



Caso 3

Integrantes (Grupo x, Sección 4)

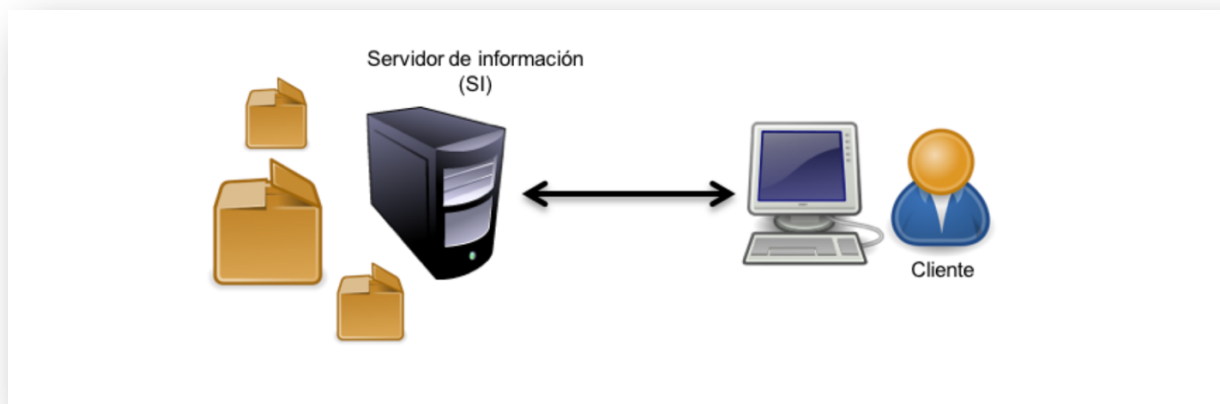
- Juan José Díaz Ortega - 202220657
- Paul Paffens - 202222496

1. Objetivos

- Comprender ventajas y limitaciones de los algoritmos para cifrar datos.
- Construir un prototipo a escala de una herramienta para soportar confidencialidad e integridad.

2. Problemática

Supondremos que una compañía de transportes ofrece a sus clientes el servicio de recogida y entrega de paquetes a domicilio. La compañía cuenta con un servidor para soportar la operación de manejo de paquetes y las consultas de los usuarios. Para el manejo de paquetes y unidades de distribución, tanto los puntos de atención al cliente como las unidades de distribución se comunican periódicamente con el servidor para informar su estado. El servidor almacena la información y atiende consultas relacionadas con estos datos vía internet.



Para este caso nos concentraremos en el proceso de atención a las consultas de los usuarios. Su tarea consiste en construir los programas servidor y cliente que reciben y responden las solicitudes de los clientes.

Servidor:

- Es un programa que guarda un identificador de cliente, un identificador de paquete y el estado de cada paquete recogido (los estados posibles son: ENOFICINA, RECOGIDO, ENCLASIFICACION, DESPACHADO, ENENTREGA, ENTREGADO, DESCONOCIDO) y responde las consultas de los clientes.
- Los estados deben representarse como constantes numéricas, excepto al presentarse mensajes a un usuario, allí deben presentarse en formato alfabético.
- Para simplificar el problema, tendremos un servidor con una tabla predefinida con: login de usuario, identificador de paquete y estado de 32 paquetes. Para consultar el estado de un paquete, un cliente envía al servidor su identificador y el identificador de un paquete, el servidor recibe los datos, consulta la información guardada y responde con el estado correspondiente. Si el identificador de usuario y paquete no corresponden a una entrada en la tabla, el servidor retorna el estado DESCONOCIDO.
- Es un servidor concurrente. El servidor principal crea los delegados por conexión al recibir cada cliente.

Cliente:

- Es un programa que envía una consulta al servidor, espera la respuesta y al recibirla la valida. Si la respuesta pasa el chequeo entonces despliega la repuesta en pantalla, si no pasa el chequeo entonces despliega el mensaje "Error en la consulta".

- Se sugiere manejarlo de forma concurrente.

Tanto servidor como cliente deben cumplir con las siguientes condiciones:

- Las comunicaciones entre cliente-servidor deben usar sockets y seguir el protocolo indicado.
- No use librerías especiales, solo las librerías estándar.
- Desarrolle en Java (java.security y javax.crypto).

