Daniel Felipe Martínez Osorio – 201910380

Alejandro González Díez – 201821205

Informe Caso 3

Infraestructura Computacional

(i) Organización de los archivos en el zip

El proyecto cuenta con la siguiente arquitectura:

- → *caso3_a.gonzalezd1_df.martinezo*
  - → *docs*
    - → *Informe.pdf*
  - → *keys*
    - → *public.key*
  - → *src*
    - → *clases*
      - → *Tabla.java*
    - → *sockets*
      - → *Cliente.java*
      - → *Conexión.java*
      - → *MainCliente.java*
      - → *MainService.java*
      - → *Servidor.java*
    - → *tests*
      - → *Tester.java*

Dentro de la carpeta del proyecto, se encuentran 3 carpetas. La carpeta *docs*, la cual contiene el pdf del informe actual, la carpeta keys, la cual contiene un archivo *public.key* en donde se guarda la llave publica del servidor, y la carpeta *src*, la cual contiene 3 paquetes de clases (*clases*, *sockets* y *tests*).

En el paquete *clases*, se encuentra la clase *Tabla*, la cual es responsable de modelar la tabla con los datos recopilados. Esta contiene la lista de nombres, la lista de los ids de paquetes y la lista de los estados de los paquetes, junto con algunos métodos adicionales que se requieren para recuperar la información de dichas listas.

En el paquete *sockets*, se encuentran las siguientes clases. La clase *Conexion*, que contiene las constantes HOST y PORT para la comunicación entre sockets. La clase *Cliente* (hereda de la clase *Conexion*), que tiene el protocolo de comunicación por parte del cliente y las operaciones de cifrado respectivas. La clase *Servidor* (hereda de la clase *Conexion*), que contiene la información del servidor y también la información del protocolo que sigue para enviar y recibir

mensajes al cliente. La clase *MainCliente,* responsable de ejecutar el socket del cliente. Y, por último, la clase *MainServidor,* responsable de ejecutar el socket del servidor.

En el paquete *tests,* se encuentra la clase *Tester,* la cual se utiliza para realizar las tareas 1-5 del presente caso.

(ii) Instrucciones para correr el protocolo (Ver Anexo)

Para correr la aplicación principal, en primera instancia, es **importante** que**,** se utilicen los datos predefinidos en la tabla de la clase *Tabla.java* (ver anexo) para que el protocolo funcione correctamente. Una vez se tengan los datos recopilados de la tabla, para correr el prototipo, **primero** se debe correr la clase *MainServidor* y **posteriormente** la clase *MainCliente.* Es **recomendable** mantener la consola del cliente abierta en todo momento, puesto que, en dicha consola se imprime de manera detallada los 13 pasos del protocolo, y también porque el usuario debe interactuar con ella para que el protocolo funcione y termine su ejecución de manera correcta (por ejemplo, al iniciar sesión, ingresar el nombre del cliente, el id del paquete, etc.). Por último, se recomienda mantener ambas consolas abiertas para ver el flujo de datos entre ambos sockets (cliente y servidor).

Para correr el prototipo iterativo, se debe correr el archivo *Tester.java,* el cual se encuentra en el paquete *tests* dentro de *src.* Este archivo llama al método *tareaIterativa(),* el cual implementa la tarea 1 (i).

Para correr el prototipo concurrente, se debe correr el archivo *Tester.java,* el cual se encuentra en el paquete *tests* dentro de *src.* Este archivo llama al método *tareaConcurrente(),* el cual implementa la tarea 1 (ii).

(iii) En primera instancia, para la generación de la llave privada y pública del servidor, se utilizó la funcionalidad de java.security *KeyPairGenerator* en el método *generarLlavesPublicaPrivada(),* se especificó que se usaría el algoritmo RSA, y se estableció que se iba a generar una llave de 1024 bits; con esto ya se obtuvieron las llaves privada y pública, y para guardar la pública en un archivo se usó *FileOutputStream* de java.io para crear el archivo *public.key* y escribir en él la llave pública. La llave privada quedó guardada como atributo privado del Servidor.

La llave simétrica se generó en el método *generateKey(tamaño)* de la clase *Cliente*. Allí se creó una llave de 256 bits, y se usó la funcionalidad *KeyGenerator* de javax.crypto y se especificó que se usaría el algoritmo AES para la llave simétrica, esta llave quedó guardada como atributo privado del Cliente.

