Informe Caso # 3

Adrián Arturo Suárez García - 202123771

Kalia Angelica González Jiménez – 202226917

1. *Organización de Archivos.*
   En la carpeta se puede encontrar dos carpetas principales para la resolución del caso. La primera es "Cliente" que contienen los archivos Cliente y ThreadCliente. La primera clase nos ayudara para realizar la conexión al servidor y crear los threads de este para que esta solución sea concurrente; simula muchos clientes haciendo peticiones. La segunda es donde se va a realizar todo el protocolo y las acciones que este debe aplicar en él; esta clase conecta con la clase "algoritmos", explicada abajo.

   Después tenemos la carpeta "Servidor" que está compuesta por los archivos ThreadServPrincipal, Algoritmos y ServPrincipal. La primera clase nos ayuda a convertir la funcionalidad en algo concurrente, es decir, asigna a cada cliente nuevo un thread aparte, simulando el servidor. La segunda clase están los algoritmos de seguridad que van a usar ambas partes del protocolo, creando todas las llaves necesarias, cifrados, descifrados, y demás funciones auxiliares. Y la última es el Servidor Principal, donde se crea la tabla de servicios y se las envía a cada servidor delegado (thread) para que realice el protocolo; cabe notar que esta clase, al manejar los threads de los servidores, usa una barrera para que todos terminen antes de imprimir los resultados de tiempos.

   Por último, se tiene dos archivos donde uno representa la llave privada del servidor y la otra su llave pública. Estas están en llavePrivada.key y llavePublica.key.

2. *Instrucciones para correr el programa y configurar los clientes.*
   Pasos para correr el programa:
   - Dirigirse al archivo ServPrincipal y dar correr o "Run".
   - Luego, dirigirse al archivo Cliente y dar correr o "Run"
     - En el archivo hay instrucciones para ejecutar cada uno de los escenarios con distintos números de clientes; en resumen, si "secuencia" es 1, se cambian los números de threads (escenario 2), y si "numCli" es 1, se cambia la secuencia (escenario 1).

   Después de realizar dichos pasos, en la terminal del servidor mostrará la conexión de cada nuevo thread y al final los tiempos finales solicitados más adelante.

   Instrucciones en configuración de clientes:

   - Diríjase a la clase Cliente donde esta tiene un atributo llamado numCli que dice la cantidad de clientes que están solicitando el servicio.
   - En dicho atributo puede poner la cantidad de clientes que desee.

```java
int numCli = 1;

try {
    //crear el socket en el lado del cliente
    for (int i = 0; i < numCli; i++) {
        socket = new Socket(SERVIDOR, PUERTO);
        String ipCliente = "3";
        new ThreadCliente(socket, ipCliente, idCli).start(); // Crear un nuevo hilo para manejar al cliente
        idCli ++;
        System.out.println("Cliente" + idCli + " conectado");
    }
} catch (Exception e) {
    e.printStackTrace();
    System.err.println("Error al conectar el cliente: " + e.getMessage());
}
```

3. *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 este punto, vamos a suponer el siguiente escenario:
   Se quiere cifrar un bloque de texto de 1KB usando el algoritmo simétrico de AES y RSA para el asimétrico. Se va a realizar el procedimiento 10000 veces y medir el tiempo total que se demora.
   Ya con este enunciado, queremos medir el tiempo entre un cifrado simétrico y otro asimétrico. Toca tener en cuenta que el primero es más rápido que el segundo.
   Para poder hallar los cálculos, se realiza la siguiente formula:

$$Operaciones\ por\ segundo\ =\ \frac{\#Operaciones}{Tiempo\ total\ (sg)}$$

   Se aplica esta fórmula para medir la velocidad de cada algoritmo. Su pongamos que AES se demora 2sg y RSA se demora 4sg