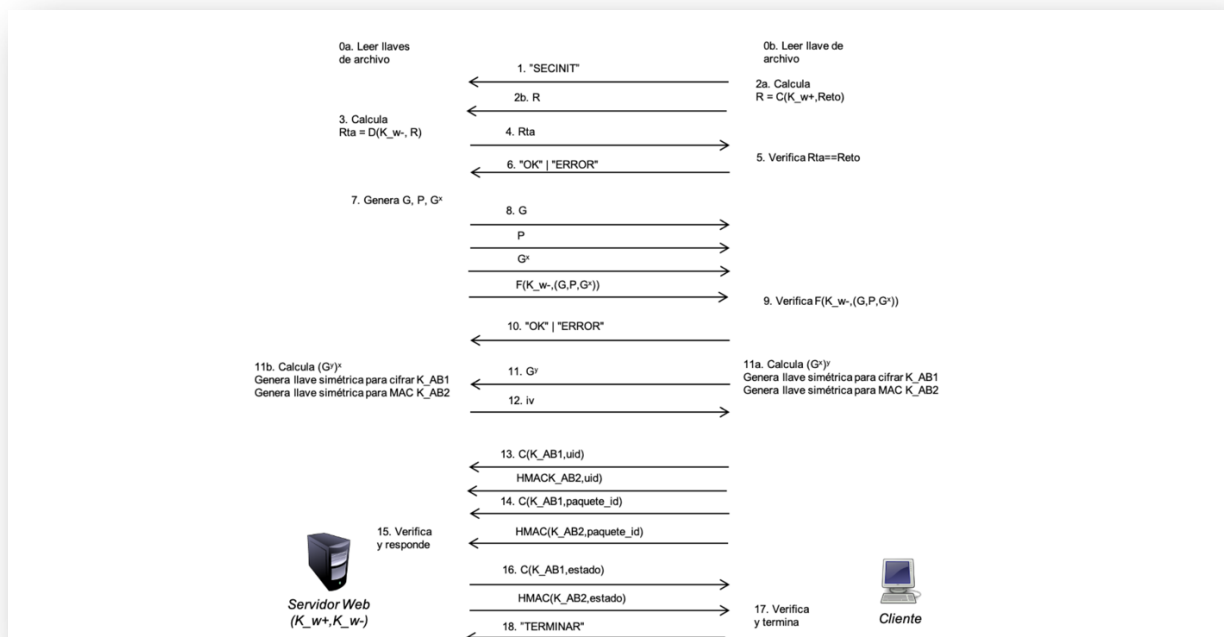


Ilustración 1

Opciones:

- En el código del servidor incluya un menú que tenga dos opciones:
 - o Opción 1: Generar la pareja de llaves asimétricas del servidor y almacenarlas en dos archivos. o Tenga en cuenta:
 - § En una instalación real los permisos asignados a estos archivos serían fundamentales; la llave pública puede ser leída por cualquiera, pero la llave privada solo debería estar al alcance del propietario.
 - § En este prototipo deberá copiar el archivo de la llave pública a un directorio donde los clientes la puedan leer.
 - o Opción 2: Ejecutar creando los delegados, cada delegado sigue el protocolo descrito en la figura 1.

3. Algoritmos usados

Algoritmos que usaremos:

- o AES. Modo CBC, esquema de relleno PKCS5, llave de 256 bits.
- o RSA. Llave de 1024.
- o El vector de inicialización para cifrado simétrico (iv) debe tener 16 bytes generados aleatoriamente o Firma: SHA1withRSA
- o HMACSHA384.

4. Llaves Simetricas

Para generar las llaves simétricas:

1. Primero calcule una llave maestra por medio de Diffie-Hellman.

```

opnsnl dparam -tst 1024
Generating DH parameters, 1024 bit long safe prime
.....+.
.....+..+.....+..+.....
.....+.....
+.....+.....
.....+.....+.....
.....+.....+.....
+++++++
*****
*
DH Parameters: (1024 bit)
P:
    00:98:e6:0e:1f:70:7f:c8:f7:b3:7f:8e:a5:ce:e0:
    b3:7d:5b:93:66:4d:19:e3:1b:71:65:ef:3c:8a:8c:
    ef:45:ac:de:cb:40:16:aa:0f:96:0f:fa:2e:b0:f6:
    a9:3d:ef:57:aa:cd:9a:23:62:bb:37:d0:75:ad:85:
    20:18:d3:71:ef:a2:60:06:05:fe:46:59:61:c7:d7:
    90:f1:19:85:ec:9f:57:2c:f0:8b:e4:2b:e7:60:3f:
    f5:07:0d:26:12:b4:d5:68:20:b1:c7:a0:22:ab:96:
    a9:ee:9a:a5:70:61:72:5c:02:f6:10:da:fe:95:45:
    ba:b3:ec:92:4b:72:d1:bc:a7
G:      2 (0x2)
recommended-private-length: 175 bits
-----BEGIN DH PARAMETERS-----
MIGLAoGBAJmDh9wf8j3s3+0pc7gs31bk2ZNGeMbcWxvPIqM70Ws3stAFgoPlg/6
LrDzqT3vV6rNmIuiuzfQda2FIBJTce+iYAYF/kZZYcfXkPEZheyfVyzwi+Qr52A/
9QCnJhk0lWggscegIquWqe6apXBhcLwC9hDa/pVFurPskktty0bynAgECAgIArw=
-----END DH PARAMETERS-----