(iv) Respuestas de las tareas 1-5

1. **Medir el tiempo**
2. **Tablas de los escenarios**
    i. **Iterativo**

Iteraciones	Tiempos (nanosegundos)	
	Cifrado Simétrico	Cifrado Asimétrico
1	4796184.0	1.0216402E7
2	221799.0	1703597.0
3	315166.0	1811876.0
4	308251.0	2217524.0
5	240575.0	1502864.0
6	269996.0	2133101.0
7	281342.0	1971846.0
8	499651.0	1866820.0
9	655059.0	1467976.0
10	191219.0	817185.0
11	236591.0	607032.0
12	454368.0	1229436.0
13	1011098.0	760686.0
14	400487.0	939824.0
15	1110330.0	1423953.0
16	644125.0	941634.0
17	311720.0	966490.0
18	242493.0	672692.0
19	248461.0	648923.0
20	323050.0	1113401.0
21	402814.0	813899.0
22	323702.0	1086272.0
23	210370.0	590358.0
24	273274.0	809402.0
25	358697.0	773369.0
26	247844.0	570394.0
27	270737.0	1131157.0
28	247179.0	978284.0
29	684764.0	928232.0
30	248537.0	799407.0
31	208030.0	1021456.0
32	272565.0	1077771.0

ii. Concurrente

4 DELEGADOS:

Iteraciones	Tiempos (nanosegundos)	
	Cifrado Simétrico	Cifrado Asimétrico
1	262511	701105
2	292323	737759
3	392987	824283
4	304217	498675

16 DELEGADOS:

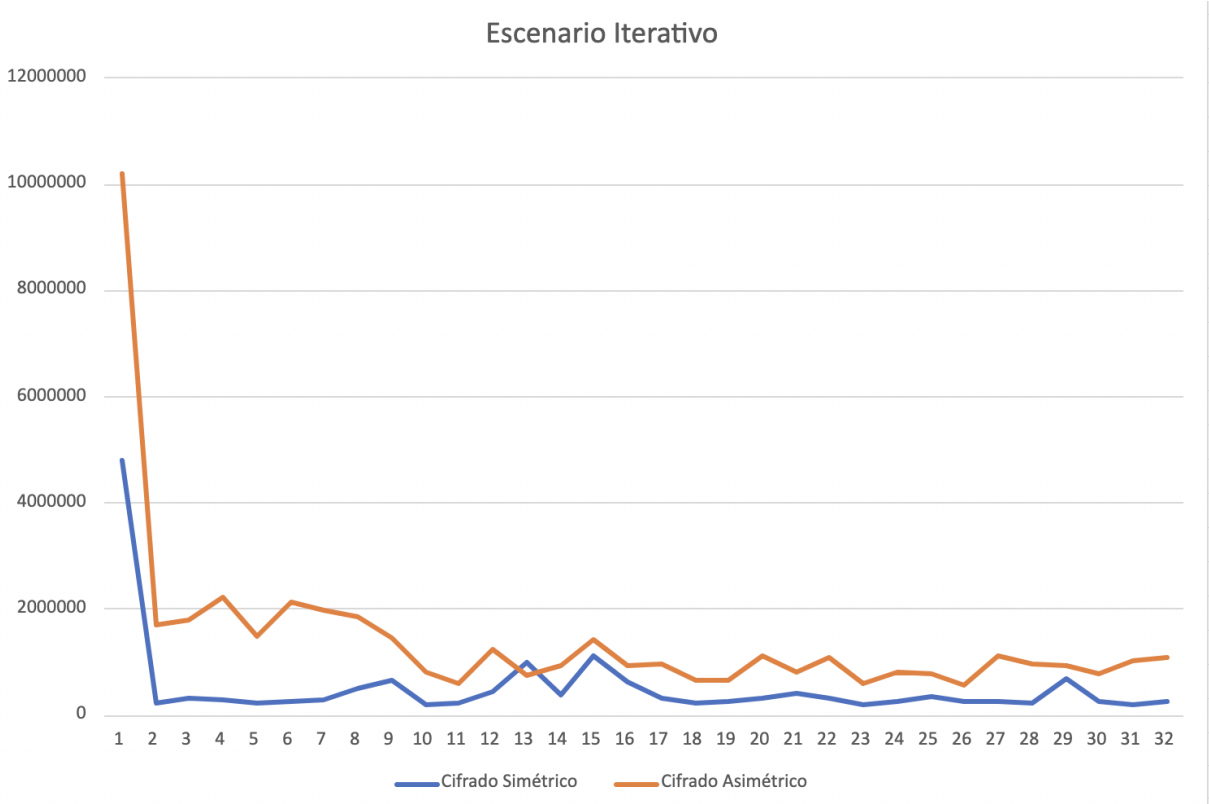
Iteraciones	Tiempos (nanosegundos)	
	Cifrado Simétrico	**Cifrado Asimétrico**
1	152793	509945
2	169075	531292
3	313810	510025
4	163468	479180
5	142402	470457
6	144292	477474
7	251560	660758
8	296601	559234
9	215883	807094
10	220277	451534
11	204752	700933
12	249522	519437
13	419101	525809
14	143350	466055
15	138580	480339
16	370487	477380

