

Caso de Estudio 3 – Canales Seguros Sistema de rastreo de paquetes en una compañía transportadora

Isabella Caputi- 202122075

Sofía Vásquez- 202123910

1. La descripción de la organización de los archivos en el zip

En el código fuente del caso, se pueden observar 5 clases en el src. Para empezar, está el GeneradorLlaves, el cual tiene un método main encargado de generar la llave pública y la llave privada para el cifrado. Además, está la clase de utilidades de cifrado, que tiene todos los algoritmos y métodos utilizados en el protocolo de la comunicación entre el ServidorPrincipal y el cliente. Adicionalmente, está la clase Servicio, la cual se utiliza para crear los servicios de la tabla que se le pasa al cliente durante el protocolo de comunicación con el servidor, para que el cliente pueda escoger un servicio.

Las clases más importantes son el ServidorPrincipal, el Cliente y el ClienteHandler. El ServidorPrincipal se corre y se conecta al puerto predeterminado. Además, inicializa un ClienteHandler (Thread), por el cual se hará la comunicación del protocolo con cada consulta de cliente. Luego el cliente se corre y se comunica con el servidor a través del mismo puerto, en este se escoge que escenario se desea (cliente con consultas secuencial o clientes concurrentes). Luego, se resuelven la cantidad de consultas necesarias o se atienden los clientes necesarios dependiendo del escenario.

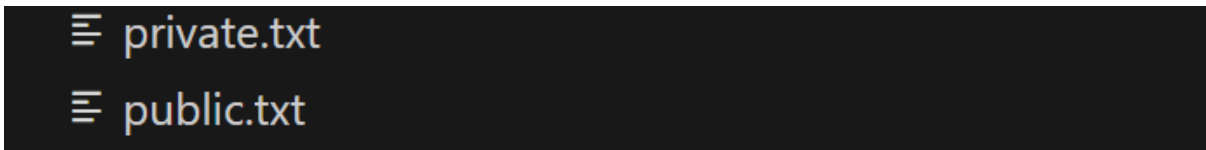
Por último, en el .zip también se puede encontrar una carpeta de docs, donde se encuentra este informe y un excel (“Datos Caso 3”) donde se hizo la recopilación de datos y las gráficas presentadas más adelante.

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

Realizamos un video de guía de como correr la aplicación:

<https://drive.google.com/file/d/1hU34PrfeYDBSf6XGBs-ou3fdn3Dzm6-H/view?usp=sharing>

Lo primero debería hacer para empezar la ejecución del programa es correr la clase GeneradorLlaves.java para crear las llaves. En este caso, **ya** están creada y aparecen en el programa de la siguiente manera (como las llaves ya estan creadas no es necesario correr esta clase):



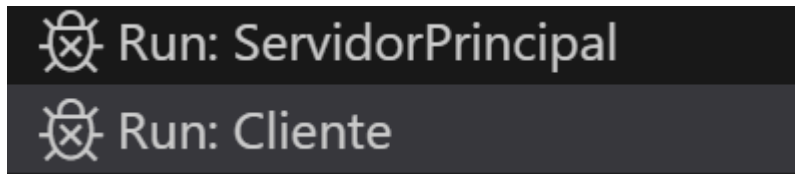
```
≡ private.txt
≡ public.txt
```

Para ejecutar el programa, diríjase a la clase `ServidorPrincipal.java` y ejecute el método `main()` haciendo clic en el icono de *play* en la parte superior derecha. En la terminal aparecerá lo siguiente:

```
PROBLEMS 30 OUTPUT TERMINAL DEBUG CONSOLE SQL HISTORY PORTS TASK MONITOR Run: ServidorPrincipal + - [Icons] x
indows [Version 10.0.22631.5039]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Sofia Toro\Documents\GitHub\Caso3_i.caputi_s.vasquez> cmd /C ""C:\Program Files\Java\jdk-21\bin\java.exe" -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Sofia Toro\AppData\Roaming\Code\User\workspaceStorage\937088da8f3e3f570b5506ae21e7ab6\redhat.java\jdt_ws\Caso3_i.caputi_s.vasquez_fd18d84f\bin" ServidorPrincipal "
Servidor listo y esperando en el puerto: 5000
```