```

P = 00:98:e6:0e:1f:70:7f:c8:f7:b3:7f:8e:a5:ce:e0:
b3:7d:5b:93:66:4d:19:e3:1b:71:65:ef:3c:8a:8c:
ef:45:ac:de:cb:40:16:aa:0f:96:0f:fa:2e:b0:f6:
a9:3d:ef:57:aa:cd:9a:23:62:bb:37:d0:75:ad:85:
20:18:d3:71:ef:a2:60:06:05:fe:46:59:61:c7:d7:
90:f1:19:85:ec:9f:57:2c:f0:8b:e4:2b:e7:60:3f:
f5:07:0d:26:12:b4:d5:68:20:b1:c7:a0:22:ab:96:
a9:ee:9a:a5:70:61:72:5c:02:f6:10:da:fe:95:45:
ba:b3:ec:92:4b:72:d1:bc:a7

G = 2 (0x2)

2. Para manejar números grandes use la clase BigInteger de Java. Los números primos deben tener 1024 bits de longitud.

```
BigInteger sharedSecret = new BigInteger();
```

Esto lo hicimos con el respectivo valor que calculamos arriba.

3. Luego use la llave maestra para calcular un digest con SHA-512

Esto lo hicimos en una clase estática dentro del servidor llamada **DiffieHellman**

4. Parta el digest en dos mitades para generar la llave para cifrado y la llave para código de autenticación: usar los primeros 256 bits se deben usar para construir la llave para cifrar, y los últimos 256 bits para construir la llave para generar el código HMAC

```
byte[] encryptionKey = new byte[32];  
System.arraycopy(digest, 0, encryptionKey, 0, 32);  
byte[] hmacKey = new byte[32];  
System.arraycopy(digest, 32, hmacKey, 0, 32);
```

Esto lo hicimos dentro del metodo handleClient() en el servidor y en el sendRequest() en el lado del cliente

5. Organización en el zip

En el archivo zip se encuentran los dos códigos junto con las llaves generadas, un csv donde ponemos los datos para hacer las gráficas y una clase de test la cual usamos para correr los diferentes escenarios, también esta este documento.

6. Instrucciones de uso

1. Correr el servidor y generar las llaves RSA

```
Seleccione una opción:  
1. Generar llaves RSA  
2. Iniciar el servidor  
1  
Llaves RSA generadas y guardadas.
```

2. Volver a correr el servidor pero esta vez iniciarlo, seleccionar el modo que quiere

```
Seleccione una opción:  
1. Generar llaves RSA  
2. Iniciar el servidor  
2  
Llaves RSA leídas desde archivos.  
Seleccione el modo de operación:  
1. Iterativo  
2. Concurrente  
1  
Servidor iterativo iniciado en el puerto 12345
```

3. Correr el cliente y seleccionar el modo de operación.

```
Seleccione el modo de operación:  
1. Iterativo  
2. Concurrente  
1
```

4. Mirar los resultados del lado de cliente

```
Estado del paquete: ENCLASIFICACION  
Estado del paquete: ENENTREGA  
Estado del paquete: ENTREGADO  
Estado del paquete: ENENTREGA  
Estado del paquete: ENTREGADO  
Estado del paquete: ENTREGADO  
Estado del paquete: ENCLASIFICACION  
Estado del paquete: ENCLASIFICACION  
Estado del paquete: ENTREGADO
```

- Dejar de correr todo y volver a correr el servidor (porque al volverlo a correr ya pinte los resultados que buscamos del lado del servidor)

