

Caso de Estudio 3 – Canales Seguros

Sistema de rastreo de paquetes en una compañía transportadora

Juan Andres Eslava Tovar – 202012035

Daniel Camilo Quimbay - 202313861

1. 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.

1.1 Implementación del Prototipo

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 respuesta en pantalla, si no pasa el chequeo entonces despliega el mensaje "Error en la consulta".
- Se sugiere manejarlo de forma concurrente.

2. Descripción de la Organización de Archivos

Explica la estructura de los archivos en el proyecto, con énfasis en la organización del contenido del archivo .zip.

- **Raíz del Proyecto:**

- logica/ - Contiene las clases de Java, incluyendo Servidor.java, Cliente.java, SecurityUtils.java, etc.
- public/ - Carpeta donde se almacena la llave pública (public.key).
- server/ - Carpeta donde se almacena la llave privada del servidor (private.key).
- docs/ - Carpeta que contiene este informe.
- dh_values.txt - Archivo que contiene los valores preestablecidos de Diffie-Hellman para P y G creados con openssl.

3. Instrucciones para Correr el Servidor y el Cliente

Describe paso a paso cómo ejecutar el sistema, tanto para el servidor como para el cliente.

3.1. Requisitos Previos

- **Java:** Tener Java instalado (Java SE-20).
- **OpenSSL:** Para generar los valores Diffie-Hellman, si se desea cambiar dh_values.txt.

3.2. Configuración de Clientes Concurrentes

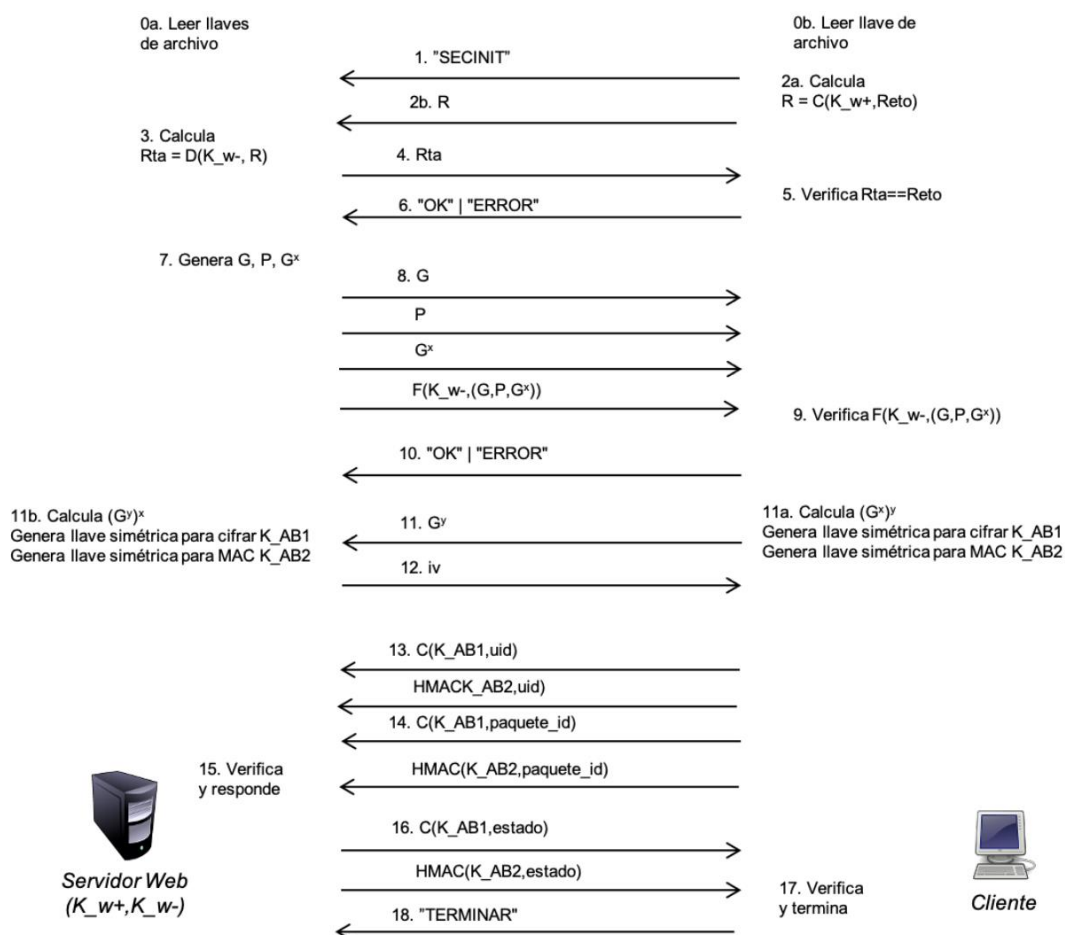
Al correr la consola y elegir la opción 2 se tendrá un input para elegir la cantidad de clientes, si se selecciona 1 va a generar un cliente iterativo con 32 peticiones. En cambio, si se selecciona cualquier otro número se ejecutará esa cantidad de clientes concurrentes.