Posteriormente, se debe ejecutar el `main()` de la clase `Cliente.java` de la misma forma, haciendo clic en *play*. Esto abrirá una nueva terminal donde se va a poder ver todos los pasos que ejecuta el cliente. Se deben ver las dos terminales de esta forma:



La terminal del cliente despliega del siguiente menú donde se puede elegir entre los dos escenarios:

- (i) Un servidor de consulta y un cliente iterativo. El cliente debe generar 32 consultas secuenciales.
- (ii) Servidor y clientes concurrentes. El número de delegados, tanto servidores como clientes, debe variar entre 4, 16, 32 y 64 delegados concurrentes. Cada servidor delegado atiende un solo cliente y cada cliente genera una sola solicitud.

```
C:\Users\Sofia Toro\Documents\GitHub\Caso3_i.caputi_s.vasquez> cmd /C ""C:\Program Files\Java\jdk-21\bin\java.exe" -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Sofia Toro\AppData\Roaming\Code\User\workspaceStorage\937088da8f3e3f570b55069ae21e7ab6\redhat.java\jdt_ws\Caso3_i.caputi_s.vasquez_fd18d84f\bin" Cliente "
=== CLIENTE DE PRUEBA DE RENDIMIENTO ===
Escoja el escenario de prueba:
1. Cliente iterativo (32 consultas secuenciales)
2. Clientes concurrentes (4, 16, 32 o 64 clientes)
>
```

Al elegir una opción, inmediatamente se empezará la ejecución de cada uno de los protocolos solicitados. Por ejemplo, al ejecutar la primera opción, en el terminal del cliente se van a generar las consultas de los 32 clientes 1 por una.

```

2. Clientes concurrentes (4, 16, 32 o 64 clientes)
> 1
Llave pública RSA del servidor cargada correctamente

=== EJECUTANDO CLIENTE ITERATIVO (32 CONSULTAS) ===

--- Consulta 1 de 32 ---
Cliente 1 iniciando conexión...
Llave pública RSA del servidor cargada correctamente
0a. Se leyo la llave publica de archivo exitosamente.
Conectado al servidor principal: localhost:5000
1. Se envio HELLO a servidor exitosamente.
2a. Reto generado exitosamente: 3998271023838568826
2b. Se envio Reto a Servidor.
4. Recibio correctamente Rta del Servidor.

```

Cuando termine de ejecutar el Cliente.java, para volver a correrlo debe espichar la fecha de arriba de su teclado (pg up) para volverlo a correr. Deberá salirle este comando y debe hacer enter

```

C:\Users\Sofia Toro\Documents\GitHub\Caso3_i.caputi_s.vasquez> cmd /C ""C:\Program Files\Java\jdk-21\bin\java.exe
" -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Sofia Toro\AppData\Roaming\Code\User\workspaceStorage\9370
88da8f3e3f570b55069ae21e7ab6\redhat.java\jdt_ws\Caso3_i.caputi_s.vasquez_fd18d84f\bin" Cliente "

```

En el caso de la opción 2, los clientes correrán todos al mismo tiempo de la siguiente manera:

```

Ingrese el número de delegados concurrentes (4,16,32,64):
>
4
Llave pública RSA del servidor cargada correctamente

=== EJECUTANDO 4 CLIENTES CONCURRENTES ===
Cliente 2 iniciando conexión...
Cliente 3 iniciando conexión...
Cliente 1 iniciando conexión...
Cliente 4 iniciando conexión...
Llave pública RSA del servidor cargada correctamente
Llave pública RSA del servidor cargada correctamente
0a. Se leyo la llave publica de archivo exitosamente.
Llave pública RSA del servidor cargada correctamente

```

NOTA: Si desea volver a correr el programa una vez más, debe volver a correr primero el servidor y luego el cliente (no intente correr el cliente sin volver a correr el servidor antes, no va a funcionar).