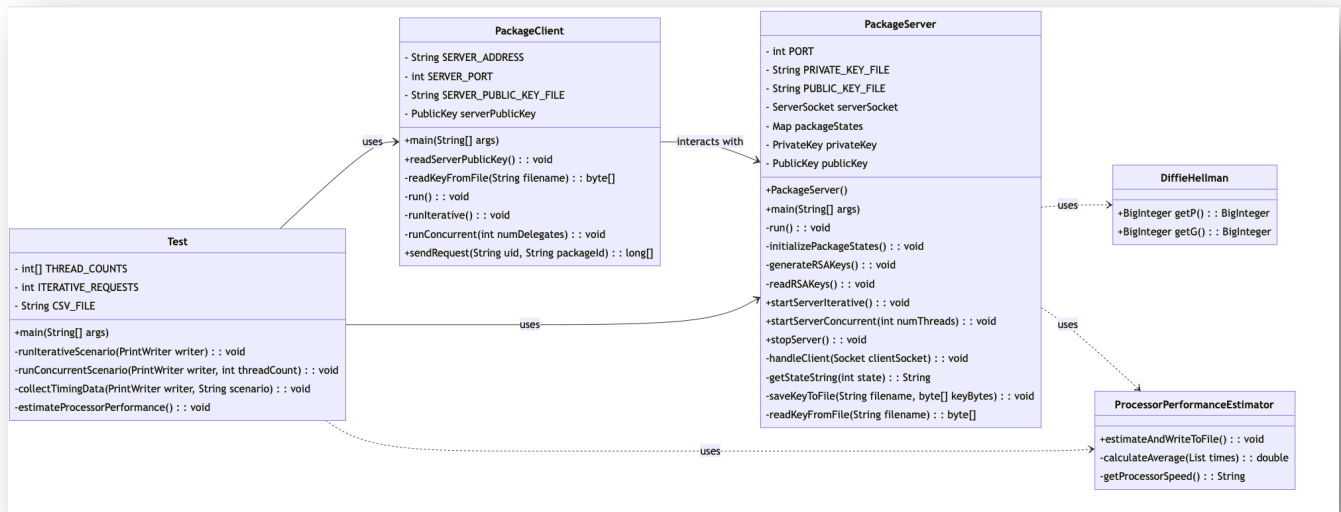
```

Servidor iterativo iniciado en el puerto 12345
Tiempo para descifrar el reto: 44899738 ns
Tiempo para generar G, P, G^x: 4418385 ns
Tiempo para verificar la consulta: 4207821 ns
Tiempo para cifrar el estado con cifrado asimétrico: 366669 ns
Tiempo para cifrar el estado con cifrado simétrico: 20900 ns
Tiempo para descifrar el reto: 1573324 ns
Tiempo para generar G, P, G^x: 5475654 ns
Tiempo para verificar la consulta: 17671 ns
Tiempo para cifrar el estado con cifrado asimétrico: 399918 ns
Tiempo para cifrar el estado con cifrado simétrico: 24288 ns
Tiempo para descifrar el reto: 2394807 ns
Tiempo para generar G, P, G^x: 5385604 ns
Tiempo para verificar la consulta: 22299 ns
Tiempo para cifrar el estado con cifrado asimétrico: 410324 ns
Tiempo para cifrar el estado con cifrado simétrico: 21337 ns

```

- Ahora que vio cómo funciona individualmente, pruebe a usar la clase Test para que sea más rápido y vea como se mira en el CSV y las gráficas generadas.

7. Diseño (Diagrama de clases)



8. Corra su programa en diferentes escenarios y mida el tiempo que el servidor requiere para hacer lo siguiente:

1. Responder el reto
2. Generar G, P y Gx
3. Verificar la consulta

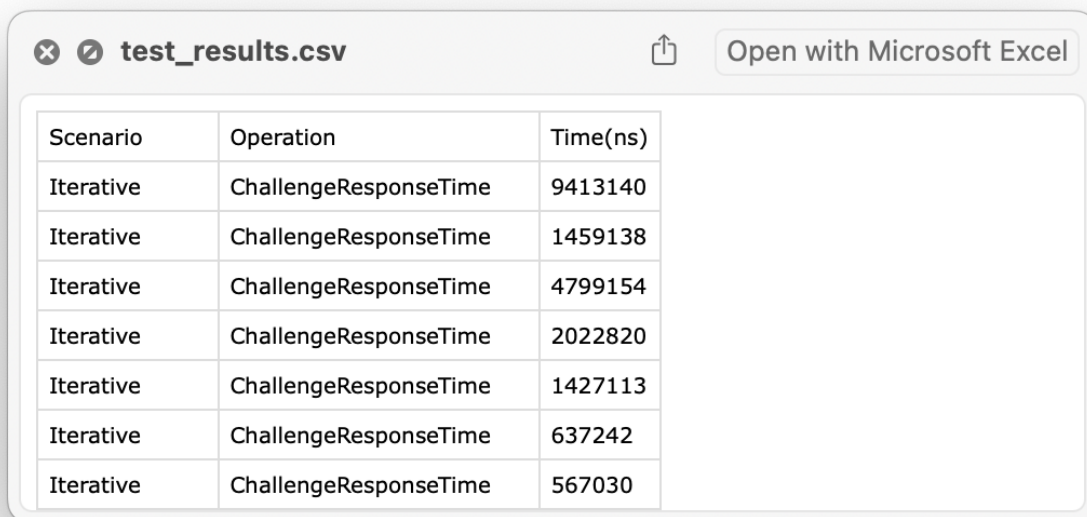
Los escenarios son:

- (i) Un servidor iterativo y un cliente iterativo. El cliente genera 32 consultas.
- (ii) Un servidor y un cliente que implementen delegados. El número de delegados, tanto servidores como clientes, debe variar entre 4, 8, y 32 delegados concurrentes. Cada cliente genera una sola solicitud

Para correr los escenarios disenamos la clase Test.java en la que se corren todos a la vez.

9. Construya una tabla con los datos recopilados.

Esto lo hicimos en un archivo test en el que generamos los resultados en CSV.



The image shows a screenshot of a CSV file named 'test_results.csv' with a table containing performance data. The table has three columns: 'Scenario', 'Operation', and 'Time(ns)'. There are eight rows of data, all for the 'Iterative' scenario, showing 'ChallengeResponseTime' values in nanoseconds.

