María Juliana Ballesteros Estupiñán - 202313216

Alejandro Bernal Lopez - 202211032

Caso 3

Para resolver el caso se implementaron las siguientes clases:

CifradoAsimetrico

Esta clase realiza operaciones de cifrado y descifrado de datos utilizando el algoritmo RSA. El método *cifrar* se encarga de cifrar un arreglo de bytes utilizando una llave pública, de esta forma solo quien posea la llave privada correspondiente puede descifrarlo. El método *descifrar* recibe un mensaje cifrado y lo descifra usando la llave privada.

CifradoSimetrico

Esta clase se encarga de realizar operaciones de cifrado y descifrado de datos mediante el algoritmo AES en modo CBC con relleno PKCS5Padding. El método *cifrar* recibe un mensaje y una clave, genera un vector de inicialización (IV) aleatorio de 16 bytes, cifra el mensaje usando la llave AES y el IV, y devuelve el IV concatenado al mensaje cifrado. El método *descifrar* recibe un arreglo de bytes que contiene el IV y el mensaje cifrado, separa ambos componentes y descifra el mensaje con la llave AES y el IV que se obtuvo. Por último, el método *generarIV* genera de forma segura un IV aleatorio mediante SecureRandom.

Cliente

Esta clase representa el programa que actúa como cliente dentro del protocolo de comunicación seguro con el servidor principal de la aerolínea. El cliente establece una conexión segura mediante sockets TCP, valida la identidad del servidor usando firma digital y realiza un intercambio de llaves de sesión mediante Diffie-Hellman. Una vez se establece la sesión segura, el cliente recibe la tabla de servicios cifrada con AES , la descifra con la llave de sesión y valida su integridad mediante HMAC. Después, el usuario selecciona un servicio disponible, cifra su solicitud con la clave de sesión y envia la petición al servidor. Por ultimo, recibe la respuesta correspondiente al servicio solicitado y la descifra.

FirmaDigital

Esta clase se encarga de implementar las funciones necesarias para realizar firmas digitales como parte del proceso de autenticación del servidor. Permite cifrar un mensaje mediante una clave privada RSA y verifica esa firma usando la clave publica RSA correspondiente. Se utiliza el algoritmo SHA256withRSA, combinando un hash SHA-256 con cifrado RSA para garantizar la integridad de los datos. La firma digital es usada al inicio de la comunicación, donde el servidor firma un reto aleatorio enviado al cliente. La clase define dos métodos principales, el método *sign* genera la firma digital de un arreglo de bytes usando la clave privada, y *verify* verifica si una firma corresponde a los datos utilizando la clave publica.

GestorLlaves

Esta clase se encarga de gestionar la carga de las llaves criptográficas necesarias para la comunicación segura entre cliente y servidor. Lee los archivos .der o .pem los cuales contienen

las llaves privadas o publicas, identifica el formato del archivo, procesa su contenido y reconstruye los objetos *PrivateKey* o *PublicKey* para que se puedan usar posteriormente en operaciones de firma digital y verificación.

ServidorDelegado

Esta clase se encarga de atender individualmente a cada cliente que se conecta al servidor. Cuando el cliente se conecta, ServidorDelegado maneja todo el proceso de autenticación, establecimiento de sesión segura, cifrado y respuesta. Primero, genera y firma un reto aleatorio que el cliente debe validar, garantizando la autenticación del servidor. Luego realiza un ontercambiio de llaves Diffie-Hellman para establecer las llaves de sesión usadas en es cifrado AES y la validación de integridad (HMAC). Después, el servidor delegado cifra la tabla de servicios disponibles y se la envía al cliente. Finalmente recibe una solicitud cifrada del cliente, la descifra y responde con la información del servicio solicitado.

MainServidor

Esta clase es el programa principal, el cual inicia el servidor del sistema de comunicación segura. Se encarga de crear un *ServerSocket* y se mantiene a la espera de conexiones entrantes de clientes. Cuando un cliente se conecta, el servidor crea un nuevo hilo que atiende esa conexión mediante una instancia de *ServidorDelegado*. Además, durante su inicialización, carga las llaves necesarias (publica y privada) usando *Gestor Llaves*, y prepara la tabla de servicios disponibles. De esta forma se garantiza que el servidor este disponible para atender múltiples clientes concurrentes y que cada uno pueda realizar el proceso del protocolo de manera independiente y segura.

