

Seguridad canales Seguros Sistema de rastreo de paquetes en una compañía transportadora

Caso de Estudio 03

Santiago Quiroz - 202216453
Johan Camilo Murcio - 202317418

Profesor Magistral

Sandra Rueda



Universidad De Los Andes, Bogotá
Departamento de Ingeniería de Sistemas y Computación

Introducción

Con el fin de generar canales adecuados y seguros para la comunicación se simuló de manera adecuada la comunicación de llaves por medio del protocolo Diffie Hellman con ayuda de comunicación de llaves en medios inseguros. Lo anterior se logró con, java.crypto y java.security librerías que ayudaron al proceso de simulación de comunicación entre cliente-servidor e implementación de seguridad en el mismo. A continuación se presentará la implementación y solución del Caso 3.

Funcionamiento e implementación

Observación: Ejecutar inicialmente el comando “cd src” seguido de “javac *.java” lo anterior se hace para la compilación del proyecto y lograr así su buen funcionamiento.

Inicialmente se empieza por medio de la creación de las llaves públicas y privadas por medio de la clase generadora de llaves, la cual por medio de la librería java.security se encarga de la creación de llaves para el servidor en donde se crean los dos archivos con los cuales se trabaja el resto del caso. El comando asociado para la ejecución de la clase es

“java KeyGenerator”. Si las llaves ya se encuentran de la carpeta, eliminarlas y ejecutar la clase que genera las llaves.



Imagen asociada a la creación de llaves

Luego se procede a encender el servidor por medio de los comandos: “java ServidorPrincipal” en donde empezará cargando las llaves previamente creadas y a escuchar por el puerto 5000 a través un socket creado justamente para la comunicación entre cliente-servidor. Adicionalmente, se inicializan los valores máximos de clientes posibles en donde cada uno de los posibles clientes será un thread (del pool de threads creado inicialmente). En el caso de que un cliente llegue a conectarse al servidor, se procede a llamar al ServidorDelegado que se encargará principalmente de la negociación de la llave Kcs (llave acordada entre cliente-servidor) como se muestra a continuación.

```
System.out.println(x:"1 Comenzando negociación Diffie-Hellman...");

int pLen = in.readInt();
byte[] pBytes = new byte[pLen];
in.readFully(pBytes);
BigInteger p = new BigInteger(pBytes);

int gLen = in.readInt();
byte[] gBytes = new byte[gLen];
in.readFully(gBytes);
BigInteger g = new BigInteger(gBytes);

KeyPair serverDH = DHHelper.generateKeyPair(p, g);

System.out.println(x:"2 Enviando llave pública del servidor...");
byte[] myPubKeyEncoded = serverDH.getPublic().getEncoded();
out.writeInt(myPubKeyEncoded.length);
out.write(myPubKeyEncoded);

System.out.println(x:"3 Recibiendo llave pública del cliente...");
int clientPubLen = in.readInt();
byte[] clientPubKeyEncoded = new byte[clientPubLen];
in.readFully(clientPubKeyEncoded);
```

Imágenes asociadas a la creación de Kcs

```
System.out.println(x:"4) Calculando llave secreta de sesión...");
byte[] sharedSecret = DHHelper.generateSharedSecret(serverDH.getPrivate(), clientPubKey);

MessageDigest sha512 = MessageDigest.getInstance(algorithm:"SHA-512");
byte[] digest = sha512.digest(sharedSecret);

SecretKey aesKey = new SecretKeySpec(Arrays.copyOfRange(digest, from:0, to:32), "AES");
SecretKey hmacKey = new SecretKeySpec(Arrays.copyOfRange(digest, from:32, to:64), "HmacSHA256");

System.out.println(x:"5) Firmando, cifrando y enviando tabla de servicios...");

ByteArrayOutputStream bos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(bos);
oos.writeObject(servicios);
oos.flush();
byte[] tablaBytes = bos.toByteArray();
```

En donde luego de esperar la respuesta del cliente se firma y se cifra la tabla de servicios para que el cliente escoja lo que desea observar. Es importante mencionar que el servidor delegado obtiene las llaves privada y pública del servidor inicial para simular la conexión entre el cliente y el servidor inicial.