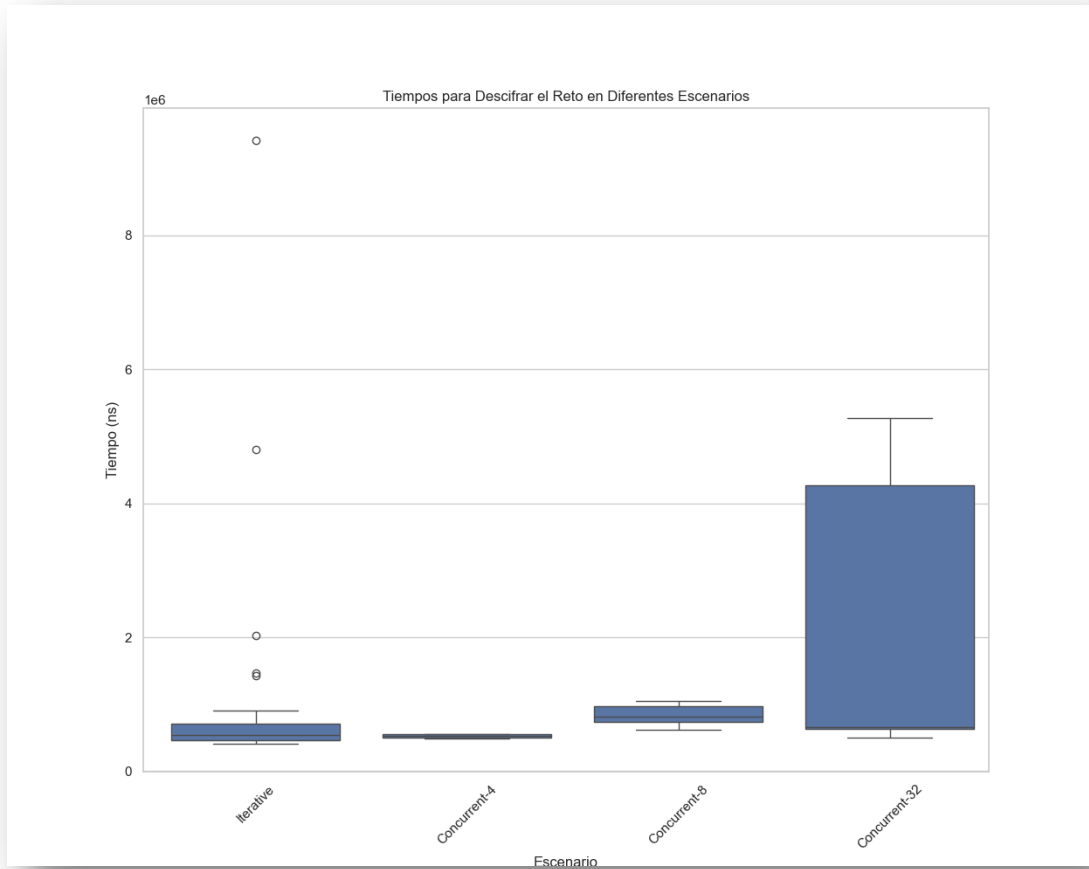
Scenario	Operation	Time(ns)
Iterative	ChallengeResponseTime	9413140
Iterative	ChallengeResponseTime	1459138
Iterative	ChallengeResponseTime	4799154
Iterative	ChallengeResponseTime	2022820
Iterative	ChallengeResponseTime	1427113
Iterative	ChallengeResponseTime	637242
Iterative	ChallengeResponseTime	567030

10. Mida el tiempo que el servidor requiere para cifrar el estado del paquete con cifrado simétrico y con cifrado asimétrico.

Para esto implementamos dentro de los metodos del servidor la toma del tiempo.