TablasServicios

Esta clase representa la tabla de servicios ofrecidos por el sistema de la aerolínea. Cada servicio se almacena en un mapa, en el cual la llave es el id del servicio. La clase permite consultar un servicio especifico, y obtener una lista de todos los servicios existentes. Además, se encarga de serializar toda la tabla en un arreglo de bytes, para que toda la información pueda ser transmitida de manera segura al cliente a traves del canal de cifrado. La tabla se inicializa con los 3 servicios y cada servicio se representa mediante la clase interna *Servicio*, la cual almacena el ID, nombre, dirección IP y el puerto correspondiente de cada servicio.

MedidorTiempos

Esta clase se encarga de medir los tiempos de cifrado, firma digital y verificación de la firma. El método *medirCifrado* ejecuta un Runnable de cifrado (AES o RSA) y mide su diración,. El método *medirFirma* ejecuta la firma digital y mide su duración. Por último, *medirVerificación* ejecuta la verificación de la firma y mide su tiempo.

UtilidadesProtocolo

Esta clase proporciona métodos de apoyo para la implementación del protocolo de comunicación. Maneja operaciones relacionadas con el intercambio de llaves de sesión usando Diffie-Hellman, el cifrado y descifrado de mensajes con AES, la protección de integridad mediante HMAC, la serialización y deserialización de mensajes, y el manejo de datos binarios a través de sockets.

Instrucciones

Para ejecutar el programa se tiene que correr primero la clase MainServidor.java y después se corre la clase clienteConcurrente.java

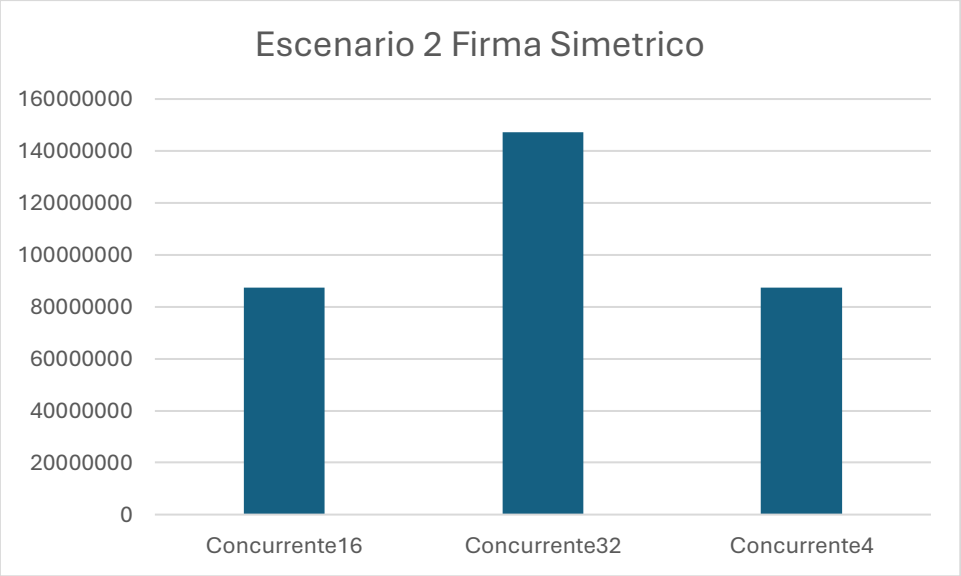
Tiempos

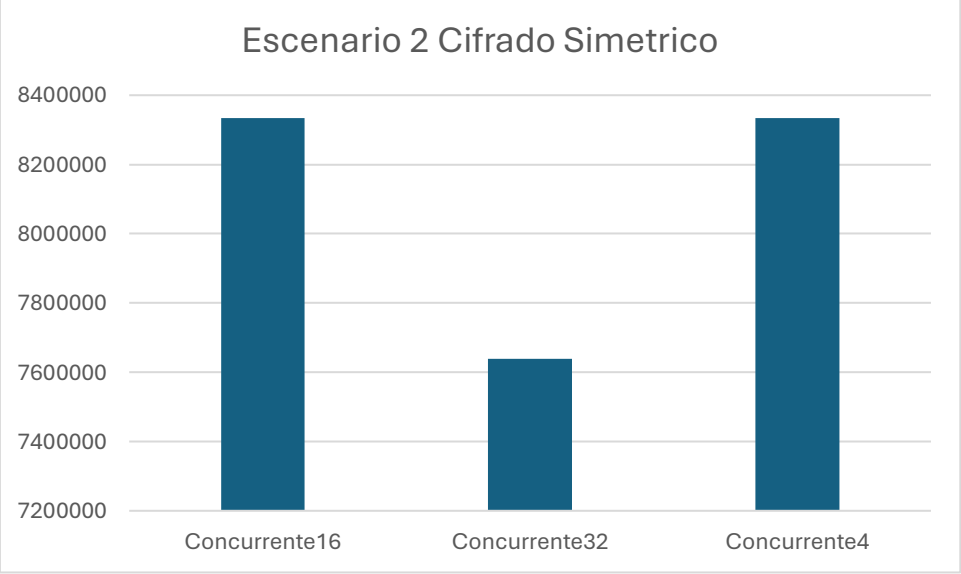
(i) Insertar tabla cifrar firmar verificar