3.3. Pasos para Ejecutar

- Ejecutar Consola
- Ejecutar Opción 1: Generar llaves (En caso de que no se encuentren generadas en las carpetas: public y server)
- Ejecutar Opción 2: Ejecutar delegados
- Determinar número de clientes (si escoge 1 cliente, hace el caso iterativo con 32 consultas).

4. Descripción de los Pasos del Diagrama

Cada delegado del sistema sigue el siguiente protocolo:



Para entender el funcionamiento del código, se hace la siguiente descripción de los pasos mostrados en el diagrama y su implementación en el código:

0a. Leer Llaves de Archivo (Servidor)

- **Descripción:** El servidor lee las llaves pública y privada necesarias para la comunicación.
- **Implementación en Código:** En Servidor.java, el método readKeysFromFile() carga las llaves desde los archivos public.key y private.key.

0b. Leer Llave de Archivo (Cliente)

- **Descripción:** El cliente lee la llave pública del servidor.
- **Implementación en Código:** En Cliente.java, el método readPublicKeyFromFile() carga la llave pública desde el archivo public.key.

1. "SECINIT" (Inicio de Sesión)

- **Descripción:** El cliente envía el mensaje "SECINIT" para iniciar la sesión.
- **Implementación en Código:** El método enviarInicio() en Cliente.java envía "SECINIT" al servidor.

2a. Calcular Reto (Cliente)

- **Descripción:** El cliente cifra un reto y lo envía al servidor.
- **Implementación en Código:** En Cliente.java, el método enviarReto() cifra el mensaje "Best group of infracomp" y lo envía al servidor.

2b. Reto

- **Descripción:** El cliente envía el reto cifrado al servidor.
- **Implementación en Código:** En Cliente.java, el método enviarReto() envía el reto cifrado.

3. Calcular Respuesta al Reto (Servidor)

- **Descripción:** El servidor descifra el reto recibido.
- **Implementación en Código:** En Servidor.java, el método recibirReto() lee el reto, y responderReto() lo descifra y envía la respuesta.

4. Respuesta al Reto

- **Descripción:** El servidor envía la respuesta al reto.
- **Implementación en Código:** En Servidor.java, el método responderReto() envía la respuesta descifrada al cliente.

