Universidad de los Andes
Ingeniería de Sistemas y Computación
ISIS 2203 Infraestructura Computacional
Semestre 2022-10

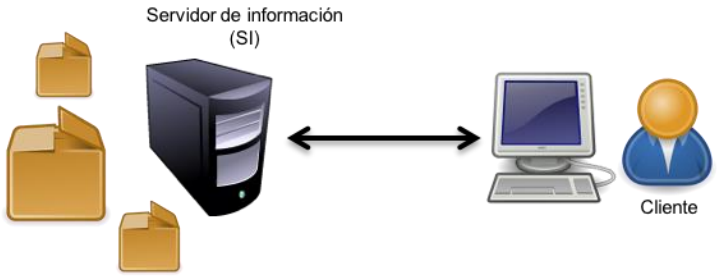
Caso de Estudio 3 – Canales Seguros
Sistema de rastreo de paquetes en una compañía transportadora

Objetivos

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

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.



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 nombre de cliente, un identificador de paquete y el estado de cada paquete recogido (PKT_EN_OFICINA, PKT_RECOGIDO, PKT_EN_CLASIFICACION, PKT_DESPACHADO, PKT_EN_ENTREGA, PKT_ENTREGADO, PKT_DESCONOCIDO), y responde las consultas de los clientes. Para simplificar el problema, tendremos un servidor con una tabla predefinida con nombre de usuario, identificador de paquete y estado de 32 paquetes. Para consultar el estado de un paquete, un cliente envía al servidor su nombre y el identificador de un paquete, el servidor recibe el identificador, consulta la información guardada y responde con el estado correspondiente. Si el identificador de usuario y de paquete no corresponden o no están en la tabla, el servidor retorna el estado ¨DESCONOCIDO¨.

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”.

Tanto servidor como cliente deben cumplir con las siguientes condiciones:

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

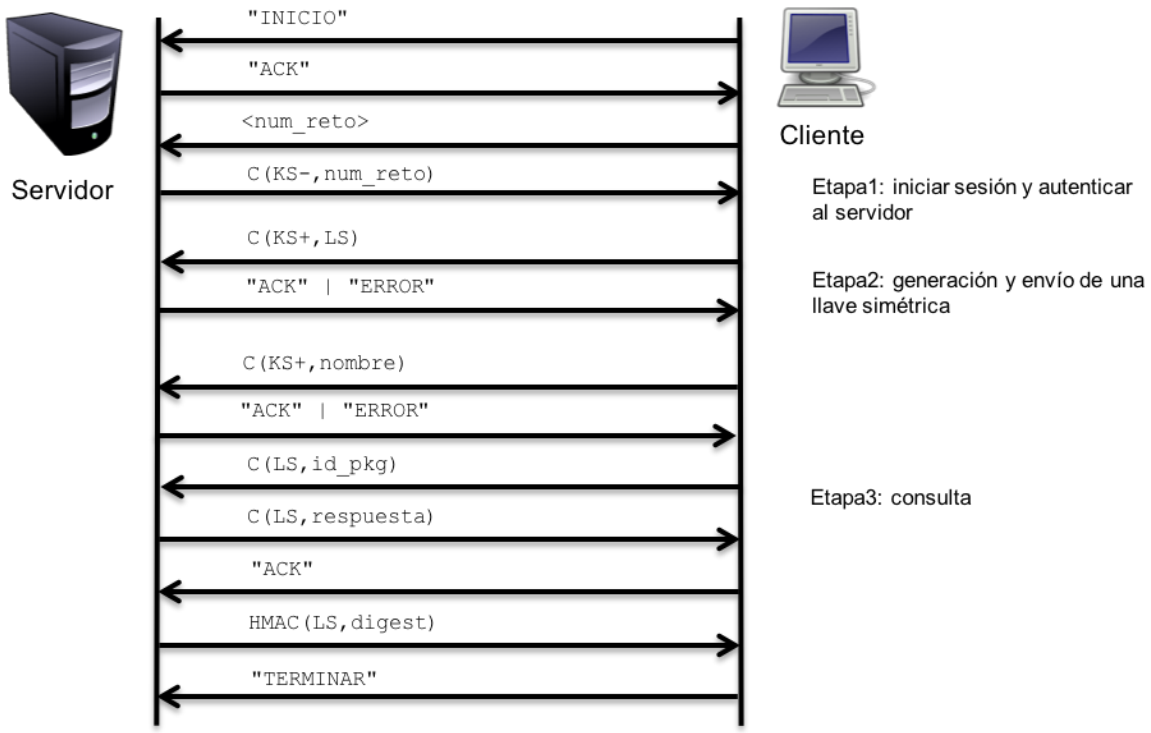
- Generaremos por adelantado las llaves pública y privada del servidor. Tenga en cuenta que en una instalación real la llave pública puede ser leída por cualquiera, pero la llave privada solo debería estar al alcance del propietario.