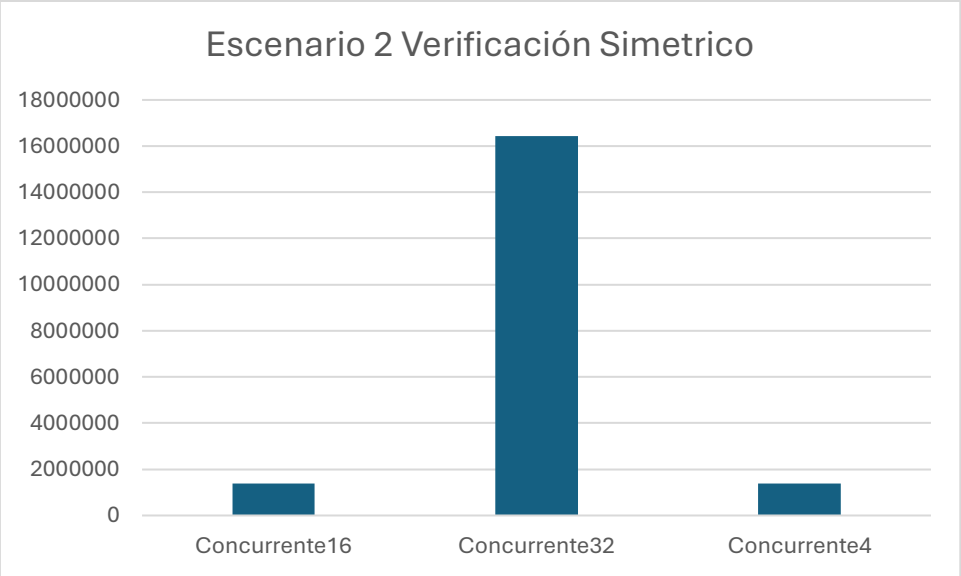
(ii)

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

3. 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 tiempo







(iv) Una que muestre los tiempos para el caso simétrico y el caso asimétrico en los diferentes escenarios.

5. Escriba sus comentarios sobre las gráficas, explicando los comportamientos observados.

Escenario para estimar la velocidad del procesador

El escenario definido para estimar la velocidad del procesador consiste en cifrar un mismo mensaje muchas veces (por ejemplo 1.000 veces) utilizando el mismo algoritmo y medir cuanto tiempo total tarda en completarse. A partir de este tiempo s epuede calcular cuantas operaciones de cifrado por segundo puede realizar la maquina. El procedimiento se lleva a cabo de la siguiente manera:

1. Antes de iniciar las operaciones de se toma el tiempo inicial usando System.nanoTime()
2. Después se cifra el mensaje 1.000 veces en un ciclo utilizando el algoritmo seleccionado (AES o RSA).
3. Se toma el tiempo final al terminar todas las operaciones.
4. Se calcula el tiempo total en segundos como:

$$\frac{Tiempo\ final - Tiempo\ inicial}{10^9}$$

(Se divide entre 10^9 para pasar de nano segundos a segundos)

5. Se calcula la cantidad de operaciones por segundo de la siguiente forma.

$$\frac{NúmeroOperaciones}{TiempoTotal(Segundos)}$$

Este escenario permite obtener una estimación del rendimiento del procesador para operaciones de cifrado. Es importante usar un mensaje pequeño y un numero grande de repeticiones para minimizar errores por variabilidad en el sistema operativo.

En la carpeta Escenario del proyecto se encuentra un programa que simula el escenario de prueba con un mensaje de 23 bytes y 1000 repeticiones.