Andrés Santiago Neira - 202123126

Juan Esteban López Ulloa – 202021417

Informe Caso 3

(i) La descripción de la organización de los archivos en el zip.

Para la implementación se realizó un proyecto Java que se subió a un repositorio en Github. El código realizado se encuentra en la carpeta source (src). Dentro de esta carpeta se encuentran dos carpetas las cuales simulan los equipos del caso; es decir una carpeta para servidor y otra carpeta para cliente.

En la carpeta servido se cuenta con una clase principal Servidor.java la cual contiene la información del servidor como el puerto (socket), numero de threads (delegados) y una llave privada la cual utilizaran los delegados para descifrar los retos para entablar conexión con los clientes. También se encuentra una clase ThreadServidor.java la cual, como dice su nombre es una clase que extiende de thread y simula los delegados que genera la clase Servidor para atender a cada uno de los clientes; de esta forma se maneja el servidor de forma concurrente para poder atender a varios clientes al tiempo. Adicionalmente se cuenta con una clase ProtocoloCliente.java la cual establece el protocolo de comunicación que maneja el cliente con el servidor y tiene toda la lógica de implementación de los algoritmos de seguridad junto con sus parámetros. Finalmente, en cuanto a la generación de las llaves públicas y privadas del paso 0 en el diagrama del enunciado se realizó otra clase llamada KeyGeneratos.java para generar estas llaves con la librería java.security; luego de generarlas se guarda la llave privada en la carpeta del servidor y la llave publica en la carpeta del cliente, ambas en un archivo de texto.

En la carpeta Cliente se tiene la clase principal Cliete.java la cual tiene la información del canal de comunicación (socket) con el servidor; además esta clase establece dicha conexión para que se dé la comunicación. También se tiene la clase ProtocoloCliente.java que, como la clase del protocolo del servidor, esta contiene la información de cómo es el protocolo de comunicación con el servidos y es la clase encargada de orquestar la comunicación; en esta clase también está la lógica que implementa los algoritmos de seguridad para cifrar y descifrar los mensajes.

Por otro lado, se realizó una carpeta adicional a las dos ya mencionadas en la que se implementaron las pruebas. La idea de esta carpeta fue segmentar el código ya realizado para realizar los diferentes tipos de pruebas. A esta segmentación de código se le agregaron algunos ciclos y un poco más del código para poder enviar diferentes clientes, peticiones, entre otros.

En cuanto a la lógica lo que se hizo fue diseñar y probar un sistema para la verificación segura de consultas de clientes en un servidor, utilizando cifrado simétrico (AES), como cifrado

Para la generación de parámetros y posterior generación de claves se realizaron los siguientes pasos:

- Para el cifrado simétrico, generamos una clave AES de 256 bits y un vector de inicialización (IV) de 16 bytes, así como una clave HMAC de 256 bits.
- Para el cifrado asimétrico, creamos un par de claves RSA (pública y privada) de 1024 bits para simular el cifrado de mensajes sensibles.

Por otro lado, para el cifrado de la información y la integridad de los datos trasmitidos mediante el canal establecido se realizó lo siguiente:

- Ciframos el id de usuario y el id de paquete utilizando la clave simétrica y generamos su HMAC correspondiente con la clave HMAC. Esto asegura que el servidor pueda descifrar y verificar la integridad de cada consulta recibida.
- La generación de HMAC protege contra modificaciones en el mensaje, ya que el servidor puede verificar que los datos no fueron alterados en tránsito.

En cuanto al proceso de verificación de consultas de los clientes sobre el estado de los paquetes se implemento usando los siguientes algoritmos:

- En cada consulta, el servidor descifra los id de usuario y paquete usando la clave AES y verifica el HMAC correspondiente para asegurar la integridad de los datos.
- Este proceso fue implementado en una clase Runnable para permitir la ejecución paralela mediante hilos, simulando la concurrencia de múltiples consultas simultáneas.

(ii) Las instrucciones para correr servidor y cliente, incluyendo cómo configurar el número de clientes concurrentes.

Para correr el programa generado el usuario se debe dirigir a la clase Servidor.java en la carpeta Servidor y correr la clase; una vez lo haga aparecerán dos opciones en la terminal, debe primero correr la opción 1 que es para crear el par de llaves (privada y pública).

Una vez se tengan los archivos con las llaves en cada una de las carpetas se puede ejecutar la opción 2 la cual lanza el servidor y lo deja escuchando para que cuando “llegue” un cliente este esté disponible.