Tenga en cuenta que unos prints estarán del lado del servidor y otros del lado de los clientes. No debe terminar la terminal del ServidorPrincipal.java ya que parara de funcionar el puerto 5000. Esto a menos que desee ejecutar otro escenario de cliente, ahí si debe terminarla y luego correr el cliente de nuevo.

3. Respuestas a todas las tareas y preguntas planteadas en este enunciado

Corra su programa en diferentes escenarios y mida el tiempo que el servidor requiere para: (i) Firmar, (ii) cifrar la tabla y (iii) verificar la consulta. Los escenarios son:

(i) Un servidor de consulta y un cliente iterativo. El cliente debe generar 32 consultas secuenciales.

(ii) Servidor y clientes concurrentes. El número de delegados, tanto servidores como clientes, debe variar entre 4, 16, 32 y 64 delegados concurrentes. Cada servidor delegado atiende un solo cliente y cada cliente genera una sola solicitud.

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 tiempos.

A partir del enunciado presentado, construimos las siguientes dos tablas recopilando los datos obtenidos. La primera muestra los tiempos de firmar, de cifrar la tabla, de verificar la consulta, de cifrado simétrico y cifrado asimétrico (todos en ms) obtenidos para la respuesta en el escenario de servidor y clientes concurrentes. La segunda muestra los mismos tiempos (en ms) para el servidor de consulta y el cliente iterativo (32 consultas).

Escenario Servidor y Clientes Concurrentes					
Delegados	Tiempo para firmar (ms)	Tiempo para cifrar la tabla (ms)	Tiempo para verificar consulta (ms)	Tiempo cifrado simetrico (ms)	Tiempo cifrado asimetrico (ms)
4	1,25	1	2,25	0,25	5
4	1	1,25	0,5	0	0,25

4	1,25	4,25	1	0,5	3,25
4	1,25	2,5	1	0	1,25
4	4,5	3,25	0,75	0,25	2,5
4	2	2,5	0,75	0,5	1,5
Promedio (4 delegados)	1,875	2,458333333	1,041666667	0,25	2,291666667
16	2,8125	0,625	6,5625	0,625	6,875
16	14,375	0,3125	7	0,125	4,625
16	6,75	2,0625	5,25	0,0625	5,8125
16	2,125	0,75	2,875	0,1875	2,9375
16	1,25	0,875	0,5625	0,4375	2,9375
16	3,125	0,4375	8,375	0,125	3,9375
Promedio (16 delegados)	5,0729166 67	0,84375	5,104166667	0,260416667	4,520833333
32	1	0,84375	19,25	0,65625	22,65625
32	63,1875	1,9375	17,75	0,09375	13,8125
32	3,53125	38,3125	7,3125	0,28125	10,5
32	2,71875	17,34375	43,03125	0,1875	23,3125
32	77,375	0,5625	23,0625	0,21875	22,625
32	8,8125	0,3125	10,5625	0,21875	8,375
Promedio (32 delegados)	26,104166 67	9,885416667	20,16145833	0,276041667	16,88020833
64	186,82812 5	30,890625	56,046875	0,25	34,140625
64	15,34375	133,78125	35,5625	0,375	34,625
64	152,25	0,28125	2,046875	0,25	1,9375
64	1,015625	0,109375	27,03125	5,796875	9,359375
64	121,79687 5	150,46875	25,921875	0,140625	12,109375
64	1,046875	0,328125	20,5	0,28125	17,609375
Promedio (64 delegados)	79,713541 67	52,64322917	27,8515625	1,182291667	18,296875