El cliente y el servidor deben comunicarse siguiendo el protocolo que se describe a continuación:

- El cliente lee la llave pública del servidor de un archivo. Es decir, suponemos que el cliente previamente obtuvo la llave pública del servidor.
- El cliente pide iniciar la sesión y espera un mensaje de confirmación del servidor (¨ACK¨).
- El cliente envía al servidor un reto (un número aleatorio de 24 dígitos).
- El servidor responde con el reto cifrado con su llave privada.
- Al recibir la respuesta, el cliente valida que el reto cifrado tenga el valor esperado; si la validación pasa entonces el cliente continúa con el protocolo; si la validación no pasa entonces el cliente termina la comunicación.
- El cliente genera una llave simétrica (LS), la cifra con la llave pública del servidor (KS+) y la envía al servidor.
- El servidor extrae la llave simétrica y responde con un mensaje de confirmación (¨ACK¨).
- El cliente envía su nombre y espera un mensaje de confirmación (¨ACK¨). Si el nombre no está en la tabla el servidor responde con un mensaje de error (¨ERROR¨).
- El cliente envía el identificador del paquete para el que está buscando información.
- Al recibir la solicitud anterior, el servidor verifica que el nombre de usuario recibido antes y el identificador del paquete estén en la tabla y correspondan. Responde con el estado del paquete.
- Para continuar con el protocolo el cliente responde con un mensaje de confirmación (¨ACK¨).
- El servidor envía un código de resumen o digest.
- El cliente debe validar la integridad de la información recibida y si la validación es exitosa entones debe desplegar la información en consola.

El servidor debe medir el tiempo que toma cifrar el reto con su llave privada, y el tiempo que toma cifrar ese mismo reto con la llave simétrica compartida. Observe que esta última operación solo se hace para medir el tiempo (no se usa)

La figura siguiente ilustra el protocolo:



PARA TENER EN CUENTA:

- Como algoritmos usaremos:

- AES. Modo ECB, esquema de relleno PKCS5, llave de 256 bits.
- RSA. Llave de 1024 bits.
- HMACSHA256.

Adicionalmente, resuelva las siguientes tareas:

1. Corra su programa en diferentes escenarios y mida el tiempo que el servidor requiere para cifrar el reto con cifrado simétrico y con cifrado asimétrico. 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, 16 y 32 delegados concurrentes. 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. Construya dos gráficas: una que muestre los tiempos para el caso simétrico (en cada uno de los escenarios) y una para el caso asimétrico (en cada uno de los escenarios).
4. Escriba sus comentarios sobre las gráficas, explicando los comportamientos observados.
5. Identifique la velocidad de su procesador, y estime cuántos retos puede cifrar su máquina por segundo, en el caso evaluado de cifrado simétrico y cuántos en el caso evaluado de cifrado asimétrico. Escriba todos sus cálculos.

Entrega:

- Cada grupo debe entregar un zip con:
  - archivos con la implementación del prototipo y los archivos de las llaves del servidor.
  - un subdirectorio docs con un informe que incluya: (i) la descripción de la organización de los archivos en el zip, (ii) las instrucciones para correr el prototipo incluyendo cómo correr el servidor y cómo correr los clientes de forma concurrente, (iii) descripción del esquema que siguieron para generación de las llaves y nombres de los archivos que las almacenan, (iv) las respuestas a las tareas 1 a 5.
  - **Al comienzo del informe, deben estar los nombres y carnés de los integrantes del grupo**. Si un integrante no aparece en el documento entregado, el grupo podrá informarlo posteriormente. Sin embargo, habrá una penalización: la calificación asignada será distribuida (dividida de forma equitativa) entre los integrantes del grupo.
- Recuerde incluir en su informe **todas las referencias** que use para resolver este proyecto.
- El trabajo se realiza en grupos de **2** personas. No debe haber consultas entre grupos.
- El grupo responde solidariamente por el contenido de todo el trabajo, y lo elabora conjuntamente (no es trabajo en grupo repartirse puntos o trabajos diferentes). Se puede solicitar una sustentación a cualquier miembro del grupo sobre cualquier parte del trabajo. Dicha sustentación será parte de la calificación de todos los miembros.
- El proyecto debe ser entregado en Bloque Neón.
- **La fecha límite de entrega es mayo 10, 2022 a las 23:50 p.m.**

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

Anexo

Dado que en Java hay problemas en la transmisión de los bytes cifrados por medio de sockets, manejaremos encapsulamiento con cadenas para transmisión. Los siguientes fragmentos presentan el código para:
(i) transformar bytes a cadenas antes de escribir en el socket por medio de println (socket_comunicacion.println( ))
(ii) leer cadenas y pasar a bytes datos leídos del socket por medio de readline (socket_comunicacion.readLine( )).

```
public String byte2str( byte[] b )
{
	// Encapsulamiento con hexadecimales
	String ret = "";
	for (int i = 0 ; i < b.length ; i++) {
		String g = Integer.toHexString(((char)b[i])&0x00ff);
		ret += (g.length()==1?"0":"") + g;
	}
return ret;
}

public byte[] str2byte( String ss)
{
	// Encapsulamiento con hexadecimales
	byte[] ret = new byte[ss.length()/2];
	for (int i = 0 ; i < ret.length ; i++) {
		ret[i] = (byte) Integer.parseInt(ss.substring(i*2,(i+1)*2), 16);
	}
	return ret;
}
```