Con el servidor disponible el usuario se puede dirigir a la clase Cliente.java en la carpeta Cliente y puede ejecutar ese archivo para crear un cliente. Una vez lo cree puede empezar hacer consultas mediante la terminal al servidor. En el programa no tiene un límite de clientes que pueda atender, el usuario que quiera usar el programa puede generar la cantidad de clientes que desee ejecutando la clase Cliente.java tantas veces como clientes quiera.

Para ver como son los datos y poder hacer inputs validos se sugiere revisar el archivo que se genera luego de correr el servidor con el nombre de “valoresBaseDatos.txt” en el cual se muestra el mapa que se realizó.

Las pruebas se hicieron en una carpeta aparte llamada Pruebas, donde hay un archivo para las pruebas iterativas y otros tres archivos para las pruebas usando Threads/delegados servidores.

NOTA: Este programa fue realizado en computadores Mac, se sugiere probarlo en uno de estos dispositivos ya que no tenemos la certeza de que funcione en otros sistemas operativos debido a que Mac ya tiene por defecto OpenSSL. En caso de probarlo en Windows se deberá agregar el PATH en el archivo ProtocoloServidor.java en la línea 57.

(iii) Tablas de datos.

32 consultas de un solo cliente:

Pruebas Iterativas (32 consultas, mismo cliente, mismo deldegado)		
escenario	**tiempo total (ms)**	**tiempo promedio (ms)**
Responder el reto	9	0.28
Generar G, P y G^X	21908	684.625
Verificar consulta	8	0.25

Tiempo de respuesta del reto:

Tiempo de respuesta del reto	
Número de delegados/Threads	**Tiempo total (ms)**
4	8
8	9
32	14
100	25

Tiempo de respuesta para generación de parámetros:

Tiempo de generación de G, P y G^X	
Número de delegados/Threads	**Tiempo total (ms)**
4	1833
8	2433
32	4401
100	55647

Tiempo de verificación de la consulta:

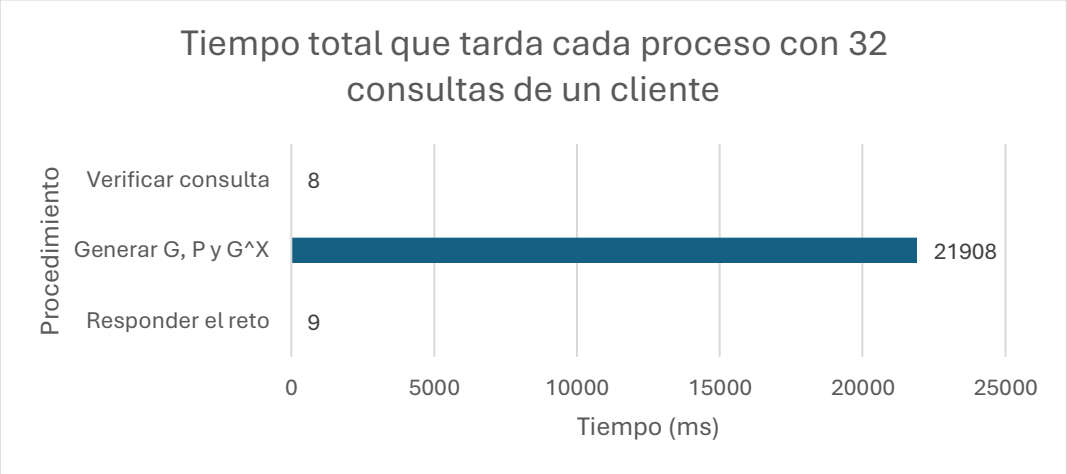
Tiempo de verificación de la consulta	
Número de delegados/Threads	**Tiempo total (ms)**
4	3