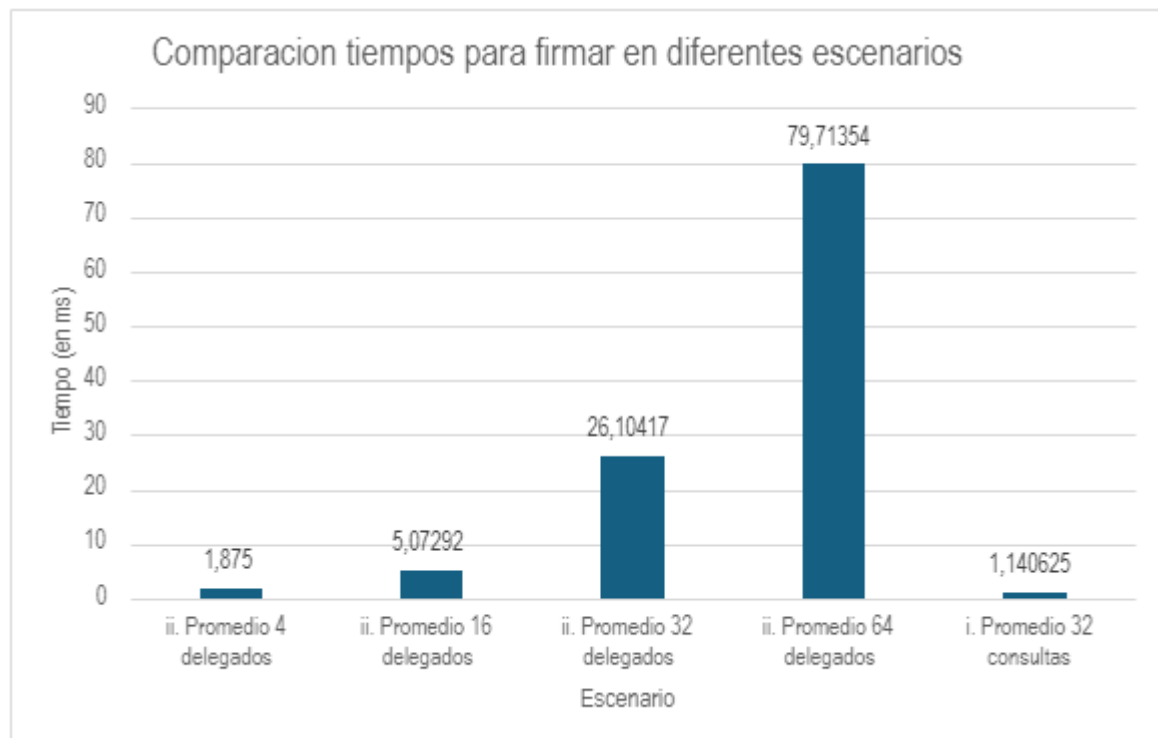
	Escenario Servidor y Cliente Iterativo
--	---

Consultas	Tiempo para firmar (ms)	Tiempo para cifrar la tabla (ms)	Tiempo para verificar consulta (ms)	Tiempo cifrado simetrico (ms)	Tiempo cifrado asimetrico (ms)
32	1,125	0,4375	0,1875	0,15625	0,46875
32	1,6875	0,9375	0,28125	0,03125	0,46875
32	0,8125	0,6875	0,1875	0,28125	0,3125
32	1,71875	0,96875	0,25	0,28125	0,625
32	0,9375	0,46875	0,40625	0,3125	0,25
33	0,5625	0,75	0,28125	0,46875	0,4375
Promedio (32 consultas)	1,140625	0,7083333333	0,265625	0,2552083333	0,4270833333