11. Construya las siguientes gráficas:

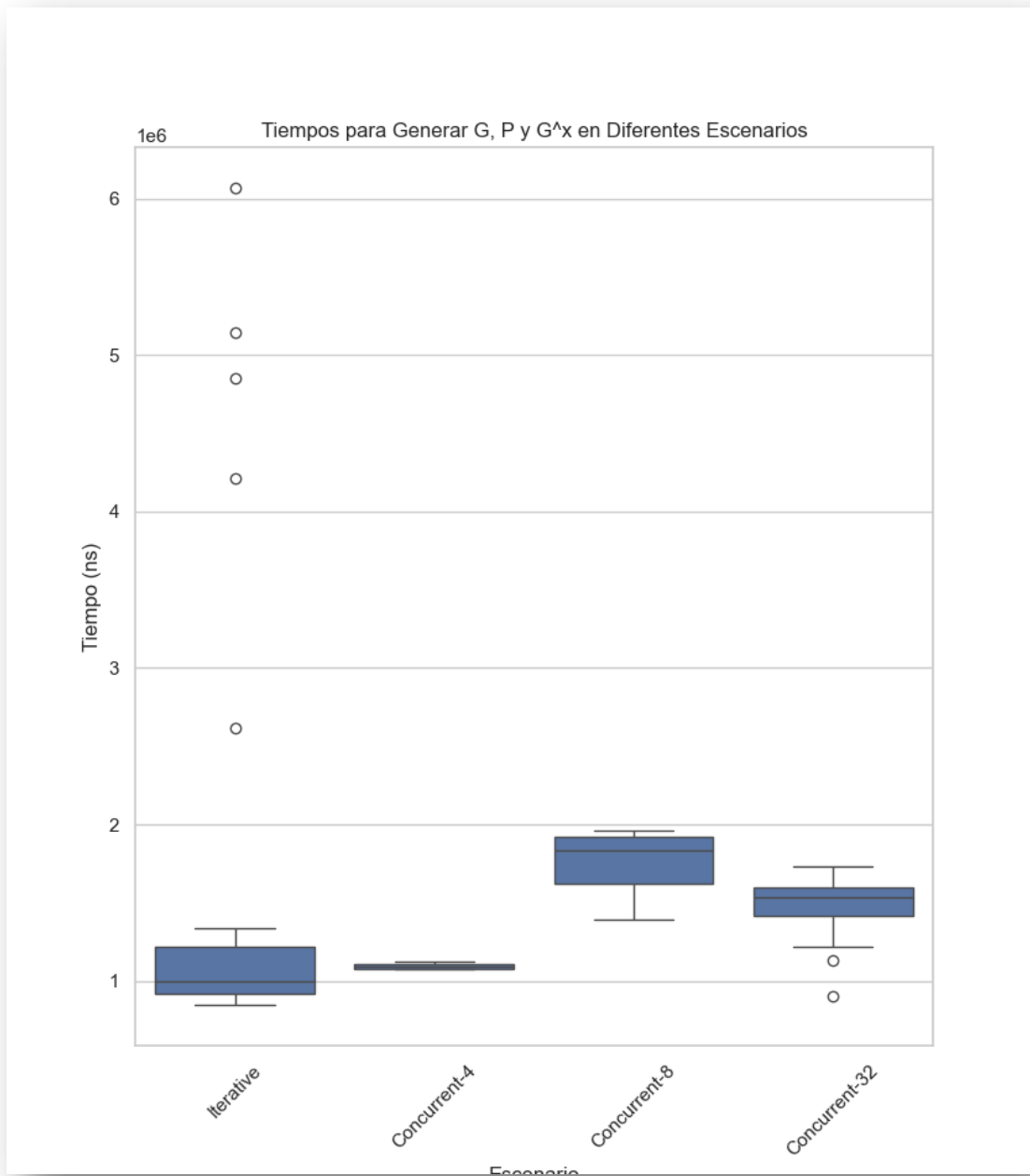
(i) Una que compare los tiempos para descifrar el reto en los diferentes escenarios



En la gráfica, se observa que los tiempos para descifrar el reto en el escenario "Iterative" son, en general, bastante bajos en comparación con "Concurrent-32", que muestra un rango de valores más amplio y algunos valores extremos.