   Resultado AES:

$$\frac{10000}{2(sg)} = 5000\ \frac{operaciones}{segundo}$$

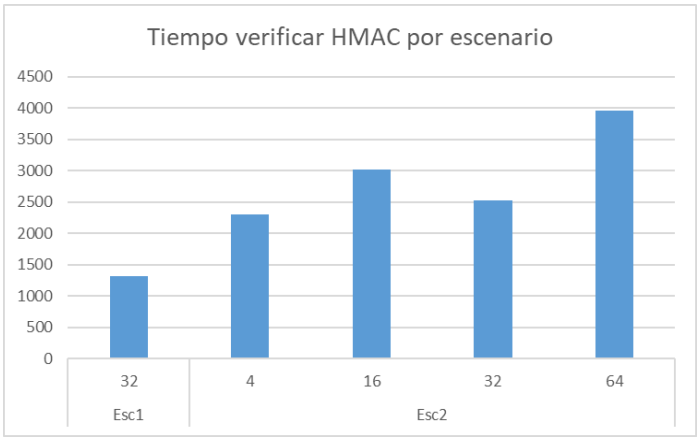
   Resultado RSA:

$$\frac{10000}{4(sg)} = 2500\ \frac{operaciones}{segundo}$$

   Ya con este ejemplo, nos damos cuenta del porque el cifrado simétrico es más veloz que el asimétrico.

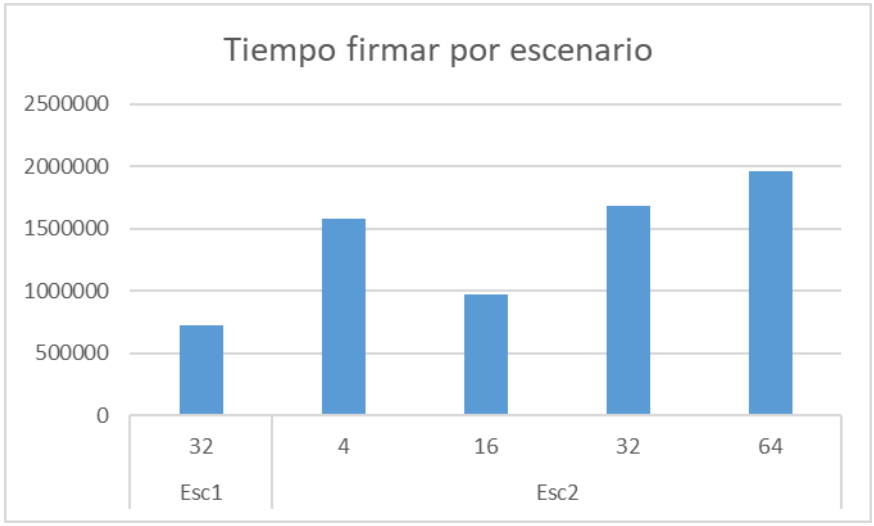
4. *Análisis Comportamiento de graficas:*

Tiempo verificación HMAC



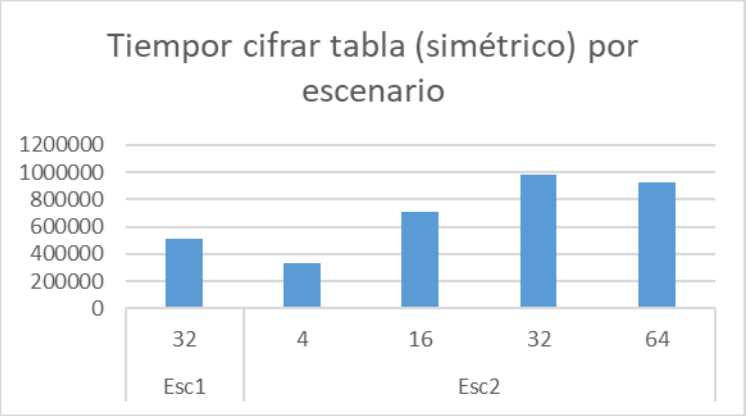
En la tabla, se observa que las peticiones secuencias son las que resuelven las 32 peticiones a un menor tiempo comparado con el otro escenario. Lo que significa que el programara es eficiente en cuanto al tiempo total acumulado.
En la concurrencia, a medida que aumentan las peticiones también aumenta el tiempo de verificación. Este aumento no es lineal dado a que por ejemplo con 32 peticiones se tarda menos que con 16.

