje. Por el contrario, RSA en este contexto únicamente realiza una operación matemática de exponenciación modular (m^e mod n), optimizada para mensajes pequeños, sin necesidad de operaciones de bloques ni IV.

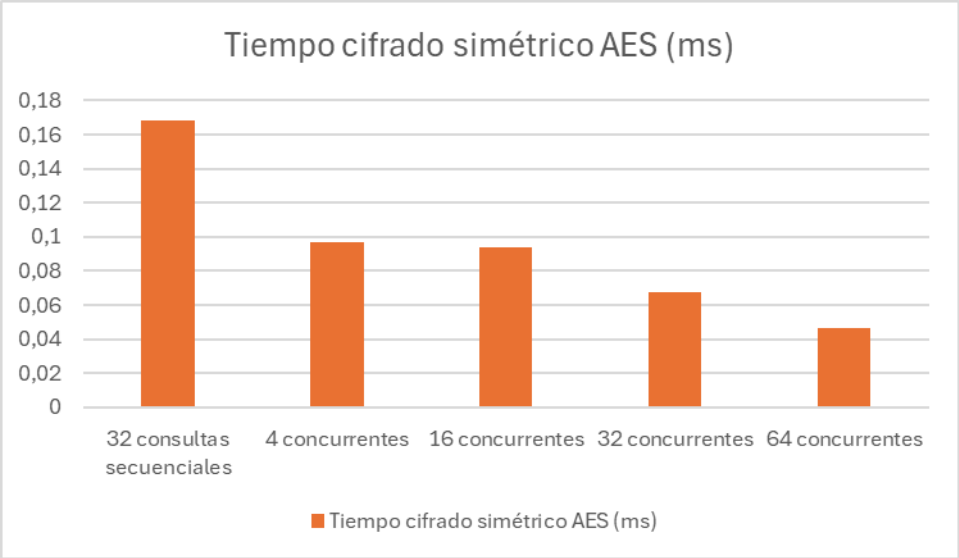
En consecuencia, aunque AES es en general mucho más rápido que RSA para el cifrado de grandes volúmenes de datos (por su eficiencia en procesamiento de bloques), en este caso particular de mensajes pequeños, el overhead operativo de AES resulta más notorio que el costo matemático de RSA, haciendo que el tiempo de cifrado simétrico sea ligeramente mayor.

Esta observación coincide con las prácticas usuales, donde RSA se utiliza típicamente para cifrar mensajes pequeños como llaves de sesión o firmas digitales, y AES se emplea para el cifrado de datos de gran tamaño, donde su ventaja de rendimiento es significativa.

Al analizar los resultados obtenidos para los tiempos de firmado, cifrado y verificación en los diferentes escenarios, se destacan los siguientes comportamientos:

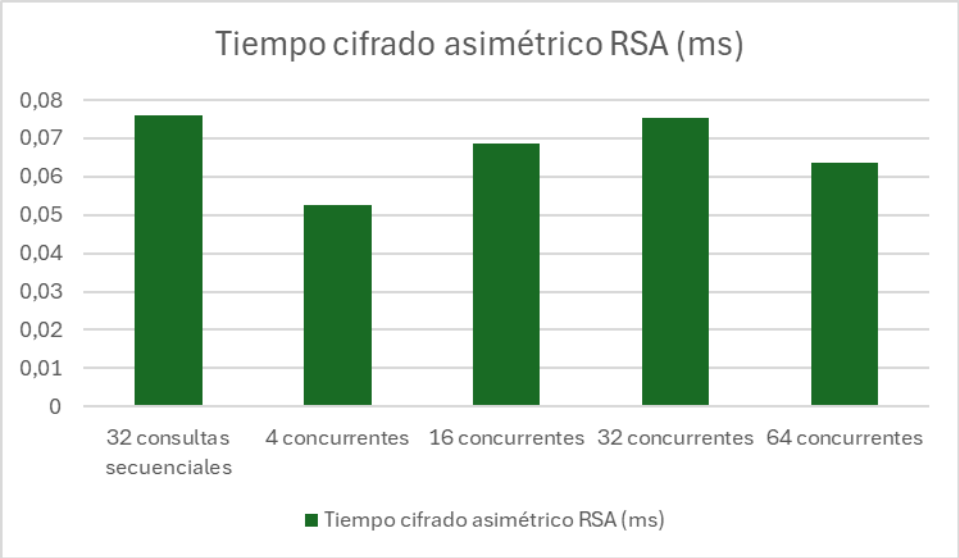


El análisis del tiempo requerido para firmar evidencia que este proceso se mantiene estable entre los distintos escenarios, con valores cercanos a los 0.3 – 0.45 milisegundos. Sin embargo, al incrementar la concurrencia a 16 y 32 clientes, se observa un leve aumento, lo cual puede atribuirse a la carga compartida entre los hilos de ejecución y la competencia por recursos de CPU. Curiosamente, al llegar a 64 clientes concurrentes, el tiempo disminuye nuevamente, lo que sugiere que el sistema operativo y el procesador podrían estar gestionando los hilos de forma más eficiente, aprovechando mejor la paralelización. Esto demuestra que la firma digital, aunque implica operaciones criptográficas, es un proceso lo suficientemente optimizado para mantener tiempos de respuesta bajos incluso bajo carga.