4. Construya las siguientes gráficas:

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

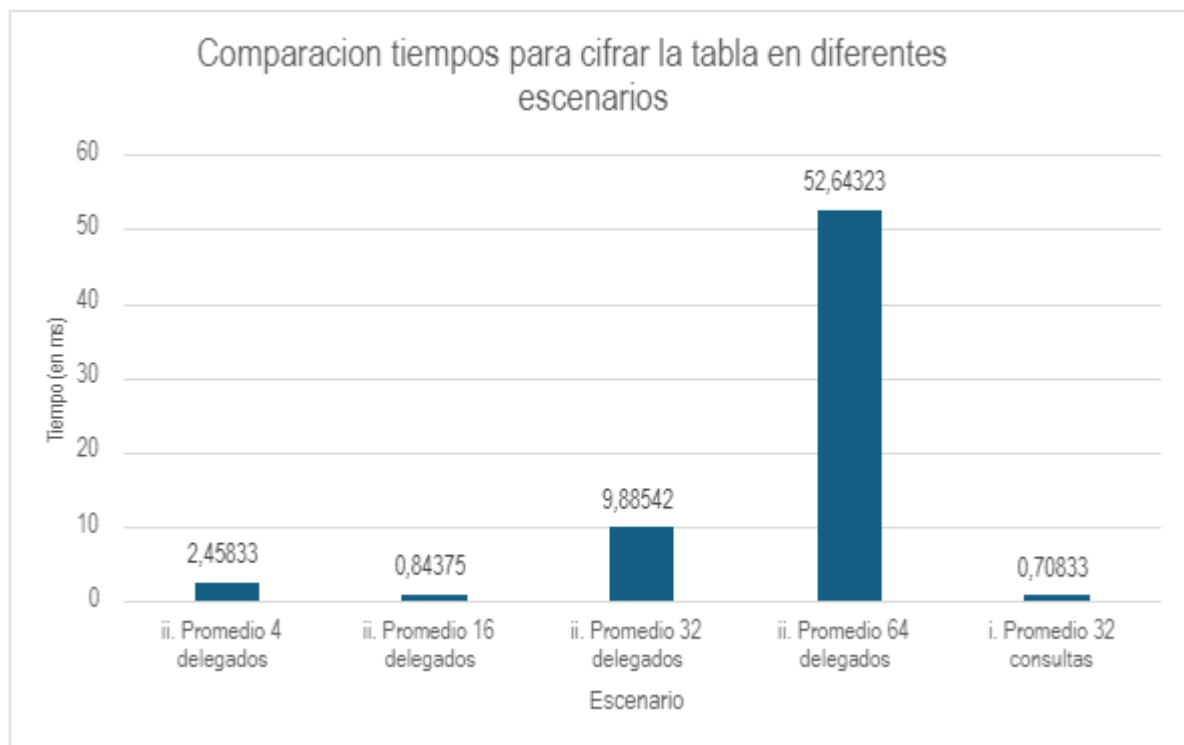
(i) Una que compare los tiempos para firmar en los escenarios



Se hizo esta gráfica a partir del promedio total encontrado entre varias ejecuciones (entre las corridas diferentes presentadas en las tablas anteriores). Se puede observar que, en promedio, a medida que aumentan los delegados en el escenario (ii), el tiempo

para firmar también aumenta significativamente, pasando de 1.875 ms con 4 delegados a 79.71 ms con 64 delegados, lo que indica una mayor complejidad y carga en el proceso de firma. Esto se debe a que, al incrementarse el número de delegados concurrentes, múltiples procesos de firma deben ejecutarse simultáneamente, generando una competencia por los recursos del procesador y provocando un mayor sobre carga de contexto y sincronización. Firmar implica operaciones criptográficas complejas, como cifrado RSA, que son especialmente sensibles a la saturación de CPU. En contraste, el escenario iterativo (i) presenta el menor tiempo promedio (1.14 ms), ya que las 32 consultas se procesan de forma secuencial, permitiendo que el servidor dedique toda su capacidad a una firma a la vez sin interferencia de otros procesos. Esto optimiza el uso de recursos del sistema y minimiza los tiempos de respuesta. Por esta razón es que las diferencias de rendimiento entre la ejecución concurrente y la ejecución secuencial son notables.