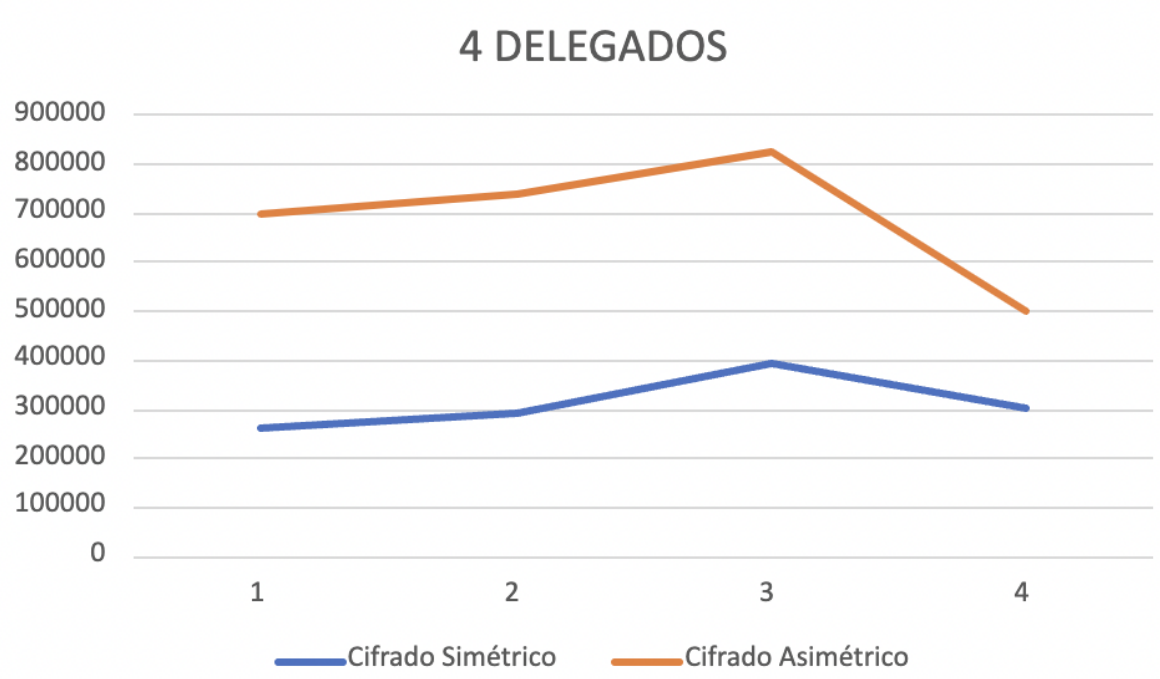
32 DELEGADOS:

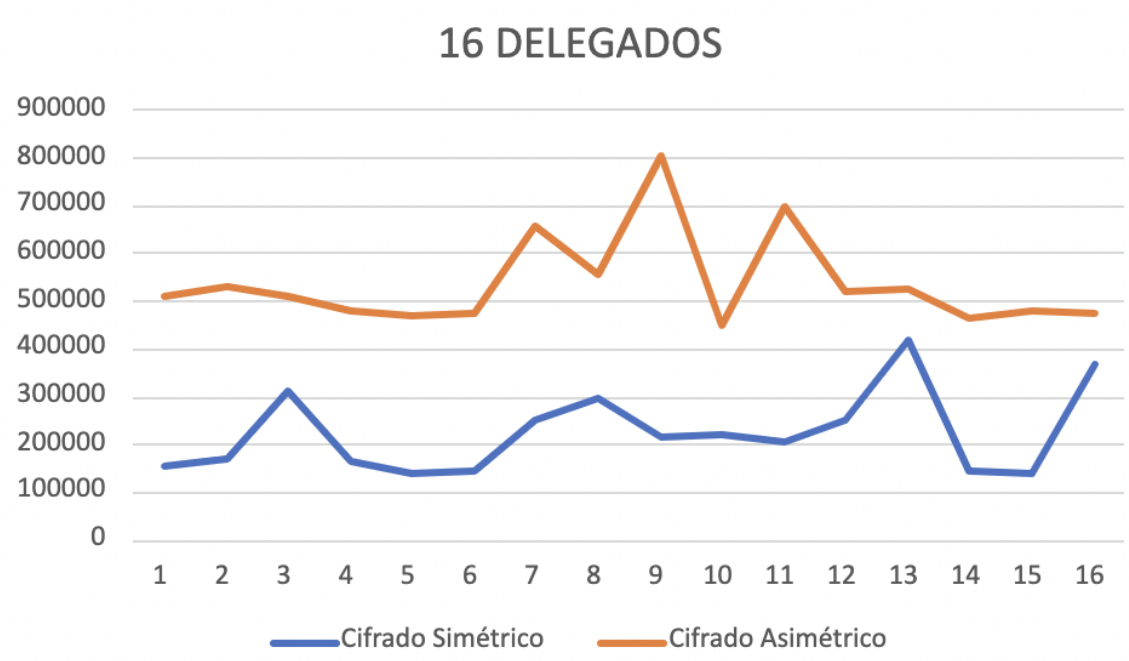
Iteraciones	Tiempos (nanosegundos)	
	Cifrado Simétrico	**Cifrado Asimétrico**
1	184956	503139
2	255713	1355275
3	147359	459922
4	162774	813603
5	154018	468450
6	178966	555079
7	221118	466345
8	197152	454162
9	200174	443374
10	185384	514398
11	139874	445931
12	201876	491489
13	134084	507920
14	205106	648580
15	307657	602201
16	148145	485014
17	186475	579895
18	157324	708969
19	180113	689229
20	180173	435848

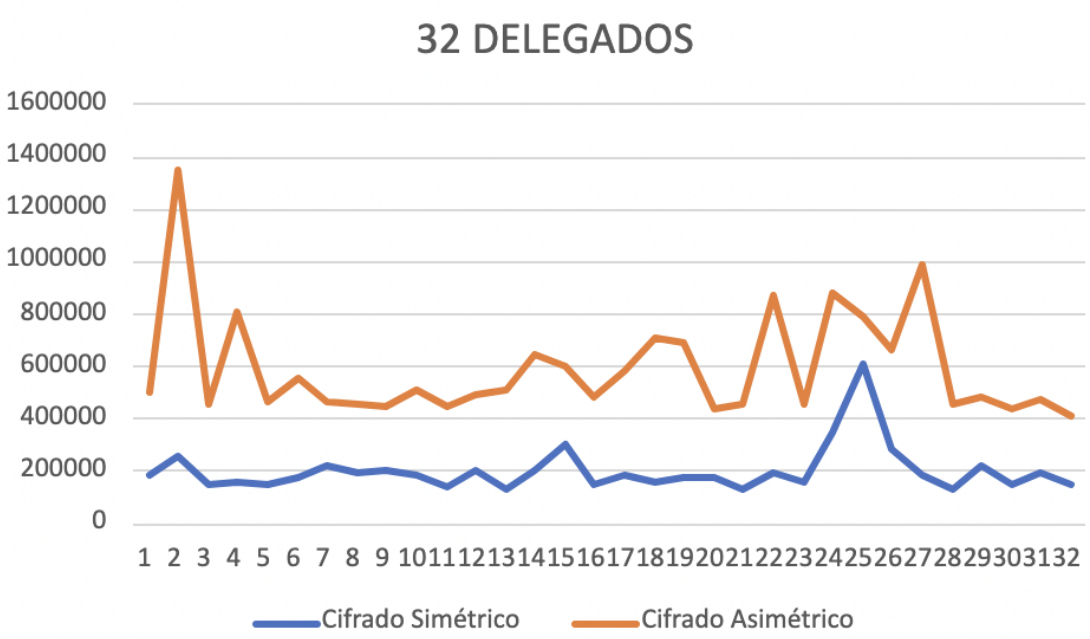
21	128565	457657
22	190923	871913
23	158478	460537
24	350864	884264
25	613974	789247
26	282028	664653
27	186448	986869
28	129446	456222
29	225742	481950
30	148844	438690
31	192807	479277
32	148452	411018