5. Verificar Reto (Cliente)

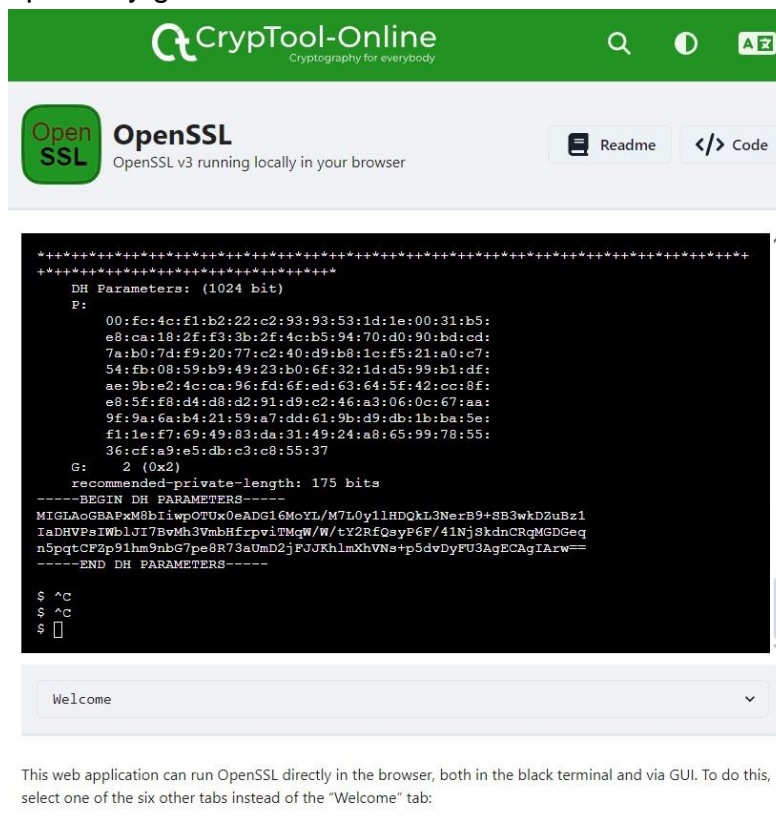
- **Descripción:** El cliente verifica la respuesta del reto.
- **Implementación en Código:** En Cliente.java, el método verificarReto() comprueba si la respuesta es correcta.

6. "OK" o "ERROR" (Confirmación del Cliente)

- **Descripción:** El cliente confirma la validación del reto con "OK" o "ERROR".
- **Implementación en Código:** En Cliente.java, verificarReto() envía "OK" si la verificación es exitosa.

7. Generar G, P, G^x (Servidor)

- **Descripción:** El servidor genera los valores G, P, y G^x usando Diffie-Hellman.
- **Implementación en Código:** En Servidor.java, el método initDiffieHellmanParameters() lee G y P desde dh_values.txt (generado por openssl y genera G^x con un valor secreto x.



8. Enviar G, P, G^x al Cliente (Servidor)

- **Descripción:** El servidor envía G, P y G^x al cliente.
- **Implementación en Código:** En Servidor.java, enviarParametrosDiffieHellman() envía estos valores y la firma al cliente.

9. Verificar Firma (Cliente)

- **Descripción:** El cliente verifica la firma de los parámetros G , P , y Gx .
- **Implementación en Código:** En `Cliente.java`, `recibirParametrosDiffieHellman()` verifica la firma con la llave pública del servidor.

10. "OK" o "ERROR" (Confirmación del Cliente)

- **Descripción:** El cliente confirma la validación de la firma con "OK" o "ERROR".
- **Implementación en Código:** En `Cliente.java`, `recibirParametrosDiffieHellman()` envía "OK" si la firma es válida.

11a. Calcular G^y (Cliente)

- **Descripción:** El cliente genera su clave pública G^y .
- **Implementación en Código:** En `Cliente.java`, `recibirParametrosDiffieHellman()` calcula G^y y lo envía al servidor.

11b. Calcular Secreto Compartido (Servidor)

- **Descripción:** El servidor calcula el secreto compartido G^{xy} .
- **Implementación en Código:** En `Servidor.java`, `calcularLlaveCompartida()` usa G^y recibido del cliente para calcular G^{xy} .

12. Enviar IV (Servidor)

- **Descripción:** El servidor genera y envía un IV para el cifrado AES.
- **Implementación en Código:** En `Servidor.java`, `generarIV()` genera un IV y lo envía al cliente.

13. Solicitud Cifrada (Cliente)

- **Descripción:** El cliente envía una solicitud cifrada con el ID de usuario y el ID de paquete.
- **Implementación en Código:** En `Cliente.java`, `enviarSolicitud()` envía los IDs cifrados y sus HMACs.

14. Verificación de Solicitud (Servidor)

- **Descripción:** El servidor verifica la solicitud del cliente usando los HMACs.
- **Implementación en Código:** En `Servidor.java`, `atenderSolicitud()` verifica los HMACs recibidos.

15. Respuesta Cifrada (Servidor)

- **Descripción:** El servidor envía la respuesta cifrada del estado del paquete.
- **Implementación en Código:** En Servidor.java, atenderSolicitud() envía el estado cifrado y su HMAC.

