Breve manual para la implementación de transferencia cifrada de archivos

Dificultades que puedes encontrar

Rubén Darío Ceballos, Juan Camilo Swan y Ana Fernanda Valderrama

A continuación en este documento se desarrolla un manual para realizar transferencia cifrada de archivos entre dos computadores. Para el cifrado se emplea el algoritmo AES, con clave de 128 bits. La clave a emplear será una clave de sesión, que se genera implementando el algoritmo Diffie-Hellman. Para asegurar posteriormente la integridad del archivo enviado, el emisor y el receptor generan un hash MD5 del archivo no cifrado.

Al final del documento, el lector debería ser capaz de enfrentarse a este desafío de la transferencia cifrada de archivos, que podría lucir como la imagen a continuación (Ver **Figura 1**).

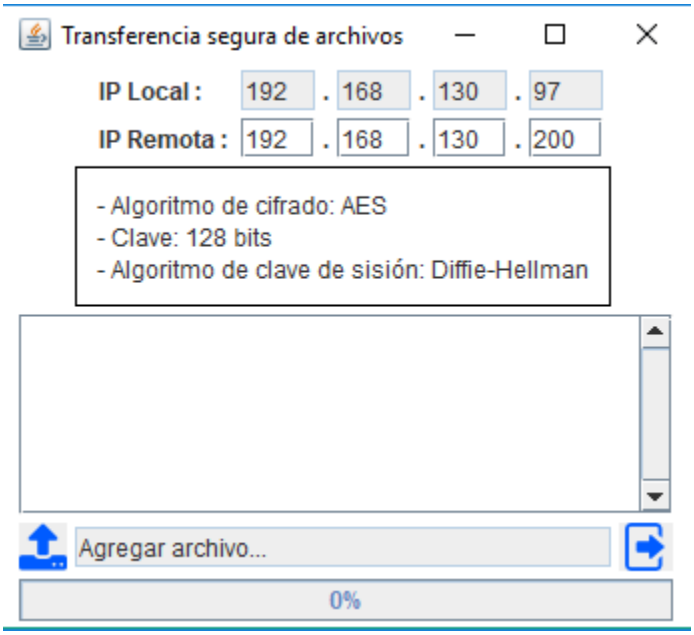

Figura 1. Interfaz transferencia de archivos cifrada.

PASOS

1. En primer lugar, se debe realizar la interacción cliente-servidor. Para la comunicación entre estos se establece una canal público confiable que emplea el protocolo TCP/IP, con esto se asegura que la transferencia de archivos y de información se va a llevar a cabo, como la transferencia de los parámetros necesarios para la creación de la clave Diffie-Hellman.

   ¡Dificultades que puedes encontrar!

A la hora de validar este canal de comunicación, si para la transmisión de los datos se está empleando el método readLine( ), puede que aparezca errores y que los datos no se reciban al otro lado del canal, por eso, es recomendable enviar y recibir bytes con la creación de un buffer.

Dato curioso:

Se conoce como *Diffie-Hellman Key Exchange* al algoritmo de intercambio de claves, surgió debido a la necesidad de tener una clave secreta de cifrado para generar seguridad en la comunicación ente un emisor y un receptor. *Diffie* y *Hellman* no fueron los primeros en trabajar en como intercambiar claves, pero sí los primeros en publicarlo, damos créditos a *James Ellis* y a *Clifford Cocks* por ser los primeros (según se sabe) en trabajar sobre esto. El algoritmo consiste básicamente en que dos partes, que van a establecer un canal de comunicación, se ponen de acuerdo para generar dos números un p de 2048 bits y un aleatorio g menor que p y mayor que 1, envían g por el canal sin ningún problema y luego cada parte genera un número aleatorio secreto (por ejemplo, el emisor genera un aleatorio a y el receptor un aleatorio b) y cada uno realizan la operación $g^x \ mod \ p$, obteniendo $A = g^a \ mod \ p$ y $B = g^b \ mod \ p$, generando un número único y enviándolo a la otra parte, en ésta se hace la operación de reversa, por ejemplo en el emisor $S = \ B^a \ mod \ p$ y en el receptor $S = A^b \ mod \ p$ obtienen el mismo S encriptando su comunicación (McDermott, 2016).

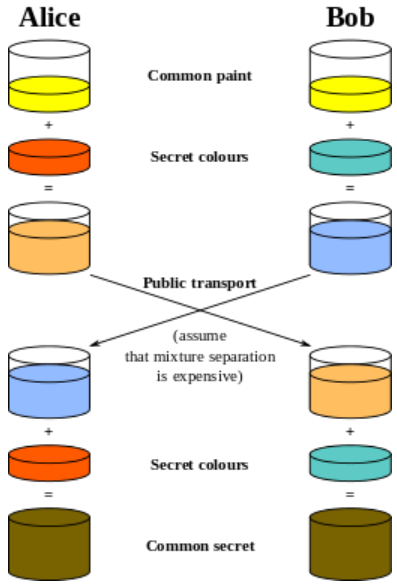

Figura 2. Ilustración Diffie-Hellman

2. Una vez el socket (canal) ha sido configurado en cliente y servidor se crea una clave secreta que cifrará la comunicación entre ellos dos, esta se crea como se mencionó anteriormente con el algoritmo conocido como *Diffie Hellman Key Exchange* esta clave será utilizada en AES. Existe una clase de Java KeyPairGenerator

que recibe como parámetro el tipo de algoritmo que se va a emplear, para luego poder crear la clave pública y la clave privada, para generar luego el acuerdo de llaves entre cliente y servidor, cada uno tiene 3 parámetros, dos números públicos iguales y uno privado (que nunca se transmite), con los cuáles genera una clave pública, que se envía de al otro, es decir el servidor al final tiene los 3 parámetros y la clave pública del servidor, y el servidor tiene los 3 parámetros y la clave pública del cliente.