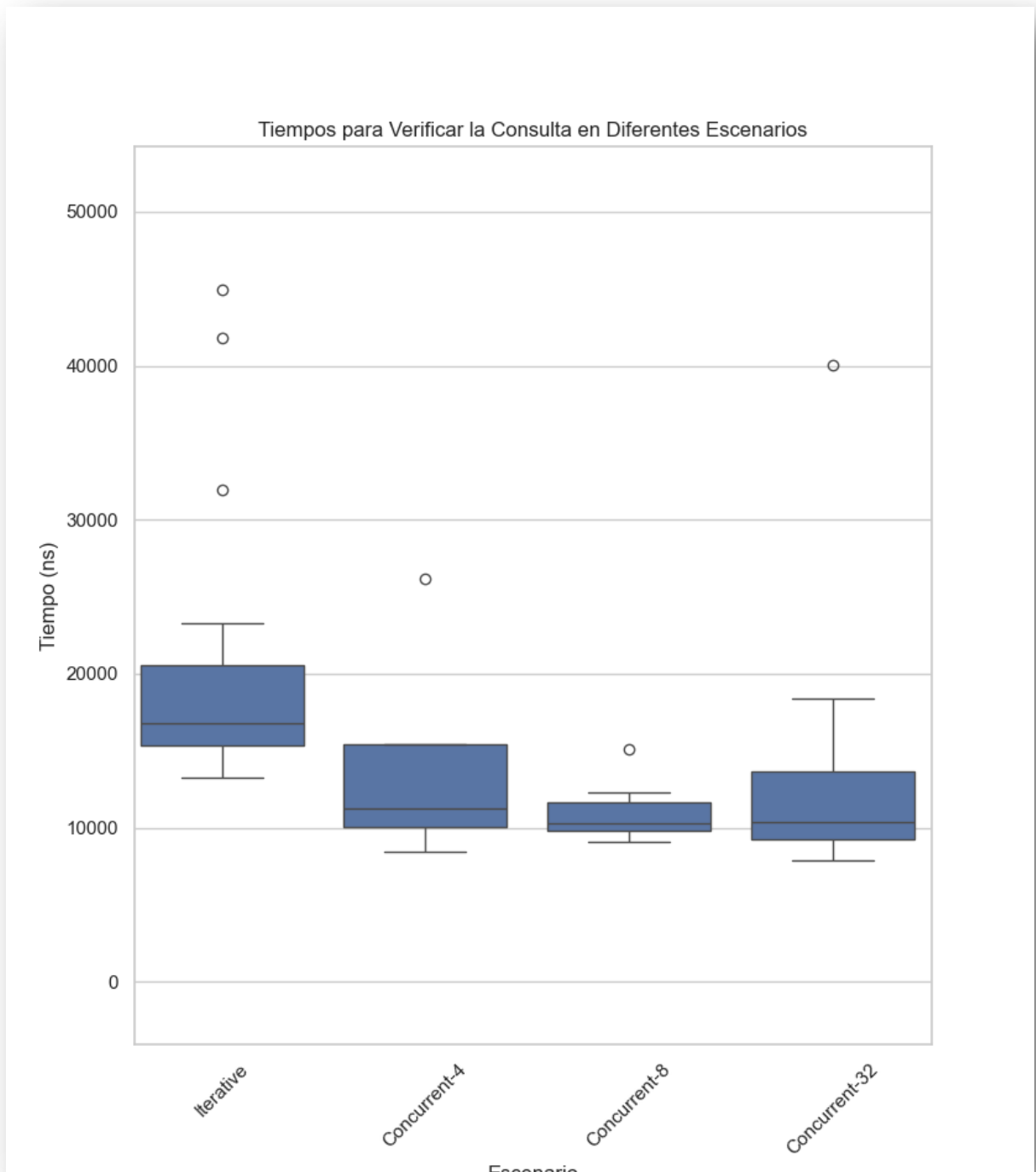
Los tiempos en los escenarios "Concurrent-4" y "Concurrent-8" son consistentes y mucho menores que en "Concurrent-32". Esto sugiere que un mayor nivel de concurrencia puede incrementar los tiempos de procesamiento en algunos casos, probablemente debido a la sobrecarga en la gestión de múltiples hilos en el servidor.

(ii) Una que compare los tiempos para generar G, P y Gx en los diferentes escenarios



La generación de los parámetros G, P y G^x muestra un comportamiento similar al anterior: el escenario "Concurrent-8" tiene tiempos relativamente altos, lo que indica un aumento en la carga cuando se incrementa el nivel de concurrencia. Sin embargo, el escenario "Iterative" tiene tiempos de generación de parámetros un poco menores y consistentes, sugiriendo que este método podría ser más eficiente en cargas más pequeñas sin concurrencia.