16. Verificación de Respuesta (Cliente)

- **Descripción:** El cliente verifica la respuesta del servidor usando el HMAC.
- **Implementación en Código:** En Cliente.java, después de recibir la respuesta, se verifica el HMAC del estado.

17. Enviar "TERMINAR" (Cliente)

- **Descripción:** El cliente envía "TERMINAR" para finalizar la conexión.
- **Implementación en Código:** En Cliente.java, al final de run(), se envía "TERMINAR" para indicar la finalización.

5. Escenarios de Prueba y Tiempos Medidos

5.1. Descripción de los Escenarios

- **Escenario 1:** Un servidor y un cliente iterativos. El cliente genera 32 consultas.
- **Escenario 2:** Un servidor y varios clientes con hilos concurrentes. Los clientes se distribuyen en 4, 8 y 32 hilos, cada cliente genera una única solicitud.

5.2. Tiempos Medidos

Para cada uno de los escenarios, se medirán los tiempos que el servidor requiere para:

1. Responder el reto.
2. Generar G , P y G^x .
3. Verificar la consulta del cliente.
4. Tiempo de cifrado simétrico y asimétrico

6. Tabla de Datos recopilados

En esta sección se incluirán las tablas de tiempos promedio medidos en cada escenario.

Tabla 1: Tiempos promedio para responder el reto

Servidor			
Tiempo responder reto			
iterativo	4 hilos	8 hilos	32 hilos
18	26	39	31
16	25	39	56
14	25	39	25
12	25	39	63
16	25	39	59
14	25	39	54
12	25	39	57
12	25	39	57
14	22	27	61
12	22	27	49
12	22	27	62
14	22	27	60
15	27	27	60
15	27	27	56
15	27	27	59
15	27	27	25
14	20	32	60
13	20	32	56
15	20	32	57
16	20	32	64
18	25	32	62
14	25	32	57
15	25	32	30
13	25	32	57
12	24	33	57
14	24	33	58
15	24	33	55
14	24	34	56
14	25	33	60
16	25	32	26
15	25	33	49
17	25	33	59
14,40625	24,15625	32,75	53,03125

Tabla 2: Tiempos promedio para generar G, P y G^x

Servidor

Tiempo generar G, P y G^x			
iterativo	4 hilos	8 hilos	32 hilos
2	1	1	4
1	2	1	2
1	1	1	3
1	1	3	2
2	2	3	2
1	3	2	4
0	3	2	2
1	2	2	4
2	1	2	2
1	1	2	4
2	1	2	4
2	2	3	4
1	1	2	2
1	1	3	2
0	2	2	1
2	2	2	2
1	1	2	1
2	2	1	1
1	1	1	1
1	1	2	1
2	1	2	0
1	1	1	0
1	1	2	1
1	1	1	1
2	1	2	1
1	1	2	1
2	1	2	1
2	2	2	1
1	2	1	1
1	2	1	1
1	1	2	1
2	2	1	3
1,3125	1,46875	1,8125	1,875

Tabla 3: Tiempos promedio para verificar la consulta

Servidor
Tiempo verificar consulta

iterativo	4 hilos	8 hilos	32 hilos
5	2	4	5
1	2	4	9
2	3	4	9
2	3	4	9
2	2	4	5
2	3	2	7
2	3	6	8
2	3	6	10
3	3	5	9
2	3	4	14
0	3	5	14
3	3	5	9
3	4	5	10
2	5	6	9
3	5	6	10
2	5	5	9
0	4	5	10
1	4	5	1
1	4	5	17
1	6	6	21
0	2	8	19
1	5	8	21
2	6	7	21
1	16	8	6
1	4	8	10
2	4	6	11
1	4	7	4
1	5	8	3
1	5	8	10
0	3	8	11
7	4	7	11
1	5	8	10
1,78125	4,15625	5,84375	10,375

7. Comparación de Tiempos de Cifrado (Simétrico vs Asimétrico)

En esta sección se incluirán los tiempos que el servidor requiere para cifrar el estado del paquete usando:

- **Cifrado Simétrico:** Tiempo requerido para cifrar el estado con AES.

- **Cifrado Asimétrico:** Tiempo requerido para cifrar el estado con RSA.