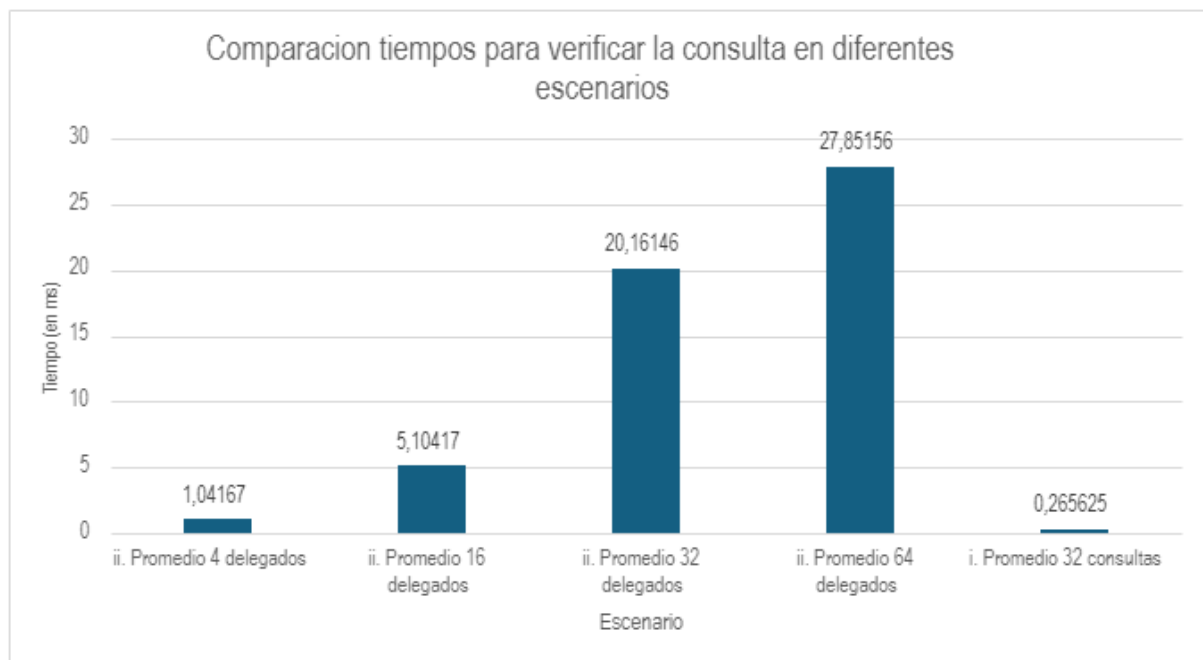
(ii) Una que compare los tiempos para cifrar la tabla en los escenarios



La gráfica muestra un comportamiento levemente diferente a la anterior. En esta, se ve que el tiempo promedio para cifrar la tabla varía según el número de delegados del escenario (ii), siendo mayor con 64 delegados (52.64 ms) y menor en el escenario de 32 consultas iterativas (0.708 ms). A diferencia de la operación de firma, aquí no se observa un crecimiento constante: el tiempo disminuye inicialmente de 4 a 16 delegados y luego vuelve a aumentar: sube para 32 delegados y especialmente en 64 delegados. Esto indica que el proceso de cifrado de la tabla no depende únicamente de la cantidad de delegados concurrentes, sino también de otros factores como la distribución de carga,