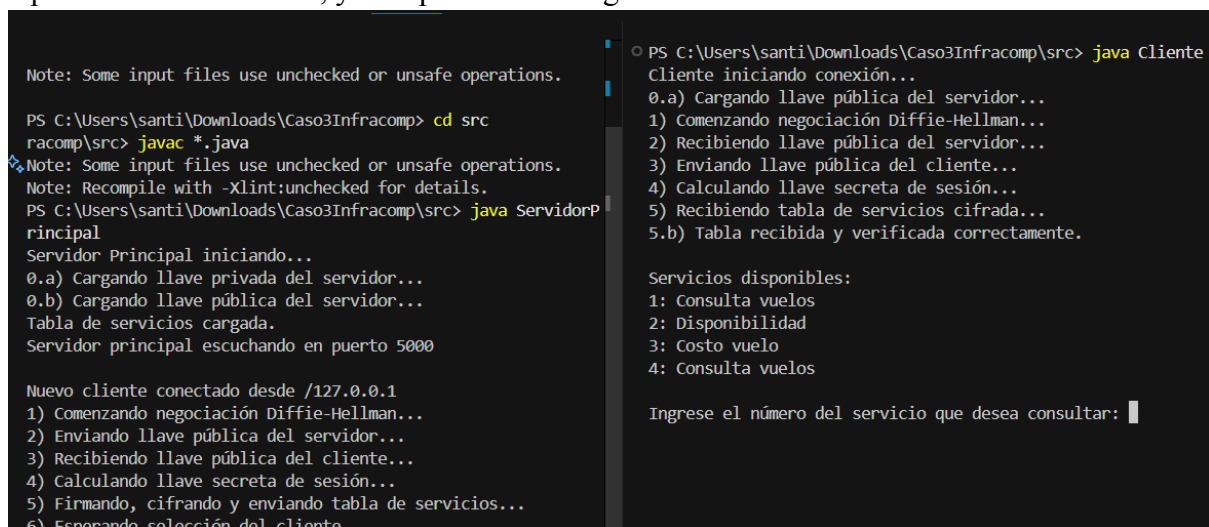
Aclaraciones: * El cliente a la hora de solicitar la conexión se une a través de un socket propio que se le pasará al ServidorDelegado y en donde enviará la información del servicio seleccionado posteriormente.

* El ServidorDelegado se crea gracias a que se busca cumplir con la escalabilidad del servidor y en el caso de que se quiera implementar más servidores facilita el manejo y conexión entre servidor y clientes importante para caso de varios clientes (iterativo y concurrente).

Con lo anterior en mente ahora se procede a explicar la implementación de los clientes en como ya se dijo anteriormente, cada uno de los clientes contiene un propio socket.

Opción un único cliente

El comando asociado para la ejecución de un único cliente es el siguiente (sigue después de los anteriormente solicitados) “java Cliente” el funcionamiento de un único cliente ya se especificó anteriormente, y lo esperado es lo siguiente:



```
Note: Some input files use unchecked or unsafe operations.
PS C:\Users\santi\Downloads\Caso3Infracomp> cd src
racomp\src> javac *.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
PS C:\Users\santi\Downloads\Caso3Infracomp\src> java ServidorP
rincipal
Servidor Principal iniciando...
0.a) Cargando llave privada del servidor...
0.b) Cargando llave pública del servidor...
Tabla de servicios cargada.
Servidor principal escuchando en puerto 5000

Nuevo cliente conectado desde /127.0.0.1
1) Comenzando negociación Diffie-Hellman...
2) Enviando llave pública del servidor...
3) Recibiendo llave pública del cliente...
4) Calculando llave secreta de sesión...
5) Firmando, cifrando y enviando tabla de servicios...
6) Esperando selección del cliente...

PS C:\Users\santi\Downloads\Caso3Infracomp\src> java Cliente
Cliente iniciando conexión...
0.a) Cargando llave pública del servidor...
1) Comenzando negociación Diffie-Hellman...
2) Recibiendo llave pública del servidor...
3) Enviando llave pública del cliente...
4) Calculando llave secreta de sesión...
5) Recibiendo tabla de servicios cifrada...
5.b) Tabla recibida y verificada correctamente.

Servicios disponibles:
1: Consulta vuelos
2: Disponibilidad
3: Costo vuelo
4: Consulta vuelos

Ingrese el número del servicio que desea consultar: 
```