Servidor Tiempo cifrado [ms]	
Asimétrico	Simétrico
1	9
1	0
0	0
0	1
0	0
0	1
1	0
0	0
2	0
0	0
0	0
1	0
0	0
1	0
0	0
0	0
2	0
0	1
1	0
1	0
0	0
0	0
0	0
1	0
0	0
0	0
2	0
0	0
0	0
0	0
0	0
0	1

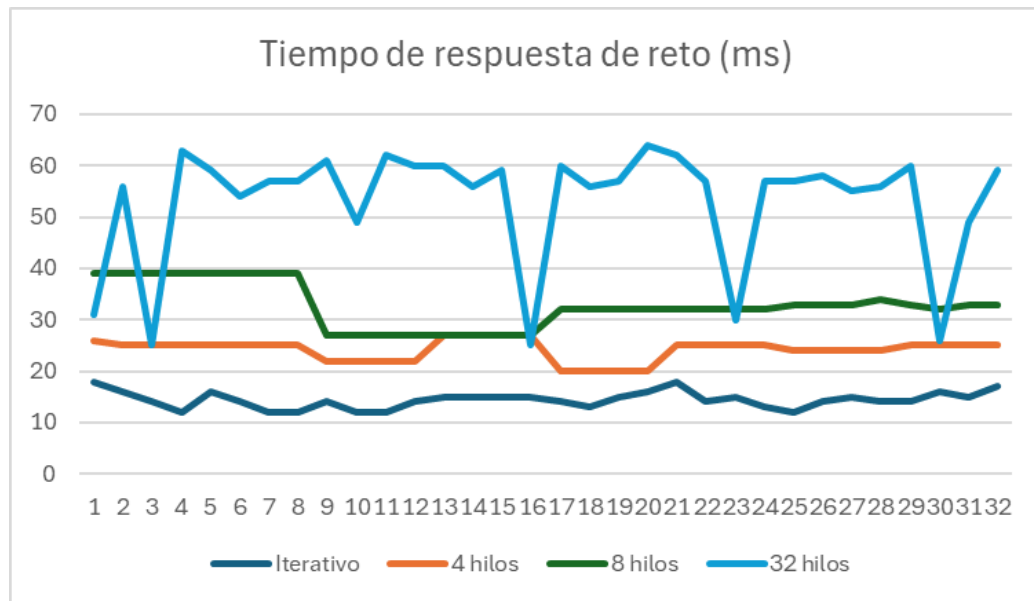
Tabla 4: Comparación de tiempos entre cifrado simétrico y cifrado asimétrico

Tipo de Cifrado	Tiempo Promedio (ms)
Simétrico (AES)	0,40625 ms
Asimétrico (RSA)	0,4375 ms

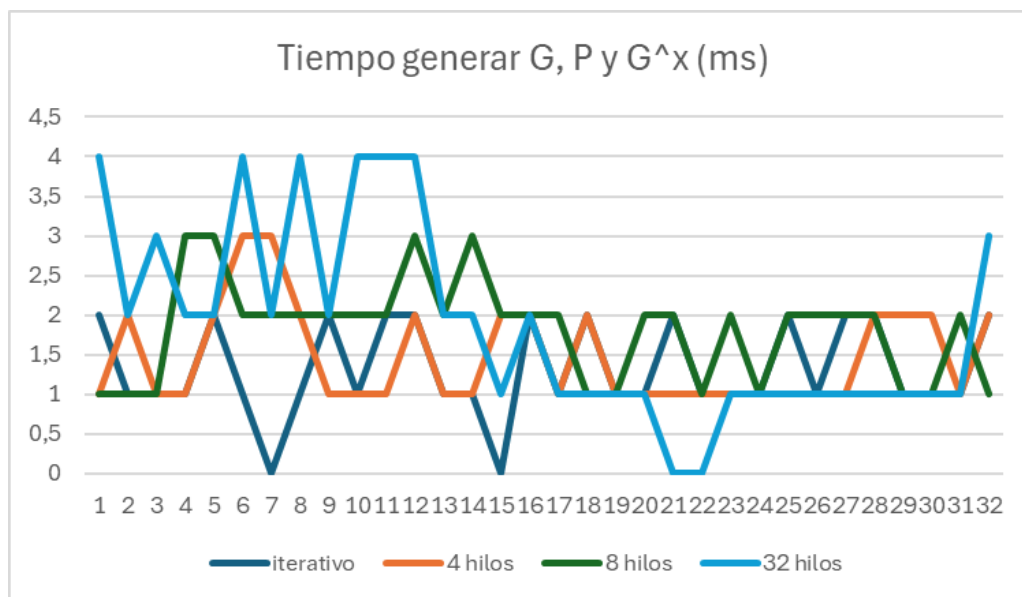
8. Gráficas Comparativas

En esta sección se incluirán las gráficas solicitadas:

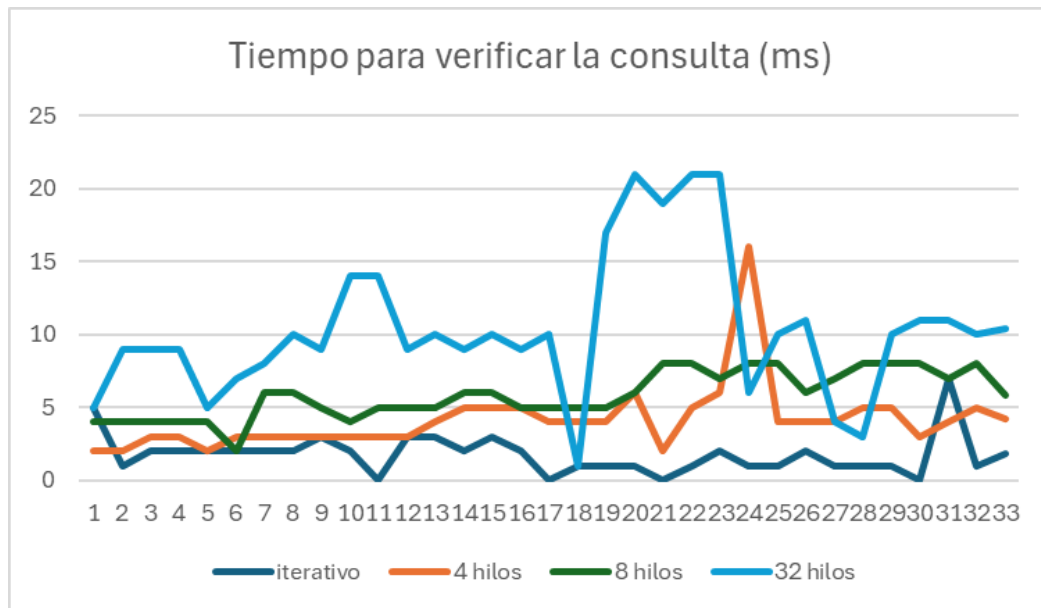
Gráfica 1: Comparación de los tiempos para descifrar el reto en los diferentes escenarios.



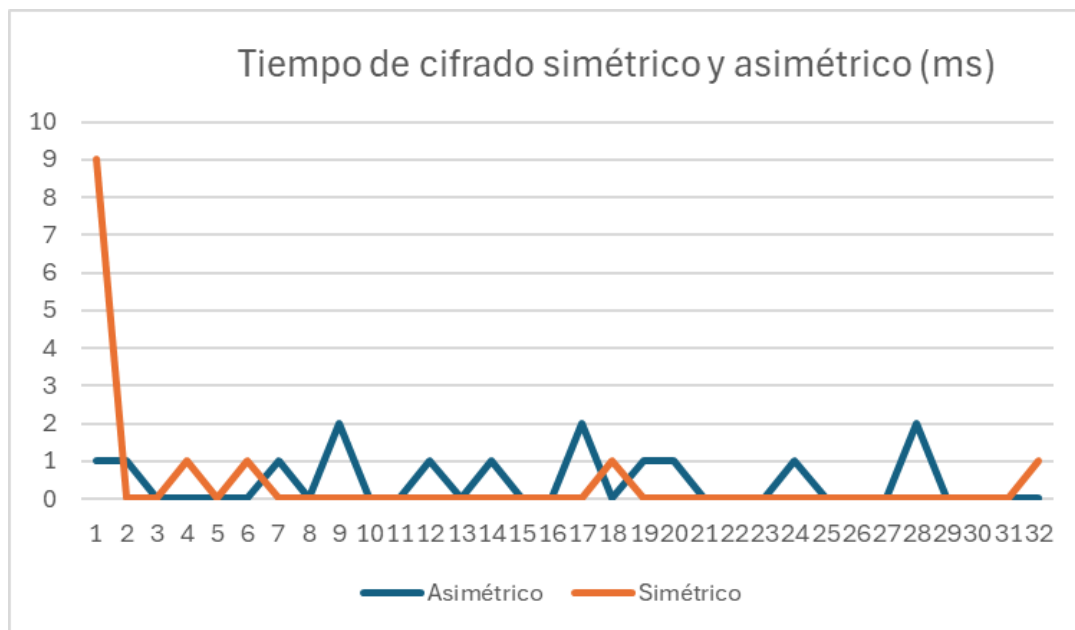
Gráfica 2: Comparación de los tiempos para generar G, P y G^x en los diferentes escenarios.