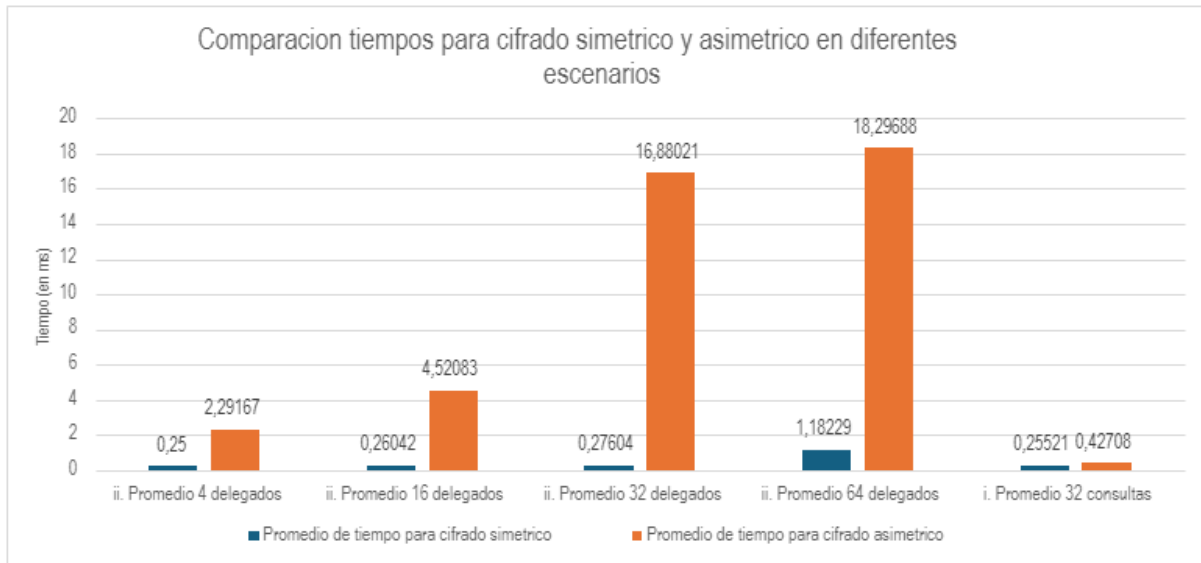
la eficiencia del sistema operativo en el manejo de threads y posibles variaciones en la disponibilidad de recursos en el servidor durante la concurrencia. De todas formas, se mantiene la tendencia general de que el escenario iterativo (i), donde las consultas son procesadas una a una, logra un menor tiempo de procesamiento. Esto se debe a que evita la carga adicional de la concurrencia masiva y permite un uso más eficiente de la capacidad de procesamiento para operaciones individuales de cifrado.

(iii) Una que compare los tiempos para verificar la consulta en los escenarios



Al igual que la gráfica del tiempo de la firma, la gráfica del tiempo promedio para verificar una consulta muestra un aumento notable conforme crece el número de delegados en el escenario (ii), pasando de 1.042 ms con 4 delegados a 27.852 ms con 64 delegados. Esto evidencia que el proceso de verificación también es altamente sensible al incremento en la cantidad de delegados concurrentes, ya que múltiples procesos de verificación simultáneos compiten por los mismos recursos de procesamiento, generando una carga adicional en el sistema. La verificación implica el cálculo y comparación de códigos HMAC, operaciones que, aunque ligeras individualmente, multiplicadas por un alto número de procesos concurrentes generan tiempos altos. En contraste, el tiempo de verificación para las 32 consultas iterativas del escenario (i) es significativamente menor (0.266 ms), ya que al procesarse secuencialmente se evita la sobrecarga de sincronización y competencia de recursos.

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



Adicional a las conclusiones observadas en los gráficos anteriores, al comparar el cifrado simétrico y asimétrico, se evidencia que el cifrado asimétrico presenta mayores tiempos de procesamiento que el cifrado simétrico en la mayoría de los escenarios, especialmente a medida que aumenta el número de delegados en el escenario (ii). Esto se debe a que el cifrado simétrico utiliza una única clave para cifrar y descifrar los datos, lo que permite realizar operaciones matemáticas más simples y rápidas, resultando en un menor consumo de recursos computacionales y manteniendo el tiempo de procesamiento bajo y relativamente estable incluso bajo carga concurrente. En contraste, el cifrado asimétrico, aunque utiliza únicamente la clave pública para cifrar, implica operaciones matemáticas mucho más complejas lo cual incrementa considerablemente la carga computacional y, por ende, el tiempo de procesamiento. Sin embargo, en el escenario de 32 consultas iterativas (escenario i), ambos tipos de cifrado muestran tiempos bajos y relativamente similares. Esto sugiere que, en condiciones de baja carga computacional, cuando el sistema puede dedicar todos sus recursos a una operación a la vez sin competencia simultánea, la diferencia de desempeño entre el cifrado simétrico y el asimétrico se reduce significativamente, permitiendo tiempos de respuesta eficientes en ambos casos.