Imagen funcionamiento Cliente

Cómo se logra ver el funcionamiento es normal y a la hora de usar los distintos servicios se logra obtener lo siguiente:

```
Note: Some input files use unchecked or unsafe operations.
PS C:\Users\santi\Downloads\Caso3Infracomp> cd src
racomp\src> javac *.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
PS C:\Users\santi\Downloads\Caso3Infracomp\src> java ServidorPrincipal
Servidor Principal iniciando...
0.a) Cargando llave privada del servidor...
0.b) Cargando llave pública del servidor...
Tabla de servicios cargada.
Servidor principal escuchando en puerto 5000

Nuevo cliente conectado desde /127.0.0.1
1) Comenzando negociación Diffie-Hellman...
2) Enviando llave pública del servidor...
3) Recibiendo llave pública del cliente...
4) Calculando llave secreta de sesión...
5) Firmando, cifrando y enviando tabla de servicios...
6) Esperando selección del cliente...
Tiempo de cálculo de HMAC en servidor (consulta): 646900 nanosegundos
7) Enviando IP y puerto del servicio seleccionado...
Tiempo cifrado simétrico (AES): 804500 nanosegundos
Tiempo cifrado asimétrico (RSA): 6099200 nanosegundos
8) Finalizando conexión con cliente.

0.a) Cargando llave pública del servidor...
1) Comenzando negociación Diffie-Hellman...
2) Recibiendo llave pública del servidor...
3) Enviando llave pública del cliente...
4) Calculando llave secreta de sesión...
5) Recibiendo tabla de servicios cifrada...
5.b) Tabla recibida y verificada correctamente.

Servicios disponibles:
1: Consulta vuelos
2: Disponibilidad
3: Costo vuelo
4: Consulta vuelos

Ingrese el número del servicio que desea consultar: 1
6) Enviando selección de servicio: 1
7) Esperando respuesta del servidor...

Servicio seleccionado:
IP: 127.0.0.1
Puerto: 6001
8) Comunicación finalizada correctamente.
PS C:\Users\santi\Downloads\Caso3Infracomp\src>
```

Los valores obtenidos fueron los siguientes

Opción cliente iterativo

Para la opción de clientes iterativos la ejecución es similar a la opción de único cliente, donde se ejecuta la clase ClaseIterativo.java, esto ejecutará por defecto la cantidad de procesos que indica el for dentro del archivo, por lo que a la hora de tratar de cambiar la cantidad de procesos, cambiar el número dentro del for que se indica a continuación.

Código iterativo:

Para el código iterativo de los clientes se utilizó un ciclo for donde vamos por la cantidad de casos deseados, en la imagen la cantidad de casos es de 32, en cada caso lo que hacemos es generar una consulta random entre las posibilidades y después se realizar todo el proceso necesario por cada consulta de cada cliente.

```
for (int i = 1; i <= 32; i++) {
    int servicioElegido = keys.get(random.nextInt(keys.size()));
    System.out.println("\nConsulta #" + i + ": solicitando servicio ID " + servicioElegido);

    Socket socketConsulta = new Socket(IP_SERVIDOR, PUERTO_SERVIDOR);
    DataInputStream in = new DataInputStream(socketConsulta.getInputStream());
    DataOutputStream out = new DataOutputStream(socketConsulta.getOutputStream());
```

Datos recopilados de los distintos escenarios de pruebas: En caso de que se necesite se pueden visualizar de mejor manera las tablas en el siguiente link

[!\[\]\(aa53ad6fea213b8b2226d3077e30533a_img.jpg\) Gráficas y tablas auxiliares Caso 3](#).

