Yair Osvaldo Cosio Escoto & Francisco Javier Luna Vázquez

Ycosio@ucol.mx & Fluna0@ucol.mx

Descripción breve

En este documento se demuestra cómo se realiza el envió de mensajes encriptados entre dos puntos y el servidor de seguridad.

DOCUMENTANCIÓN

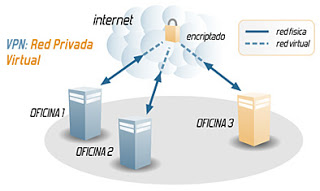
[Subtítulo del documento]

# Introducción

## Conocimientos básicos:

**Encriptación:**

Encriptación es el proceso mediante el cual cierta información o texto sin formato es cifrado de forma que el resultado sea ilegible a menos que se conozcan los datos necesarios para su interpretación. Es una medida de seguridad utilizada para que al momento de almacenar o transmitir información sensible ésta no pueda ser obtenida con facilidad por terceros.



**Clase “Random”:**

La clase Random proporciona un generador de números aleatorios. Es más flexible que la función random de la clase Math. La generación de números aleatorios adquiere gran relevancia para un programador.

**Clase “Socket”:**

Esta clase implementa sockets de cliente (también llamados simplemente "enchufes"). Un socket es un punto final para la comunicación entre dos máquinas.  
  
Socket público (Proxy Proxy)

Crea un socket no conectado, especificando el tipo de proxy, si lo hay, que debe ser usado independientemente de cualquier otra configuración.

Si existe un administrador de seguridad, su checkConnect método es llamado por la dirección del proxy y el número de puerto como sus argumentos. Esto podría resultar en una SecurityException.

Ejemplos:

Socket s = new Socket (Proxy.NO\_PROXY); creará un enchufe normal ignorando cualquier otra configuración proxy.

Socket s = new Socket (nuevos Proxy (Proxy.Type.SOCKS, nueva InetSocketAddress ("socks.mydom.com", 1080))); creará una toma de conexión a través de los SOCKS especificados del servidor proxy.

**Clase “Cipher”:**

Esta clase proporciona la funcionalidad de un sistema de cifrado criptográfico para cifrado y descifrado. Con el fin de crear un objeto de cifrado, la aplicación llama al método getInstance de la Cipher, y pasa el nombre de la transformación solicitada a la misma. Opcionalmente, se puede especificar el nombre de un proveedor.

Una transformación es una cadena que describe el funcionamiento (o conjunto de operaciones) que se realiza en la entrada dada, para producir una salida. Una transformación siempre incluye el nombre de un algoritmo criptográfico, por ejemplo:

Cipher c = Cipher.getInstance("DES/CBC/PKCS5Padding");

**Clase “SecretKeySpec”:**

Esta clase especifica una clave secreta de una manera independiente del proveedor.

Se puede utilizar para construir un SecretKey a partir de una matriz de bytes, sin tener que pasar a través de un SecretKeyFactory (a base de proveedor).

Esta clase sólo es útil para las claves secretas primas que pueden ser representados como una matriz de bytes y no tienen parámetros clave relacionados con ellos, por ejemplo, DES o Triple DES llaves.

public SecretKeySpec(byte[] key,

int offset,

int len,

String algorithm)

Parámetros:

key - el material clave de la clave secreta. Los primeros len bytes de la matriz a partir de las compensados se copian para proteger contra la modificación posterior, ambos inclusive.

Int offset - el desplazamiento en clave donde comienza el material clave.

len - la longitud del material de claves.

algoritmo - el nombre del algoritmo de clave secreta que se asocia con el material de clave dada. Véase el Apéndice A de la Guía de referencia Arquitectura de criptografía de Java para obtener información acerca de los nombres de algoritmo estándar.

# IMPLEMENTACIÓN

## INICIAR EL SERVIDOR

El primer paso para que los mensajes puedan ser enviados, encriptados y recibidos será iniciar el servidor que nos brindara este servicio y seguridad de la informacion.

