**Informe: Caso 3**

**Carlos Eduardo Ramírez Martínez (201921729)**

**Néstor Felipe González García (201912670)**

**Instrucciones para correr el prototipo.**

Se debe correr la clase ‘Cliente’, ya que esta contiene el método main(). Una vez se corra esta clase, aparecerá un mensaje preguntando que tipo de cifrado desea: si se escribe ‘true’ usará cifrado simétrico, de lo contrario, si se escribe ‘false’ usará cifrado asimétrico. Seguidamente, se le pedirá cuantos clientes desea ejecutar, aquí debe ingresar el número deseado. La respuesta del programa serán los ID’s de los clientes, junto con el mensaje solicitado. Este mensaje se muestra con el siguiente formato:

Mensaje del cliente <id>: <mensaje solicitado>

**Organización de archivos del programa.**

El programa consta de cuatro (4) clases: ‘Cliente’, ‘Repetidor’, ‘Servidor’, ‘Util’. Las primeras tres clases representan los participantes de la comunicación con sus respectivos ID’s, sockets y protocolos. La clase ‘Util’ contiene variables globales al programa como las referencias a las properties del las llaves y los mensajes, métodos de acceso a las llaves y métodos de cifrados generales. En la carpeta ‘data’ se encuentran los archivos .properties que contienen las llaves generadas por el programa y los 10 mensajes con su respectivo ID.

**Esquema de generación y almacenamiento de llaves.**

En primera instancia, el método main se encuentra dentro de la clase ‘Cliente’ y se encarga de tomar los datos del usuario, descritos en el inciso anterior y, a partir de las opciones seleccionadas, se encarga de generar las llaves para cada cliente y la(s) llave(s) para el servidor y el repetidor. Para las llaves simétricas se utiliza ‘AES’ para la generación de llaves, y ‘RSA’ para el caso asimétrico; la clase KeyGenerator de la librería Java.crypto es la encargada de generar las llaves de ambos protocolos. Cabe aclarar que cada cliente tiene una llave única, sin embargo, los servidores (servidor padre y sus delegados) tienen una única llave, así como sus homólogos repetidores. Las llaves generadas son almacenadas en un archivo properties donde serán accedidas por el resto de las clases. Finalmente, el método main crea e inicializa los clientes, repetidor y servidor.

**Análisis de ejecución.**

Para la medición de tiempo, se utilizó ‘System.currentTimeMillis()’ al inicio y al final de los intervalos requeridos, y se suma la diferencia de estos tiempos a una variable global.

Imagen que contiene Texto

Descripción generada automáticamente

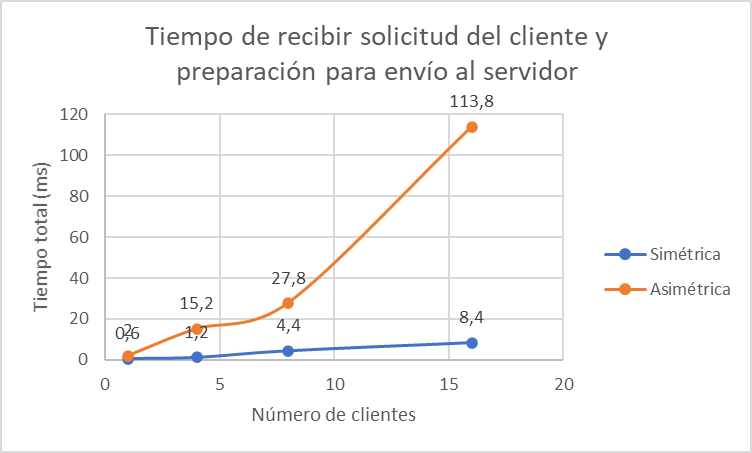
Figura 1: Ejemplo de toma de tiempo.

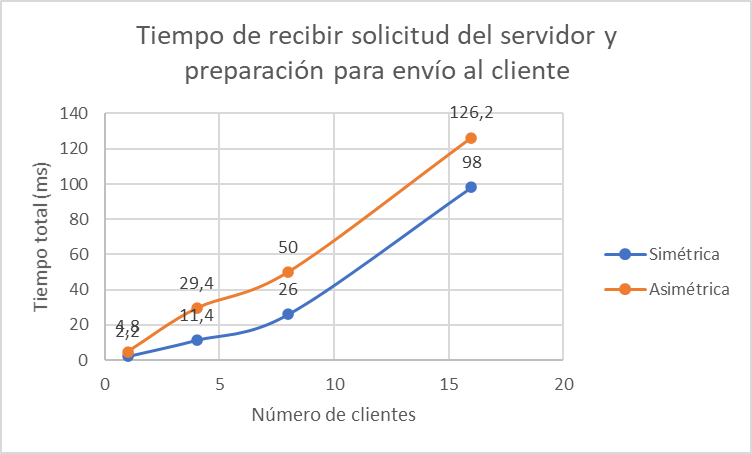
Para cada caso requerido, se realizaron 5 pruebas y se registró además el promedio de los resultados. La información está resumida en las siguientes tablas (tiempo en milisegundos).





A partir de estos datos, se construyeron las siguientes gráficas:





A partir de lo anterior, se puede evidenciar que el cifrado asimétrico toma más tiempo que el cifrado simétrico, debido a que las operaciones que involucra el asimétrico son de mayor complejidad. Además, como era de esperarse, entre mayor número de clientes se ejecuten, mayor será el tiempo total que tome la gestión de las peticiones. Finalmente, se evidenció que, en la gran mayoría de los casos, el tiempo de gestión (recibir, descifrar y volver a cifrar) para el mensaje enviado por el cliente fue menor que el tiempo de gestión para el mensaje enviado por el servidor, esto debido a que el mensaje enviado por el cliente es de una longitud menor que el mensaje enviado por el servidor.

**Estimación velocidad del procesador.**

Las pruebas de ejecución del programa fueron ejecutadas en una máquina con un procesador Intel Core i5-8250 de 1.6GHz, sin embargo, este procesador permite como frecuencia máxima 3.4GHz. Se observó que el procesador corre a una frecuencia promedio de 3.3 GHz como se observa en la siguiente imagen, tomada durante la realización de las pruebas.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

A partir de los tiempos tomados en el análisis del inciso anterior, se encontró que el tiempo promedio para gestionar (recibir, descifrar y volver a cifrar) un mensaje fue de 1.4 ms para simétrico y 3.4 ms para el asimétrico. Estos datos fueron extraídos al sacar el promedio de los dos tiempos obtenidos para 1 cliente, para ambos modos de cifrado. Se realizaron, a partir de estos datos, los siguientes cálculos:

**Referencias.**

Oracle (2020) Java™ Cryptography Architecture Standard Algorithm Name Documentation. Recuperado de: [https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames .html#Cipher](https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames%20.html#Cipher)