Gráfica 3: Tiempos para verificar la consulta en los diferentes escenarios.



Gráfica 4: Comparación de tiempos entre cifrado simétrico y cifrado asimétrico en los diferentes escenarios.



9. Análisis de Resultados

9.1 Comentarios sobre las Gráficas

1. Tiempo en descifrar el reto:

Vemos que para descifrar el reto es el más demorado de las 4 gráficas, esto es porque para descifrar el reto se usó cifrado asimétrico el cuál es más demorado que el simétrico. Además, notamos que conforme suben los hilos de ejecución el tiempo de

respuesta también aumenta ya que aumentos la carga sobre el procesador, por lo que tiene menos recursos.

2. Tiempo en generar G, P y G^x

Pudimos ver que, a diferencia de las demás gráficas, en este caso no hubo una gran diferencia entre los 4 casos, pues en este paso G y P estaban definidas en el archivo .txt generado por openssl, por lo que generar "x" es lo único que cambiaba y podía generar una pequeña diferencia, al igual que la generación de G^x , dependiendo del x generado.

3. Tiempo para verificar la consulta

Notemos que como se esperaba, la cantidad de tiempo que se demora en verificar la consulta es mayor conforme aumentan los hilos de ejecución debido a que hay menos recursos para atender las solicitudes de los clientes.

4. Tiempo en de cifrado simétrico vs asimétrico

Aquí podemos ver que el cifrado simétrico es más rápido que el cifrado asimétrico. Los datos de la tabla se vieron afectados también por un dato de 9ms de cifrado asimétrico. Por otro lado, si ignoramos esto también seguiremos viendo que el simétrico es mayor en velocidad.

10. Velocidad del Procesador y Operaciones por Segundo

10.1 Velocidad del Procesador

Procesador: AMD Ryzen 7 6800H with Radeon Graphics 3.20 GHz.

10.2 Cálculo de Operaciones por Segundo

Estimación de cuántas operaciones de cifrado puede realizar el procesador por segundo en:

- **Cifrado Simétrico:**
 - Tiempo promedio por operación: 0.40625 ms
 - Operaciones por segundo = $1000 \text{ ms} / 0.40625 \text{ ms/operación} = 2461.54$ operaciones por segundo
- **Cifrado Asimétrico:** Basado en el tiempo promedio medido para RSA.
 - Tiempo promedio por operación: 0.4375 ms
 - Operaciones por segundo = $1000 \text{ ms} / 0.4375 \text{ ms/operación} = 2285.71$ operaciones por segundo

10.3 Cálculos y Justificación

Referencias:

- *Cryptography and network security*, W. Stallings, Ed. Prentice Hall, 2003.
- *Computer Networks*. Andrew S. Tanenbaum. Cuarta edición. Prentice Hall 2003, Caps 7, 8.
- *Blowfish*. Página oficial es: <http://www.schneier.com/blowfish.html>
- *RSA*. Puede encontrar más información en: <http://www.rsa.com/rsalabs/node.asp?id=2125>
- *CD X509*. Puede encontrar la especificación en: <http://tools.ietf.org/rfc/rfc5280.txt>
- *MD5*. Puede encontrar la especificación en : <http://www.ietf.org/rfc/rfc1321.txt>