String vector = "0123456789abcdef"; **//antes clave**

String claveA = "AAAAAAAAAAAAAAAA"; **//antes llaveA**

String claveB = "BBBBBBBBBBBBBBBB"; **//antes llaveB**

String llaveAB;

Random rnd = new Random();

try {

**//Iniciar Servidor**

ServerSocket socketServidor = new ServerSocket(8080);

**//Aceptar conexiones**

Socket socket = socketServidor.accept();

**//Nueva conexión, entrada y salida de datos**

System.out.println("Nueva conexión.");

BufferedReader entrada = new BufferedReader(

new InputStreamReader(socket.getInputStream()));

PrintWriter salida = new PrintWriter(socket.getOutputStream(), true);

**//Primer mensaje NO encriptado**

String msgin = entrada.readLine();

System.out.println(msgin);

String sg[]= new String[2]; //USUARIO1,USUARIO2,Nounce

sg = msgin.split(",");

llaveAB = "" + rnd.nextLong(); //Se genera una vector numerica aleatoria de longitud 16

if (llaveAB.length() > 16) {

llaveAB = llaveAB.substring(llaveAB.length() - 16);

System.out.println("LlaveAB generada: " + llaveAB);

}

**//Respuesta encriptada del servidor**

System.out.println("Enviando datos al cliente: ");

**//Encriptación de mensaje de B**

EncriptacionAES aesb = new EncriptacionAES(vector, claveB);

byte[] mensajeB = aesb.encripta(llaveAB + "," + sg[0]);

String encriptadoHexaB = aesb.convierteHex(mensajeB);

System.out.println("Mensaje B encriptado: " + mensajeB);

**//Encriptación de mensaje de A + Mensaje encriptado de B**

EncriptacionAES aes = new EncriptacionAES(vector, claveA);

byte[] textoEncriptado = aes.encripta(sg[2] + "," + sg[1] + "," + llaveAB + "," + encriptadoHexaB);

String encriptadoHexa = aes.convierteHex(textoEncriptado);

System.out.println("Texto encriptado en hexadecimal: " + encriptadoHexa);

**//Envio del mensaje Encriptado**

salida.print(encriptadoHexa);

**//Finalizar sesion**

salida.close();

socket.close();

socketServidor.close();

} catch (Exception e) {

System.out.println("Error: " + e.toString());

}

## ENCRIPTACION

Para hacer posible el transporte seguro de los mensajes entre los dos puntos será necesario la implementación de una clase de encriptación, en este caso: “EncriptacionAES”.

private final String vector;

private final String clave;

private Cipher cifrador;

private SecretKeySpec llave;

**En este punto declararemos el constructor de la clase para asegurarnos que se inicialice cuando lo invoquemos por primera ves.**

public EncriptacionAES(String vector, String clave) throws Exception {

this.vector = vector;

this.clave = clave;

cifrador = Cipher.getInstance("AES/CBC/PKCS5Padding", "SunJCE");

llave = new SecretKeySpec(clave.getBytes("UTF-8"), "AES");

}

**Ahora podemos observar los diferentes métodos empleados por esta clase como son: encriptar, desencriptar y conversiones de tipos de datos.**

public byte[] encripta(String textoPlano) throws Exception {

cifrador.init(Cipher.ENCRYPT\_MODE, llave, new IvParameterSpec(vector.getBytes("UTF-8")));

return cifrador.doFinal(textoPlano.getBytes("UTF-8"));

}

public String desencripta(byte[] textoCifrado) throws Exception {

cifrador.init(Cipher.DECRYPT\_MODE, llave, new IvParameterSpec(vector.getBytes("UTF-8")));

return new String(cifrador.doFinal(textoCifrado), "UTF-8");

}

public String convierteHex(byte[] datos) throws Exception {

StringBuilder constructor = new StringBuilder();

for(byte b : datos) {

constructor.append(String.format("%02X", b));

}

return constructor.toString();

}

public byte[] converteByte(String datos) throws Exception {

int tamano = datos.length();

byte[] constructor = new byte[tamano / 2];

for(int indice = 0; indice < tamano; indice += 2) {

constructor[indice / 2] = (byte)((Character.digit(datos.charAt(indice), 16) << 4) + Character.digit(datos.charAt(indice + 1), 16));

}

return constructor;

}

## CLIENTES

Para finalizar procedemos a declarar las clases para los clientes “A” y “B” los cuales se comunicaran por medio del chat en consola.

static Random rnd = new Random();

static String vector = "0123456789abcdef"; //antes clave

static String claveA = "AAAAAAAAAAAAAAAA"; //antes llaveA

static String claveAB;

static EncriptacionAES aes2;

static String msgout;

static String msgin;

public static void main(String[] args) {

try {

**//Iniciar socket**

Socket socket = new Socket("localhost", 8080);