¡Dificultades que se pueden encontrar!

En esta parte la principal dificultad es no tener claridad sobre la forma en la que se trabaja *Diffie- Hellman* en Java, es decir, no tener claros los parámetros para la creación de la clave, porque se desconoce por ejemplo, la existencia de clases en Java que tienen todos los métodos necesarios para la creación de la clave e inicialización de la clave como lo son DHParameterSpec (que crea los parámetros necesarios) y KeyPairGenerator (que inicializa el *Diffie-Hellman*, que se pone como parámetro).

⚠ **Advertencia:**

Diffie-Hellman es lo primero que debe hacerse porque sin esto no se puede realizar la encriptación para asegurar la comunicación.

3. El servidor es el encargado de realizar el cifrado del archivo que se va a enviar. El archivo se lee en bytes, se cifra en bytes y es enviado en bytes para evitar que se corrompa. Éste se cifra con la clave secreta y privada del servidor empleando AES del tipo CBC (encadenamiento de bloques cifrados) y PKCS5Padding, el modo de organización y agrupamiento de bloques CBC se escoge porque es el más difícil de atacar, al ser un proceso en cadena el descifrado de un bloque depende de todos los que le preceden, y PKCS5Padding es recomendado porque rellena el archivo a un tamaño específico, así en caso de la existencia de un hombre la mitad, éste no sabría cuál es el tamaño del archivo, y cómo tratar de descifrarlo.

4. En el servidor se había generado un hash MD5, antes del cifrado y envío del archivo, esto se realiza con el fin de imprimir sobre el mensaje una huella dactilar, ésta se conoce con el nombre de *message digest*, se considera imposible que dos mensajes tengan el mismo *message digest* (MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992). Luego, en el cliente cuando el archivo es recibido en bytes, antes de descifrarlo, se recibe el MD5 generado y enviado por el servidor, se desencripta el archivo en bytes y se genera el *message digest* por parte del cliente, así se comparan el recibido por el servidor y el generado por el cliente para verificar la integridad del archivo.

⚠ **Advertencia**

Al calcular el *message digest* es importante: en primer lugar, saber que algoritmo se va a emplear, en este caso MD5 (es necesario para crear la instancia). Hacer reset() para que después empiece a leer desde el inicio la cadena de bytes, se descifra el

archivo y se hace messageDigest.update(archivodescifrado) para actualizar la instancia de "message digest". Por último, se hace digest ( ), y se obtiene el resultado del hash en bytes. Para comparar con él generado por el cliente ambos hash se pasan a hexadecimal.

5. Una vez, se ha revisado si el archivo no está corrupto el cliente envía una respuesta al servidor de transferencia correcta, de lo contrario envía un mensaje de transferencia incorrecta.

6. Si la trasferencia se ha realizado correctamente el servidor cierra el canal.

CONCLUSIONES

Este proyecto integra muchos de los conceptos de cifrado que se estudiaron en el curso, pero además son necesarios conocimientos de programación en red, que fue necesario aplicar en el código y afianzar un poco más. Por ejemplo, se presentaron muchas dificultades en la transferencia del archivo por emplear un método, que por lo general siempre causa problemas entonces fue necesario consultar y realizar la transmisión de datos de otra manera. Pero es interesante enfrentarse a este tipo de desafíos y saber que los conocimientos adquiridos son aplicables en diferentes campos y que son necesarios para la protección de la información.

BIBLIOGRAFÍA

McDermott, J. (28 de Enero de 2016). *How Does Diffie-Hellman Key Exchange Work?* Obtenido de Learining Tree International: https://blog.learningtree.com/how-does-diffie-hellman-key-exchange-work/

MIT Laboratory for Computer Science and RSA Data Security, Inc. (Abril de 1992). *RFC 1321: The MD5 Message-Digest Algorithm.* Obtenido de IETF: https://tools.ietf.org/html/rfc1321

Para ejemplificación y orientación en el código:

Bruno. (7 de Diciembre de 2017). *Creating a Simple Java TCP/IP Server and Client Socket.* Obtenido de Pegaxchange: https://www.pegaxchange.com/2017/12/07/simple-tcp-ip-server-client-java/

C., S. (28 de Febrero de 2012). *Getting the IP address of the current machine using Java.* Obtenido de Stackoverflow: https://stackoverflow.com/questions/9481865/getting-the-ip-address-of-the-current-machine-using-java

ORACLE. (2017). *How to Use GridBagLayout.* Obtenido de Java Documentation: https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

Tongy, Z. (18 de Marzo de 2014). *Diffie Hellman key exchange example java*. Obtenido de Youtube: https://www.youtube.com/watch?v=j1yDsfWEAlA

Toth, C. (1 de Julio de 2013). *Exception in thread "main" java.security.InvalidAlgorithmParameterException: Prime size must be multiple of 64, and can only range from 512 to 1024 (inclusive) at com.sun.crypto.provider.DHKeyPairGenerator.initialize(DHKeyPairGenerator.java:120) at java.s.* Obtenido de Stackoverflow: https://stackoverflow.com/questions/17248095/diffie-hellman-key-exchange