Tiempo firmar



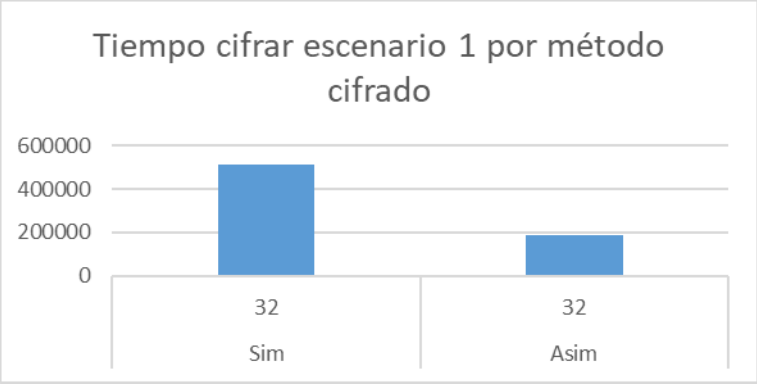
En la tabla vemos que el tiempo de la firma en el escenario secuencial es menor a los concurrentes. Por el otro lado, en el escenario concurrente vemos que las 64 peticiones tuvieron una alta cantidad de tiempo y que también hubo un descenso con 16. Lo que concluimos que el programa puede ser eficiente o no dependiendo de la cantidad de solicitudes.

Tiempo Cifrar (simétrico)



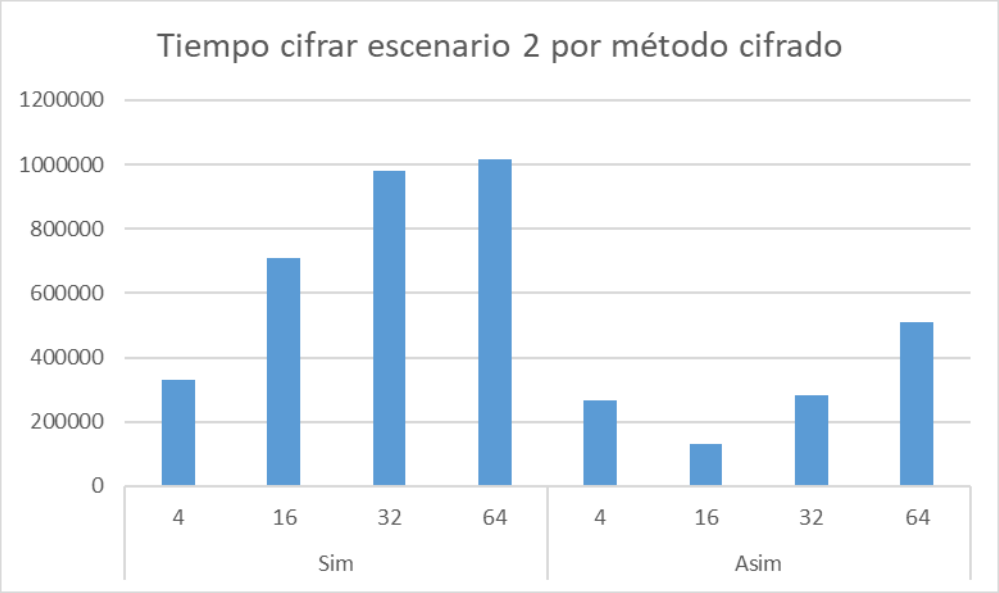
En esta tabla vemos como el algoritmo simétrico afecta en el tiempo de ejecución del algoritmo. Para ello, en el escenario 1 vemos que tuvo un promedio de 500000 en total. En el segundo caso, observamos que al realizar 4 peticiones concurrentes obtiene la menor cantidad de tiempo lo que significa que el algoritmo de AES tuvo un buen rendimiento.

Tiempo cifrar con clientes secuenciales por método de cifrado



Proseguimos a comparar el rendimiento del cifrado simétrico y asimétrico. Para este caso, cuando son secuenciales, el algoritmo más eficiente es el asimétrico dado a que el manejo de concurrencia permite optimizar los tiempos de respuesta antes las peticiones realizadas por los diferentes clientes.

Tiempo crifrar clientes al tiempo por método cifrado



En esta última tabla, observamos que en el escenario 2 el algoritmo asimétrico tiende a ser el más eficiente y veloz a la hora de responder las solicitudes. Esto se debe a que, al manejar concurrencia, se atiende mejor las peticiones de cada cliente. Por el otro lado, cada thread debe esperar a que uno acabe para que sea atendido.