**//Aceptar y enviar información**

BufferedReader entrada = new BufferedReader(

new InputStreamReader(socket.getInputStream()));

PrintWriter salida = new PrintWriter(socket.getOutputStream(), true);

**//Enviar cadena de conexión de nodos**

msgout = "A,B," + Integer.toString(rnd.nextInt(100));

salida.println(msgout);

System.out.println("Enviando cadena:" + msgout);

**//Recibiendo primera cadena encriptada**

System.out.println("Cadena de conexión recibida del servidor: ");

String datos = entrada.readLine();

String sg[];

**// = new String[3];//Nounce,Usurio2,KEY,InformaciónCifrada**

**//Desencriptando primer mensaje**

EncriptacionAES aes = new EncriptacionAES(vector, claveA);

byte[] encriptadoBytes = aes.converteByte(datos);

String textoDesencriptado = aes.desencripta(encriptadoBytes);

System.out.println("Cadena desencriptada: " + textoDesencriptado);

sg = textoDesencriptado.split(",");

claveAB = sg[2];

msgout = sg[3];

// while((datos = entrada.readLine()) != null) {

// System.out.println(datos);

// }

entrada.close();

socket.close();

} catch (Exception e) {

System.out.println("Error: " + e.toString());

}

**//-------------------CONEXION CON B-------------------------//**

try {

System.out.println("Intentado conexión con B");

**//Iniciar socket**

Socket socketB = new Socket("localhost", 8081);

**//Aceptar y enviar información**

BufferedReader entradaB = new BufferedReader(

new InputStreamReader(socketB.getInputStream()));

PrintWriter salidaB = new PrintWriter(socketB.getOutputStream(), true);

**//Enviar cadena de conexión de nodo B**

EncriptacionAES aesB = new EncriptacionAES(vector, claveAB);

salidaB.println(msgout);

System.out.println("Enviando cadena de conexión: " + msgout);

**//Recibiendo Respuesta Nounce de B**

msgin = entradaB.readLine();

System.out.println("Mensaje recibido del ClienteB: " + msgin);

**//Desencriptando primer mensaje**

byte[] encriptadoBytesB = aesB.converteByte(msgin);

String textoDesencriptadoB = aesB.desencripta(encriptadoBytesB);

System.out.println("Texto de cliente B: " + textoDesencriptadoB);

**//Respuesta al clienteB**

int nounceB = Integer.parseInt(textoDesencriptadoB);

nounceB --;

encriptadoBytesB = aesB.encripta(Integer.toString(nounceB));

msgout = aesB.convierteHex(encriptadoBytesB);

System.out.println("Respuesta encriptada: " + msgout);

salidaB.println(msgout);

String entradaTeclado;

Scanner entradaEscaner = new Scanner (System.in);

**//Creación de un objeto Scanner;**

while (true) {

msgin = entradaB.readLine();

System.out.println("MENSAJE DE B :" + msgin);

System.out.println("RESPUESTA: \n");

entradaTeclado = entradaEscaner.nextLine();

salidaB.println(entradaTeclado);

System.out.println("-------------");

}

} catch (Exception e) {

System.out.println("Error: " + e.toString());

}

}

# Referencias

Apr, Walter Sagástegui y Manuel Sierra. *Random.* Obtenido de http://aprenderaprogramar.com/index.php?option=com\_content&view=article&id=240:generacion-de-numeros-aleatorios-en-java-rangos-clase-random-ejemplos-ejercicios-resueltos-cu00906c&catid=58:curso-lenguaje-programacion-java-nivel-avanzado-i&Itemid=180Cuervo, V. (25 de Febrero de 2012). *Linea de codigo.* Obtenido de http://lineadecodigo.com/java/crear-la-ruta-de-un-fichero-con-java/

Oracle docs, Java™ Platform. *Class Socket.* Obtenido de http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html

Oracle docs, Java™ Platform*. Class Cipher.* Obtenido de http://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html

Oracle docs, Java™ Platform.*. Class SecretKey.* Obtenido de https://docs.oracle.com/javase/7/docs/api/javax/crypto/spec/SecretKeySpec.html

Blogspot, Gretel, Abdaleis y Analida. *Encriptacion de Datos.* Obtenido de http://encripdedatos.blogspot.mx/