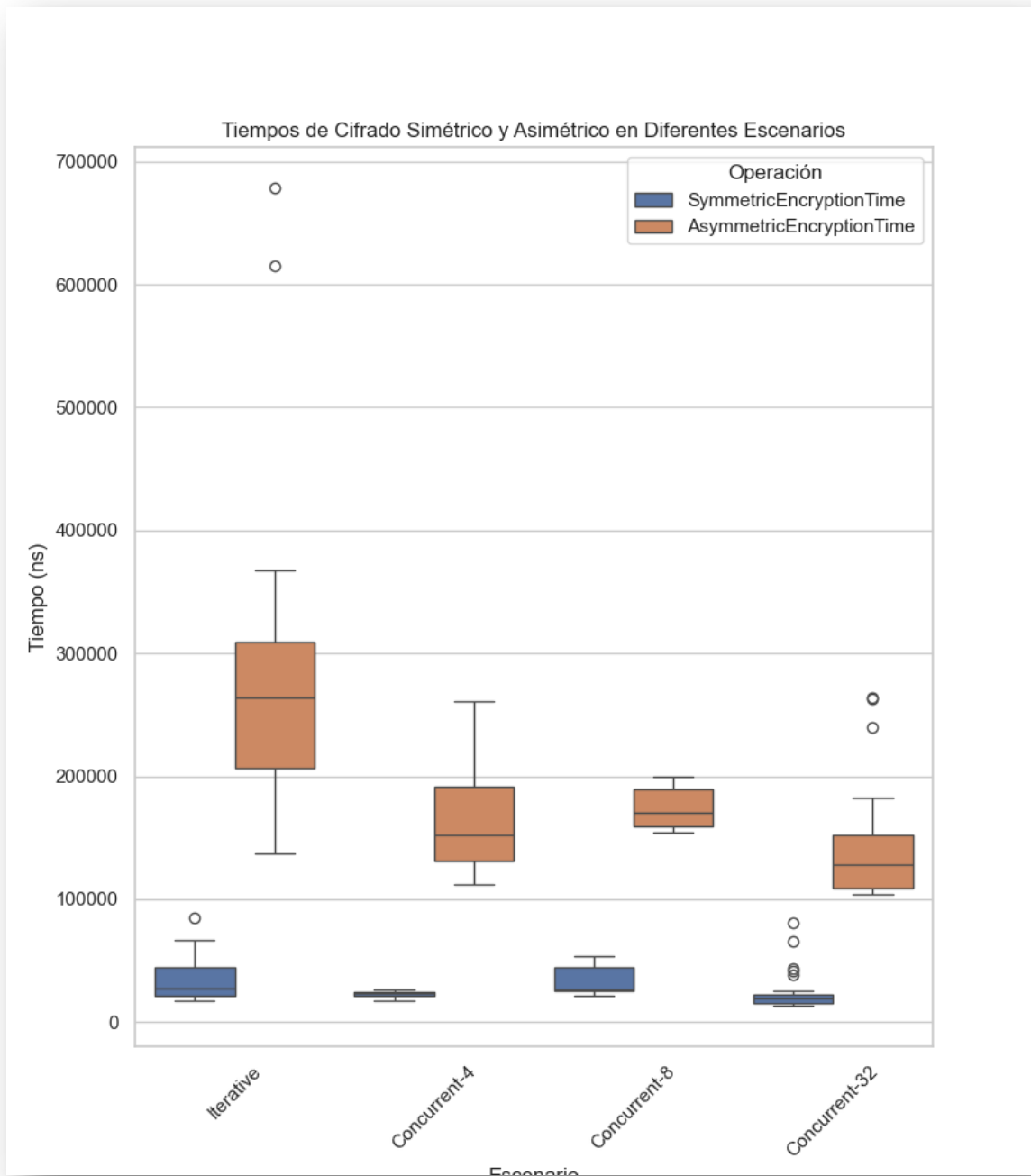
(iii) Una que muestre los tiempos para verificar la consulta en los diferentes escenarios



Los tiempos de verificación son mucho menores que los tiempos de descifrado o generación de parámetros en general, lo cual es esperado debido a la simplicidad de la operación de verificación en comparación con las otras tareas.

Nuevamente, el escenario "Concurrent-8" y "Concurrent-32" muestran más variabilidad, aunque los valores en estos escenarios se mantienen dentro de un rango aceptable, indicando que la concurrencia afecta de manera menos significativa esta operación.

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



La comparación entre el cifrado simétrico y asimétrico muestra que el cifrado asimétrico tiene tiempos considerablemente más altos en todos los escenarios, lo cual es esperado, dado que los algoritmos de cifrado asimétrico generalmente requieren más tiempo computacional. El tiempo para el cifrado simétrico se mantiene relativamente bajo y constante en todos los escenarios concurrentes, lo que indica que esta operación es menos afectada por el nivel de concurrencia y puede manejarse eficientemente incluso con múltiples hilos.

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

Estos resultados sugieren que, si bien el sistema puede manejar la concurrencia, un alto nivel de hilos (como en el escenario "Concurrent-32") tiende a incrementar los tiempos de ciertas operaciones como el descifrado y la generación de parámetros. Además, el cifrado asimétrico es notablemente más costoso en términos de tiempo en comparación con el cifrado simétrico, lo que respalda el uso de cifrado simétrico para operaciones de alta frecuencia en sistemas concurrentes. El resto de observaciones están bajo las propias graficas.

13. Identifique 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 la parte de los cálculos adjuntamos en la carpeta docs un documento hecho en latex en el que explicamos los cálculos, los resultados a estos cálculos están en el archivo processor_performance.txt

Processor Performance Estimation
=====
Processor Speed: 2.5 GHz

Symmetric Encryption Calculations:

Total Symmetric Encryption Time (ns): 29091.70
Average Symmetric Encryption Time (s): 0.000029091697
Symmetric Encryption Operations per Second: 34374.07

Asymmetric Encryption Calculations:

Total Asymmetric Encryption Time (ns): 203725.97
Average Asymmetric Encryption Time (s): 0.000203725974
Asymmetric Encryption Operations per Second: 4908.55

Calculations Explained:

-
1. Calculamos el tiempo promedio en nanosegundos sumando todos los tiempos individuales de las operaciones y dividiéndolo entre la cantidad de operaciones.
 2. Convertimos el tiempo promedio de nanosegundos a segundos dividiendo por 1,000,000,000 para obtener el tiempo en segundos.
 3. Estimamos las operaciones por segundo tomando el inverso del tiempo promedio en segundos ($1 / \text{tiempo_promedio_segundos}$).

Nota: Estos cálculos se basan en los datos proporcionados en el archivo 'test_results.csv'.