Imagen tabla único cliente:

Prueba único cliente		
#Clientes	Tiempo cifrado Simétrico	Tiempo cifrado Asimétrico
1	804500	6099200
Tiempo cifrado simétrico (AES): 804500 nanosegundos		
Tiempo cifrado asimétrico (RSA): 6099200 nanosegundos		
8) Finalizando conexión con cliente.		

Valores obtenidos en cliente iterativo

Prueba iterativa 32 clientes			
Cliente	Tiempo cálculo HMAC (ns)	Tiempo cifrado simétrico (AES) (ns)	Tiempo cifrado asimétrico (RSA) (ns)
1	333400	647400	3934800
2	344600	765700	1323500
3	347800	572300	702700
4	420100	707300	522600
5	301100	746900	642000
6	228900	430000	559200
7	280600	632600	1411200
8	379400	402400	549400
9	253400	455100	674300
10	285600	475200	513800
11	395300	393700	412400
12	223600	392200	990500
13	312900	486000	604200
14	328200	749800	615400
15	223900	448700	643500
16	206200	586300	534700
17	218700	561800	389200
18	263700	376600	596900

19	203300	309800	339500
20	204800	411900	464800
21	190000	381900	403700
22	149900	352000	513800
23	115600	472900	412400
24	277500	570100	990500
25	194900	303100	604200
26	236500	474600	615400
27	117700	236500	643500
28	159700	167500	534700
29	322000	266500	389200
30	187300	309000	596900
31	79900	167200	339500
32	159700	124700	464800

```

Nuevo cliente conectado desde /127.0.0.1
1) Comenzando negociación Diffie-Hellman...
2) Enviando llave pública del servidor...
3) Recibiendo llave pública del cliente...
4) Calculando llave secreta de sesión...
5) Firmando, cifrando y enviando tabla de servicios...
6) Esperando selección del cliente...
Tiempo de cálculo de HMAC en servidor (consulta): 768900 nanosegundos
7) Enviando IP y puerto del servicio seleccionado...
Tiempo cifrado simétrico (AES): 2007300 nanosegundos
Tiempo cifrado asimétrico (RSA): 3668100 nanosegundos
8) Finalizando conexión con cliente.

Nuevo cliente conectado desde /127.0.0.1
1) Comenzando negociación Diffie-Hellman...
2) Enviando llave pública del servidor...
3) Recibiendo llave pública del cliente...
4) Calculando llave secreta de sesión...
5) Firmando, cifrando y enviando tabla de servicios...
6) Esperando selección del cliente...
Tiempo de cálculo de HMAC en servidor (consulta): 710500 nanosegundos