3. Graficas




4 DELEGADOS
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
1
2
3
4
Cifrado Simétrico
Cifrado Asimétrico


16 DELEGADOS
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Cifrado Simétrico
Cifrado Asimétrico



4. Comentarios respecto a las graficas

En cuanto al escenario iterativo (32 iteraciones), se evidencia como, para ambos algoritmos (simétrico y asimétrico) existe un pico de demora en la primera iteración, no tenemos clara la razón, pero asumimos que se debe a que, en esta iteración, el programa crea nuevas variables, atributos, métodos, etc. que hacen que el algoritmo se demore más por cuestión de nanosegundos.

Otro aspecto que notamos respecto a las gráficas es que, para todos los casos, la ejecución del algoritmo de cifrado asimétrico fue más lenta que la ejecución del algoritmo simétrico, cosa que debería pasar, pues se sabe que el tiempo de los algoritmos simétricos es menor al de los algoritmos asimétricos.

Asimismo, se evidencia que no hubo mayor diferencia entre los tiempos del escenario iterativo, con los tiempos del escenario concurrente.

5. Cálculos (velocidad del procesador)

Para realizar los cálculos a continuación, se utilizó el tiempo de demora promedio en nanosegundos (con base en todos los casos) del algoritmo simétrico y asimétrico.

$$T_{simetrico} = 331.470{,}01$$

$$T_{asimetrico} = 904.680{,}64$$

Primero se realizó la conversión a segundos, dividiendo el tiempo en nanosegundos por el factor 10^9.

$$T_{simetrico} = \frac{(331.470,01)}{10^9} = 0,0003315$$

$$T_{asimetrico} = \frac{(904.680,64)}{10^9} = 0,0009047$$

Una vez teniendo los tiempos en segundos, se procedió a dividir 1 entre el tiempo de demora encontrado en el paso anterior, para así obtener el número de retos, que mi maquina puede cifrar en promedio en 1 segundo.

$$T_{simetrico} = \frac{1}{0,0003315} = 3.016,86$$

$$T_{asimetrico} = \frac{1}{0,0009047} = 1.105,36$$

Referencias:

- https://www.programarya.com/Cursos-Avanzados/Java/Sockets
- https://www.baeldung.com/java-aes-encryption-decryption
- https://www.baeldung.com/java-rsa
- https://howtodoinjava.com/java/java-security/java-aes-encryption-example/
- https://gist.github.com/lesstif/655f6b295a619306405621e02634a08d

Anexo:

Los datos predefinidos de la tabla de la clase *Tabla.java* son los siguientes (es importante escribir los datos en consola tal y como aparecen en la tabla).

	Nombre Cliente	**ID del PKT**	**Estado del PKT**
1	carlos	id-0	PKT_EN_OFICINA
2	juan	id-1	PKT_RECOGIDO
3	maria	id-2	PKT_EN_CLASIFICACION
4	laura	id-3	PKT_DESPACHADO
5	carlos	id-4	PKT_EN_ENTREGA
6	juan	id-5	PKT_ENTREGADO
7	maria	id-6	PKT_DESCONOCIDO
8	laura	id-7	PKT_EN_OFICINA
9	carlos	id-8	PKT_RECOGIDO
10	juan	id-9	PKT_EN_CLASIFICACION
11	maria	id-10	PKT_DESPACHADO
12	laura	id-11	PKT_EN_ENTREGA
13	carlos	id-12	PKT_ENTREGADO
14	juan	id-13	PKT_DESCONOCIDO
15	maria	id-14	PKT_EN_OFICINA
16	laura	id-15	PKT_RECOGIDO