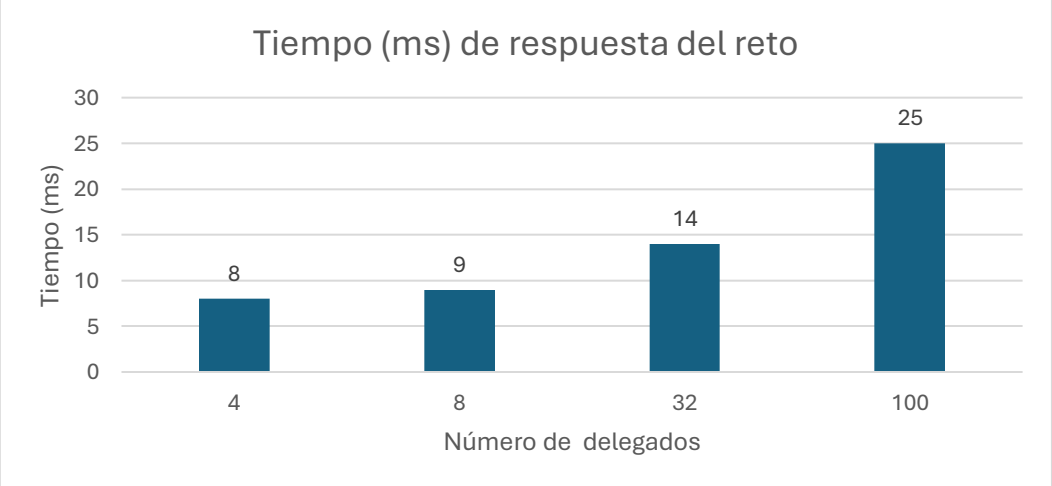
8	6
32	7
100	18

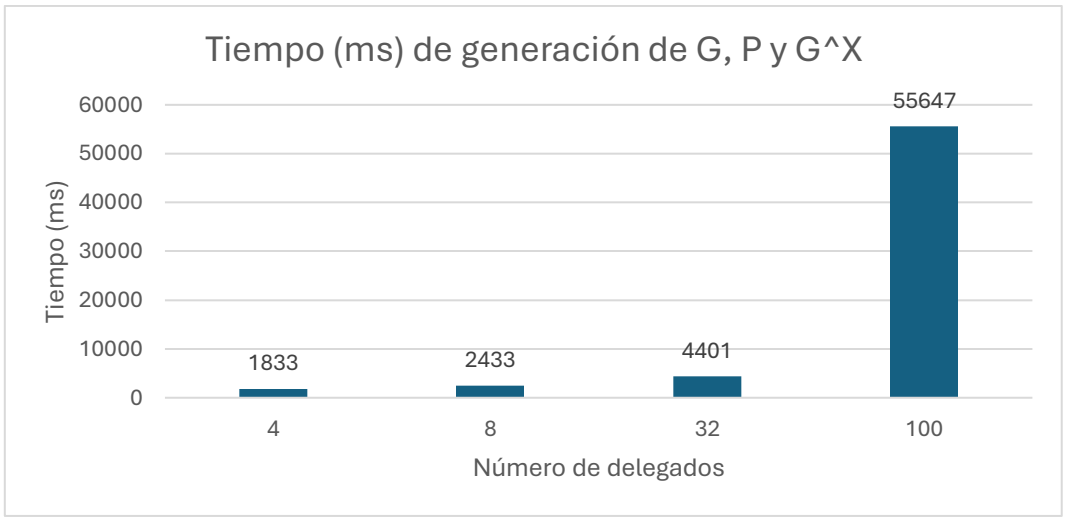
Comparación de cifrado simétrico y asimétrico:

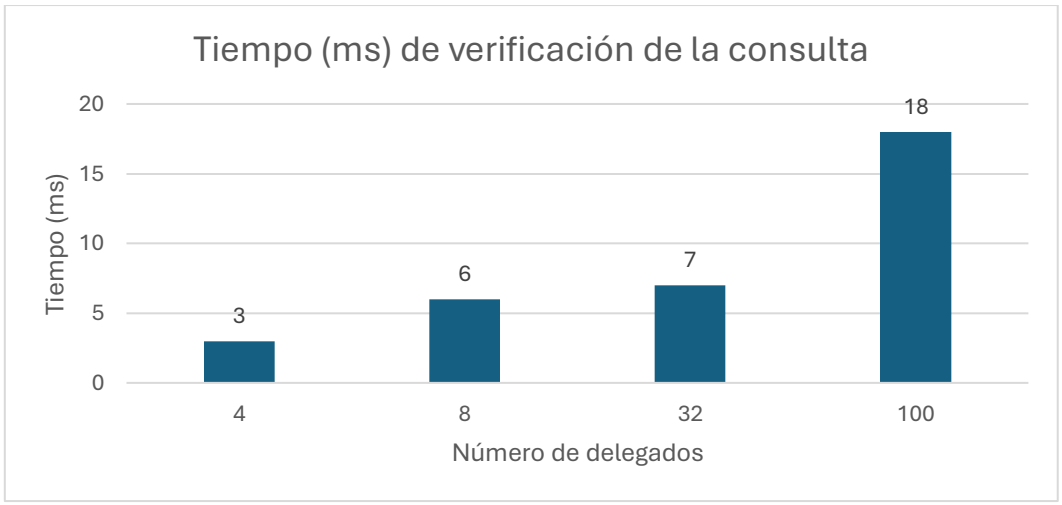
Cifrado		**Iteraciones**
Simétrico	**Asimétrico**	
29	14	100
26	33	1000
27	190	10000
26	1088	100000