7) Enviando IP y puerto del servicio seleccionado...
Tiempo cifrado simétrico (AES): 1837200 nanosegundos
Tiempo cifrado asimétrico (RSA): 2456700 nanosegundos
8) Finalizando conexión con cliente.

Nuevo cliente conectado desde /127.0.0.1
1) Comenzando negociación Diffie-Hellman...
Servicio ID: 3, Nombre: Costo vuelo , IP: 127.0.0.1, Puerto: 6001
Servicio ID: 4, Nombre: Consulta vuelos , IP: 127.0.0.1, Puerto: 6001
Servicio ID: 2, Nombre: Disponibilidad , IP: 127.0.0.1, Puerto: 6001
Servicio ID: 3, Nombre: Costo vuelo , IP: 127.0.0.1, Puerto: 6001
Servicio ID: 4, Nombre: Consulta vuelos , IP: 127.0.0.1, Puerto: 6001
[cliente concurrente] Tiempo de cálculo de HMAC de la tabla: 263000 nanosegundos
[Cliente concurrente] HMAC verificado correctamente
Listado de servicios:
Servicio ID: 1, Nombre: Consulta vuelos, IP: 127.0.0.1, Puerto: 6001
Servicio ID: 2, Nombre: Disponibilidad , IP: 127.0.0.1, Puerto: 6001
Servicio ID: 3, Nombre: Costo vuelo , IP: 127.0.0.1, Puerto: 6001
Servicio ID: 4, Nombre: Consulta vuelos , IP: 127.0.0.1, Puerto: 6001
PS C:\Users\santi\Downloads\Caso3Infracomp\src> ^C
PS C:\Users\santi\Downloads\Caso3Infracomp\src> java ClienteIterativo
Cliente Iterativo iniciando...
Servicios recibidos correctamente.

Consulta #1: solicitando servicio ID 3
IP: 127.0.0.1, Puerto: 6001

Consulta #2: solicitando servicio ID 3
IP: 127.0.0.1, Puerto: 6001

Consulta #3: solicitando servicio ID 2
IP: 127.0.0.1, Puerto: 6001

Consulta #4: solicitando servicio ID 1

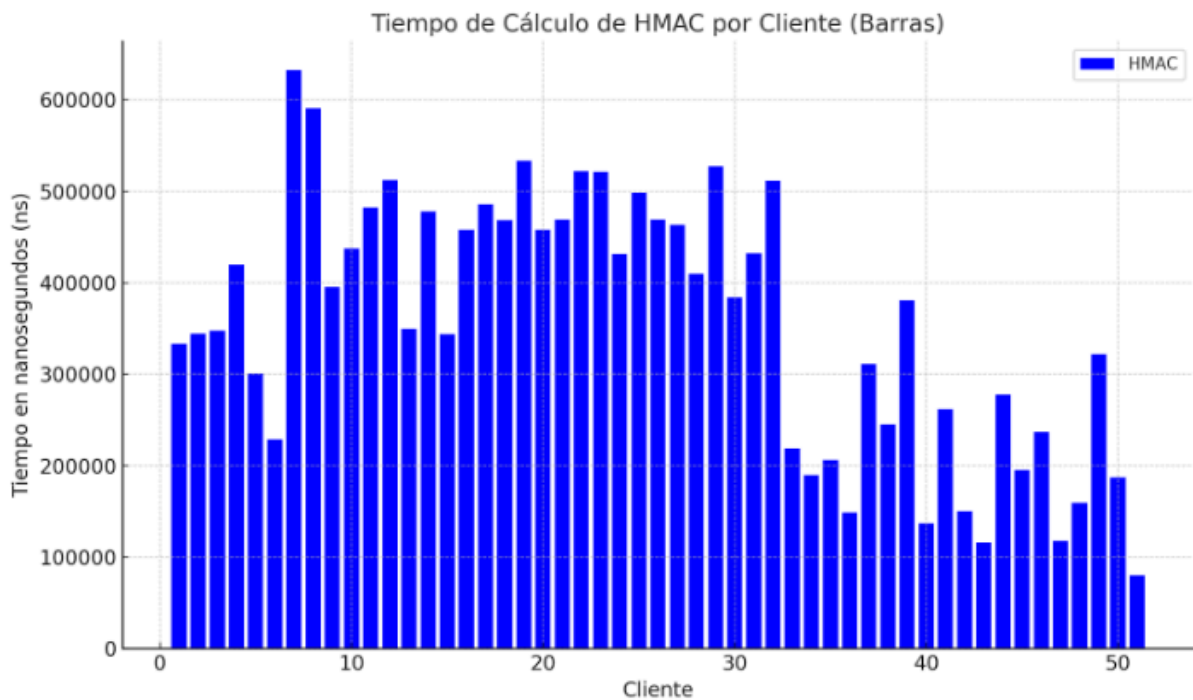
```

Los datos son congruentes con los teóricos ya que como se puede evidenciar en las gráficas adjuntas los valores tienden a la teoría (valores de cifrado simétrico < valores de cifrado asimétrico). Lo cual influye bastante en el rendimiento de la respuesta del servidor en el caso de calcular grandes volúmenes de datos pues como en la gráfica se muestra se ve afectado debido a factores como el número de clientes. Adicionalmente, se dice que nuestros datos son congruentes con los teóricos siguiendo valores similares a lo largo de la ejecución (reafirmando la teoría ya que es un cliente a uno).