17	carlos	id-16	PKT_EN_CLASIFICACION
18	juan	id-17	PKT_DESPACHADO
19	maria	id-18	PKT_EN_ENTREGA
20	laura	id-19	PKT_ENTREGADO
21	carlos	id-20	PKT_DESCONOCIDO
22	juan	id-21	PKT_EN_OFICINA
23	maria	id-22	PKT_RECOGIDO
24	laura	id-23	PKT_EN_CLASIFICACION
25	carlos	id-24	PKT_DESPACHADO
26	juan	id-25	PKT_EN_ENTREGA
27	maria	id-26	PKT_ENTREGADO
28	laura	id-27	PKT_DESCONOCIDO
29	carlos	id-28	PKT_EN_OFICINA
30	juan	id-29	PKT_RECOGIDO
31	maria	id-30	PKT_EN_CLASIFICACION
32	laura	id-31	PKT_DESPACHADO

A continuación, se presenta un ejemplo de cómo debería quedar la consola del Servidor y del Cliente al finalizar la consulta. Para el ejemplo, se utilizó al cliente con nombre carlos y con el paquete cuyo id es id-0.

Consola del Servidor:

```
*** Servidor ***

Iniciando Servidor en el puerto: 1234
Esperando conexion...
Cliente conectado

Cliente: INICIO

Cliente: Reto <26122804166867573827720 8>

Cliente: LS <Vg7og2U1vSGmTSv9jHf2KsTKtgMWkp3BuboS5HyfXE7ug3/pgF81skHh8koyrxhu93INNzabQZowHOe1vburLBqa8jHZDGzWdaIYV6qoHa76/PXKaafDq86szGeHaVGylUkX8bSAKuJqxuMH9o3p8BXIz9l

Cliente: Nombre <ccIobcXcAdfaUOmUQ0zeWOZ/mOwa6+/lrIua3JLHf9ddMhe6JWKD0M9veMLfDrOCAojSPo35cR/QER/nJs2pvFO1sg40EA6Wu7yZgJ7K3Wwxv+2tujf3PahYIZxC75fGjbpnki/i/htGTz4L1eJ15XZ

Cliente: ID-PKT <aGBugtBPTLamPNEf8JvUwg==>

Cliente: TERMINAR
```

Consola del Cliente:

```
*** Cliente ***

[0] Se ha leido la llave publica del Servidor

[1] Escriba INICIO para iniciar sesion:
INICIO

[2] Servidor: ACK

[3] Enviando el reto al Servidor: 26122804166867573827720 8

[4] Servidor: Reto Cifrado <qsvF+VfZ1azYCiVYJ+6ynv+chV95nTlXysCsDcGUH3qyiRYvYcmVI7r0TjVkx9JVSTW2cgGY+OqLQk11+uS/bVlzIvlkS9qOaQ3eL3EJxUReYXkRReAU4n/eEtZ0Ra3ycfd2VsEPHc6Y

[5] Enviando la llave simetrica al Servidor: Vg7og2U1vSGmTSv9jHf2KsTKtgMWkp3BuboS5HyfXE7ug3/pgF81skHh8koyrxhu93INNzabQZowHOe1vburLBqa8jHZDGzWdaIYV6qoHa76/PXKaafDq86szGe

[6] Servidor: ACK

[7] Escriba su nombre:
carlos

[7] Enviando nombre cifrado al Servidor: ccIobcXcAdfaUOmUQ0zeWOZ/mOwa6+/lrIua3JLHf9ddMhe6JWKD0M9veMLfDrOCAojSPo35cR/QER/nJs2pvFO1sg40EA6Wu7yZgJ7K3Wwxv+2tujf3PahYIZxC75f

[8] Servidor: ACK

[9] Escriba el identificador de paquete:
id-0

[9] Enviando id de paquete cifrado al Servidor: aGBugtBPTLamPNEf8JvUwg==

[10] Servidor: Estado PKT <PKT_EN_OFICINA>

[11] Escriba ACK para confirmar:
ACK

[12] Resumen
        - El reto enviado fue: 26122804166867573827720 8
        - El nombre del Cliente que consulta es: carlos
        - El ID del paquete consultado es: id-0
        - El estado del paquete consultado es: PKT_EN_OFICINA
        - El tiempo de cifrado simetrico del reto fue de: 300860.0 nanosegundos
        - El tiempo de cifrado asimetrico del reto fue de: 7.5900326E7 nanosegundos

[13] Terminando conexion con el Servidor...

Fin de la conexion.
```