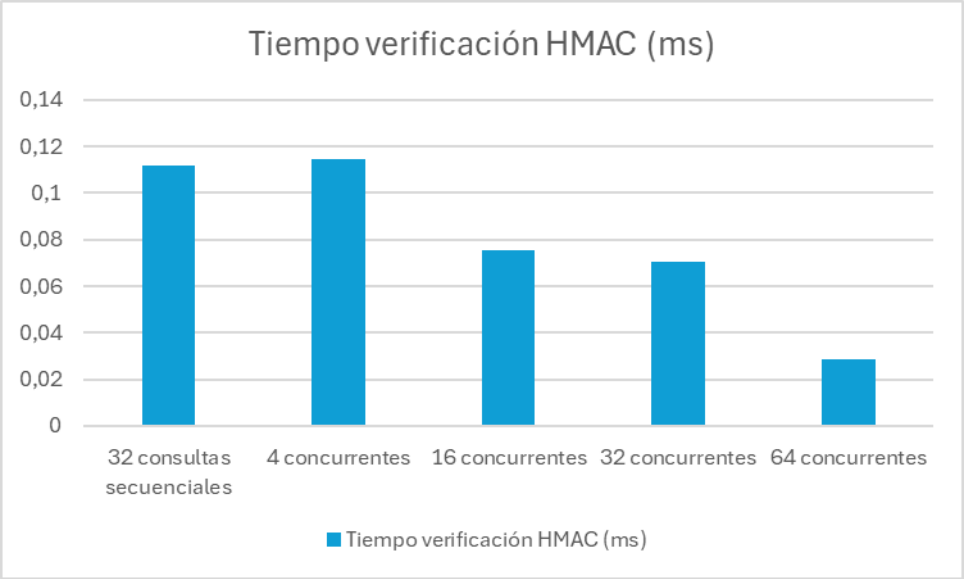


En el caso del cifrado simétrico con AES, los resultados obtenidos muestran una tendencia decreciente en el tiempo de cifrado a medida que aumenta la cantidad de clientes concurrentes. En el escenario secuencial, el tiempo promedio es más alto, cercano a los 0.16

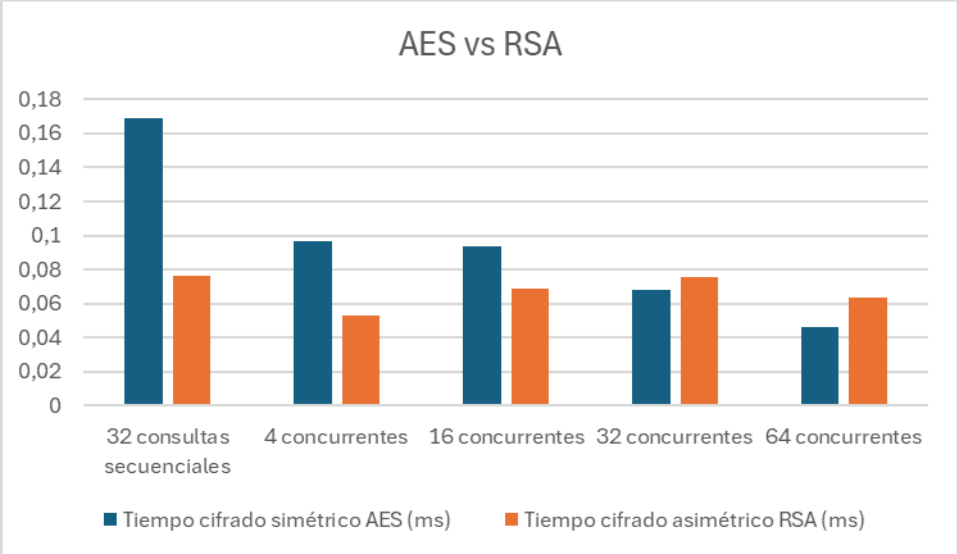
ms, mientras que al llegar a 64 clientes concurrentes este tiempo disminuye notablemente, alcanzando valores alrededor de 0.06 ms. Esta reducción podría deberse a las optimizaciones que el hardware aplica a operaciones con AES, como el uso de instrucciones especiales, que permiten ejecutar el cifrado de manera eficiente en paralelo. En resumen, el cifrado simétrico no solo es rápido por naturaleza, sino que además se ve favorecido por la concurrencia y los recursos del sistema, manteniendo su eficiencia en condiciones de alta demanda.