Conclusión general: De manera general, al analizar los cuatro gráficos presentados, se concluye que el tiempo de cifrado y verificación de datos aumenta proporcionalmente con el número de delegados involucrados. Finalmente, se puede observar que, en todas las gráficas, el escenario (i) de consultas iterativas es aquel que obtiene los mejores tiempos en todas las categorías dadas, indicando que las operaciones son más eficientes con un posible menor nivel de delegación o menor complejidad.

6. 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 estimar el rendimiento de nuestro procesador, partimos de los siguientes datos:

Procesador: Intel Core i7-1255U (12^a generación)

- 10 núcleos físicos (2 de alto rendimiento + 8 de eficiencia energética).
- 12 hilos (procesadores lógicos).

Tamaño promedio de consulta cifrada (tabla de servicios): aproximadamente 1 KB (1024 bytes).

Además, encontramos la información sobre la velocidad del procesador. Esto se hizo al poner en el terminal el siguiente comando:

```
PS C:\Users\Sofia Toro> Get-CimInstance -ClassName Win32_Processor | Select-Object Name, MaxClockSpeed
```

Name	MaxClockSpeed
12th Gen Intel(R) Core(TM) i7-1255U	1700

Con esto podemos concluir que la velocidad máxima registrada es 1700 MHz, es decir, 1.7 GHz.

Con esta información, calculamos el número aproximado de operaciones de cifrado que el procesador puede realizar por segundo, distinguiendo entre cifrado simétrico (AES) y cifrado asimétrico (RSA).

Para estimar la velocidad de procesamiento de la máquina de manera precisa, seleccionamos el promedio del escenario de 32 consultas iterativas. Esta elección se debe a que en un entorno iterativo las consultas se procesan secuencialmente, sin competencia simultánea por los recursos del procesador. De esta manera, los tiempos medidos reflejan de forma más cierta la capacidad real del CPU para realizar operaciones de cifrado, sin interferencias derivadas de la ejecución paralela de múltiples hilos.

Basándonos en el escenario de 32 consultas iterativas, obtuvimos los siguientes tiempos promedio:

- **Tiempo promedio para cifrado simétrico (AES):** 0.2552 milisegundos por operación.
- **Tiempo promedio para cifrado asimétrico (RSA 1024 bits):** 0.4271 milisegundos por operación.

Como sabemos cuánto tarda en promedio una operación, podemos calcular cuántas operaciones caben en 1 segundo (**1000 ms**).

$$\text{Operaciones por segundo} = \frac{1000\text{ms/segundo}}{\text{Tiempo por operacion (ms)}}$$

Para el cifrado simétrico (AES):

$$\text{Operaciones por segundo} = \frac{1000\text{ms/segundo}}{0.2552 \text{ (ms)}} \approx 3919 \text{ operaciones por segundo}$$

Para el cifrado asimétrico:

$$\text{Operaciones por segundo} = \frac{1000\text{ms/segundo}}{0.4271 \text{ (ms)}} \approx 2342 \text{ operaciones por segundo}$$

A partir de las mediciones realizadas bajo un escenario de ejecución iterativa, se estimó la capacidad de procesamiento de la máquina en operaciones de cifrado. Se encontró que el procesador Intel Core i7-1255U puede realizar aproximadamente 3919 operaciones de cifrado simétrico (AES) y 2342 operaciones de cifrado asimétrico (RSA 1024 bits) por segundo. Estos resultados reflejan que las operaciones son casi el doble de cifrados simétricos por segundo comparado con el asimétrico.

Referencias Bibliográficas:

- Dotnet-Bot. (n.d.). *Algorithmparametergenerator Clase (Java.security)*. (Java.Security) | Microsoft Learn. <https://learn.microsoft.com/es-es/dotnet/api/java.security.algorithmparametergenerator?view=net-android-34.0>
- Hernandez, J. (2024, January 27). *Cifrado simétrico y asimétrico: guía completa sobre criptografía*. Prey Project. <https://preyproject.com/es/blog/tipos-de-cifrado-simetrico-o-asimetrico-rsa-o-aes>