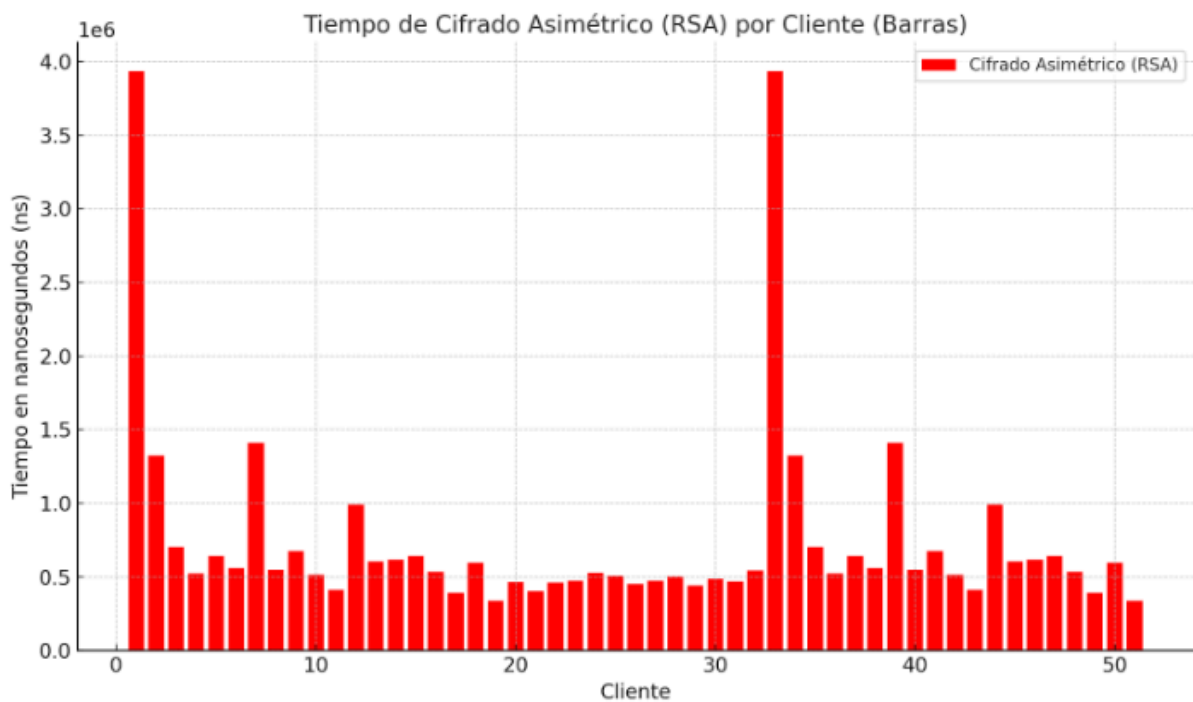
Valores obtenidos en cliente concurrente

Gráficas:

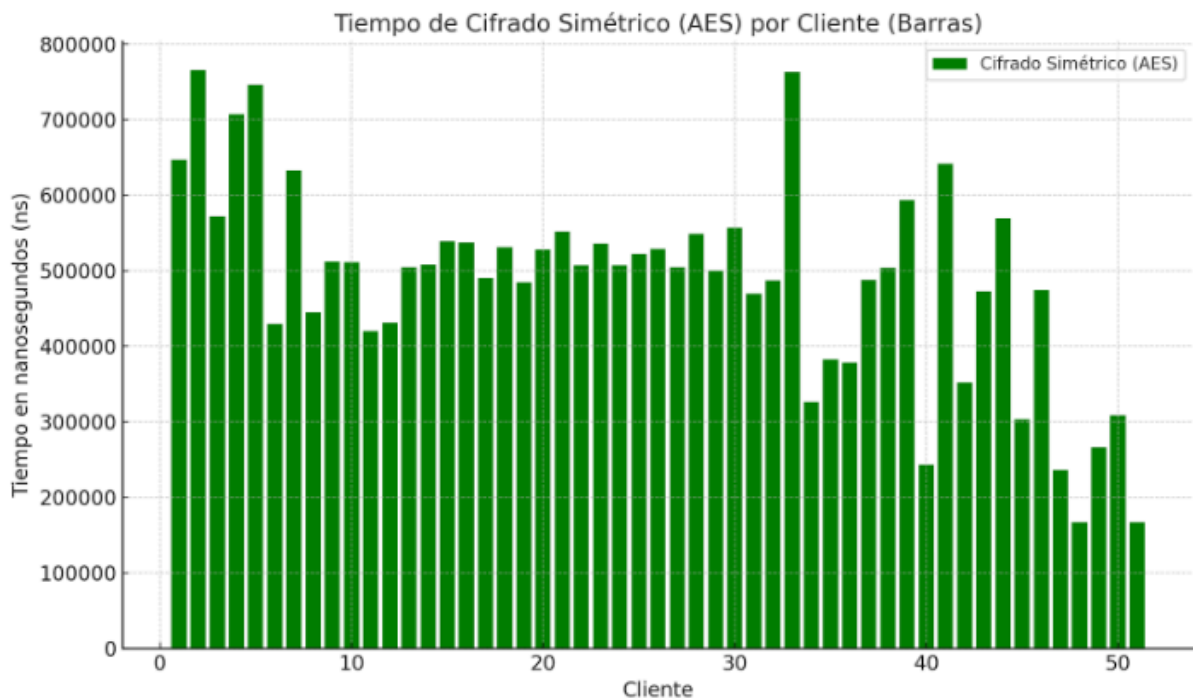
Cliente iterativo



Como se puede observar los datos oscilan bastante pero continúan teniendo unos picos y unos bajos siendo así estables, aquella estabilidad se reafirma cuando el promedio de datos dio: 248318 ns.



La gráfica muestra que pese al dato atípico registrado en el primer valor fueron estables sobre la media de: 612518 ns ignorando el primer dato atípico posiblemente presentado debido a limitaciones de hardware.



Así mismo las gráficas del cifrado simétrico muestran que los valores aumentan considerablemente pero se mantienen en promedio de 449303 ns lo cual es congruente con lo teórico y debido a que sus datos no fueron tan atípicos se consideraron todos.

Adicionalmente debido a facilidades a la hora de la toma de gráficas solo se tomaron los primeros 20 valores. Pero en realidad sí se tomaron los 32.

Conclusiones:

Para concluir, se sabe así que los valores obtenidos entran en un margen del aceptable al previsto lo cual deja inferir que la simulación es correcta. Adicionalmente, los valores dieron congruentes a los teóricos en cuanto a tiempo de cifrado mostrando una ventaja significativa del simétrico respecto al asimétrico, aunque a su vez considerando factores como la seguridad es importante considerar el uso de cifrado asimétrico a la hora de levantar servidores.

Continuando, observando los resultados obtenidos a lo largo de las simulaciones del caso iterativo se permite concluir que gracias a factores como el hardware se pueden tener resultados atípicos a los promedio pero aún así se comporta de manera congruente a lo esperado.

En cuanto a los caso del concurrente, se dice que se tuvo dificultades debido a que no se tomo en cuenta que habían threads que se peleaban por el recurso generando así errores en la salida pero pese a lo anterior y después de arreglarlo se puede concluir que el cifrado

En cuanto a los casos propuestos, se logra ver y comprobar el como la eficiencia de los recursos a lo largo de las simulaciones no aumenta desde cierto punto, y de ahí en adelante se muestra un desperdicio de recursos (como es el caso de aumentar la cantidad de marcos) consiguiendo valores similares pese aumentar el consumo de espacio (desde la creación de páginas más grandes hasta aumento en los marcos de página.+