El comportamiento del cifrado asimétrico con RSA muestra un patrón distinto al del cifrado simétrico. Aunque los tiempos se mantienen en un rango aceptable (entre 0.06 y 0.08 ms), no se observa una mejora clara con el aumento en la cantidad de clientes concurrentes. Esto se debe a que RSA es un algoritmo mucho más intensivo en cómputo, especialmente en comparación con AES. Las operaciones con exponentes grandes y módulos de 1024 bits no se benefician tanto de la paralelización como los algoritmos simétricos. Además, RSA no suele estar acelerado por hardware de forma tan extendida como AES. En consecuencia, el cifrado con RSA es más costoso en términos de tiempo y recursos, lo que lo hace menos adecuado para escenarios donde se requiere alta eficiencia y grandes datos, aunque sigue siendo útil para establecer llaves de sesión o firmar digitalmente.



Respecto a la verificación de la integridad mediante HMAC, los resultados muestran una clara mejora en el rendimiento a medida que se incrementa el número de clientes concurrentes. El tiempo promedio de verificación disminuye progresivamente desde más de 0.13 ms en el caso secuencial hasta menos de 0.04 ms con 64 clientes concurrentes. Esta tendencia puede explicarse por el hecho de que las operaciones de HMAC son computacionalmente ligeras y fácilmente paralelizables. En escenarios concurrentes, los múltiples hilos que ejecutan verificaciones pueden hacerlo de forma independiente sin interferencia, lo cual saca provecho del procesador multinúcleo. Estos resultados respaldan el uso de HMAC como un mecanismo altamente eficiente para garantizar la integridad de los datos en sistemas con alta carga de usuarios.



En esta tabla podemos comparar los tiempos que tuvieron las dos formar de cifrar, los cuales son totalmente opuestos, donde se evidencia que a medida que la carga aumenta el

cifrado simétrico baja su tiempo y el asimétrico comienza a aumentarlo. Esto, en resumen, podría ser porque el asimétrico es computacionalmente muy costoso y a medida que va teniendo mayor carga se va demorando más. En cambio, AES a medida que aumenta la carga y que la puede ejecutar en diferentes hilos podría estarse ejecutando en paralelo logrando que los tiempos puedan disminuir.

Finalmente, para estimar la velocidad de cifrado de la máquina utilizada, se diseñó un escenario controlado donde el servidor ejecuta únicamente operaciones de cifrado de una tabla de datos, sin incluir firma, verificación ni otras tareas. Se midió el tiempo promedio requerido para cifrar dicha tabla utilizando dos algoritmos: AES que es simétrico y RSA que es asimétrico, en condiciones de la misma carga concurrente, en este caso 64 clientes.

Para calcularla vamos a hacer uso de la siguiente fórmula:

Operaciones por segundo: = 1000 ms/ tiempo promedio por operación.

Datos:

	AES	RSA
Tiempo promedio por operación	0,046	0,064
Operaciones por segundo	21739,13043 op/seg	15625 op/seg

Los resultados muestran que el procesador es capaz de ejecutar aproximadamente 21739,1304 operaciones de cifrado simétrico por segundo, y unas 15625 operaciones de cifrado asimétrico por segundo, en condiciones óptimas. Esta diferencia refleja la naturaleza más eficiente del cifrado simétrico, tanto en términos de complejidad algorítmica como de soporte a nivel de hardware. Por este motivo, el cifrado simétrico es más adecuado para escenarios con alta demanda de rendimiento, mientras que el cifrado asimétrico se reserva típicamente para tareas como el intercambio de claves o la firma digital.

3 Referencias

1. GeeksforGeeks. (2023, January 27). *KeyPairGenerator genKeyPair() method in Java with Examples*. GeeksforGeeks. https://www.geeksforgeeks.org/keypairgenerator-genkeypair-method-in-java-with-examples/

Este ejemplo se usó para entender una opción de generación de parejas de llaves (públicas y privadas) usando el módulo java.security y las clases que ofrece (KeyPair, KeyPairGenerator).

2. GeeksforGeeks. (2023b, May 6). *Java Implementation of DiffieHellman Algorithm between Client and Server*. GeeksforGeeks. https://www.geeksforgeeks.org/java-implementation-of-diffie-hellman-algorithm-between-client-and-server/

   Se usó este recurso como base en la implementación de Diffie-Hellman en la clase ServidorDelegado.

3. *Java examples for javax.crypto.spec.IVParameterSpec*. (n.d.). Javatips.net. https://www.javatips.net/api/javax.crypto.spec.ivparameterspec

   Estos ejemplos sirvieron en la implementación del vector de inicialización (IV) usando la clase IVParameterSpec en java.

4. Kumar, P. (2022, August 3). *Java Socket Programming - Socket Server, Client example*. DigitalOcean. https://www.digitalocean.com/community/tutorials/java-socket-programming-server-client

   Se usó como guía en la elaboración del Cliente-Servidor en la solución. Así, se usa la clase Socket de java.net para hacer más fácil la implementación de la conexión.

5. TutorialsPoint. (2025, March 25). *Creating MAC in Java cryptography*. https://www.tutorialspoint.com/java_cryptography/java_cryptography_creating_mac.htm

   Se usó como referencia para la autenticación de los HMAC.