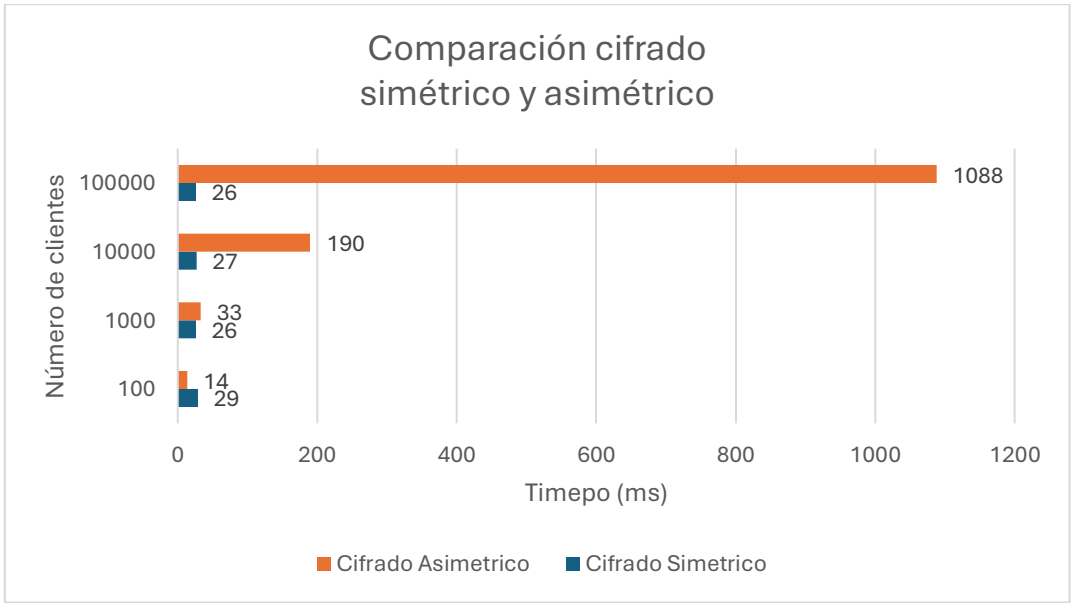
(iv) Gráficas






Tiempo (ms) de generación de G, P y G^X
Tiempo (ms)
60000
50000
40000
30000
20000
10000
0
1833
2433
4401
55647
4
8
32
100
Número de delegados


Tiempo (ms) de verificación de la consulta
Tiempo (ms)
20
15
10
5
0
3
6
7
18
4
8
32
100
Número de delegados


Comparación cifrado simétrico y asimétrico
Número de clientes
100000
10000
1000
100
1088
26
190
27
33
26
14
29
0
200
400
600
800
1000
1200
Timepo (ms)
Cifrado Asimetrico
Cifrado Simetrico

(v) Respuestas a las preguntas planteadas.

En la primera grafica que se muestra del tiempo que tarda cada uno de los procesos (verificación de consulta, Generación de parámetros y Repuesta del resto) cuando un cliente realiza 32 consultas se puede evidenciar que el proceso que mas tiempo tardo fue la generación de los parámetros tomando 621 milisegundos, más de medio segundo lo cual es una cantidad considerable teniendo en cuenta que los otros dos procesos estudiados no sobrepasaron los 10 milisegundos.

En general se puede ver en todas las graficas que a medida que el numero de delegados (tanto clientes como servidores) aumenta, el tiempo aumenta. Esto es coherente pues al aumentar la cantidad de personas aumenta la cantidad de solicitudes y por lo tanto el tempo total que se demorará en programa en ejecutarse aumenta. Además, se puede ver que, como se dijo anteriormente, el proceso que mas tiempo consume es el de generación de los parámetros para la llave simétrica. Sin embargo, en la última tabla que hace referencia a la ultima grafica se puede ver que el algoritmo simétrico es mucho más rápido que el algoritmo asimétrico cuando se trabaja con varias iteraciones (varios clientes); es decir varios clientes realizando consultas. Por lo tanto, se puede concluir, que si bien el cifrado simétrico funciona mucho más rápido al momento de cifrar y descifrar los estados, hay que tener en cuenta que para cada cliente es importante generar los parámetros para hacer posible este algoritmo simétrico, por lo que debe tenerse precaución con la elección del método, ya que si el sistema a diseñar tiene que generar claves muy seguido puede llegar a desearse el algoritmo asimétrico. Esto puede sonar contradictorio dados los datos que se muestran en la última tabla, sin embargo, debe tenerse en cuenta que solo se toma el tiempo de cifrado y no de generación de parámetros para encontrar la llave compartida.