Tema 5. Técnicas de Programación Segura.

Objetivos

- Identificar principios y prácticas de programación segura.
- Analizar técnicas y prácticas criptográficas.
- Definir políticas de seguridad.
- Emplear algoritmos criptográficos.
- Utilizar sockets seguros para la transmisión de la información.
- Utilizar técnicas de autenticación y autorización.

Contenidos

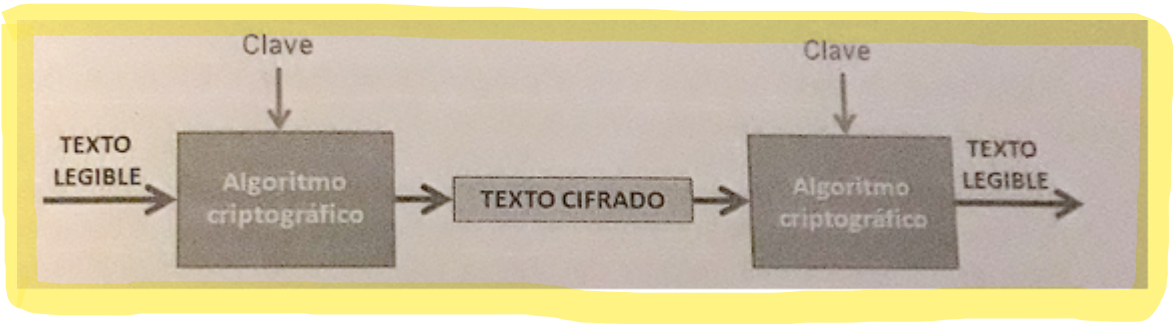
1.- Introducción.

Cuando se envía algo a través de sockets se envía como "texto plano", es decir, no sabemos si hay alguien usando un sniffer en la red y por tanto no sabemos si alguien está capturando los datos. Cualquier sistema que pretenda ser seguro necesitará usar cifrado.

La Criptografía es la disciplina que estudia el modo de transformar un mensaje (texto original) en un texto cifrado (criptograma) mediante una operación de cifrado que hace imposible a un tercero tomar conocimiento del contenido del mensaje.

Encriptar un texto es aplicarle un algoritmo que, en relación a una cierta variable (clave de encriptación), lo transforma en otro incomprensible e indescifrable por parte de quien no posee la clave. Si a ese texto cifrado se le aplica el mismo algoritmo, dependiente de una clave se obtiene el texto legible original. Este proceso se conoce como descifrado o desencriptación.

La encriptación/desencriptación fue inventada y utilizada originalmente para fines de seguridad en las trasmisiones de mensajes militares.



El *Cifrado Cesar* es un tipo de cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. El cifrado César muchas veces puede formar parte de sistemas más complejos de codificación.

Como todos los cifrados de sustitución alfabética simple, el cifrado César se descifra con facilidad y en la práctica no ofrece mucha seguridad en la comunicación.

Ejemplo: Si el alfabeto es el siguiente: ABCDEFGHIJKLMNÑOPQRSTUVWXYZ0123456789

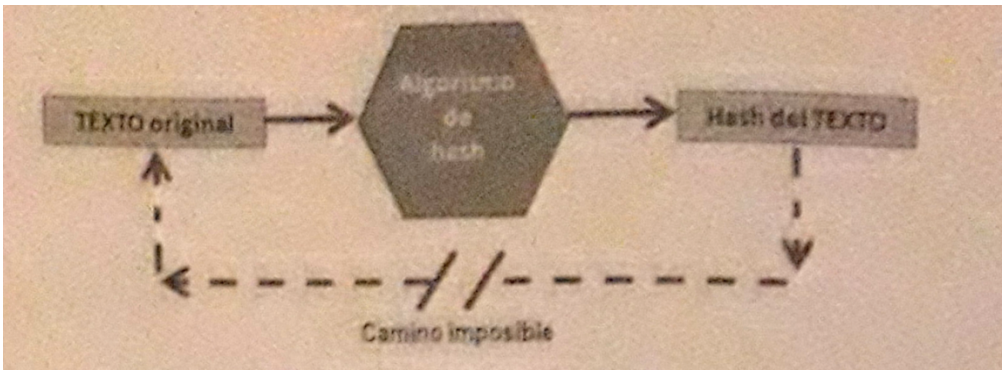
El mensaje HOLA MUNDO, con clave 1 sería IPMBNVÑEP

El descifrado simplemente implicaría el método inverso. Si el desplazamiento es un valor distinto de 1, lo único que hay que hacer es construir el alfabeto rotado tantas veces como el desplazamiento.

2.- Clases de algoritmos Criptográficos.

a) Criptografía mediante funciones de una sola vía.

Dado un mensaje se le aplica una función denominada función hash que permite obtener el llamado "resumen del mensaje" o "message digest". Es prácticamente imposible obtener el mensaje a partir de la función de hash por lo que reciben el nombre de algoritmos de una sola vía. Los algoritmos de cifrado de una sola vía más utilizados son MD5 y SHA-1.



Ejemplo: Obtener el hash para los algoritmos de una vía MD5, SHA1 y SHA256.

```
package ejemplohash1;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import javax.xml.bind.DatatypeConverter;

public class EjemploHash1 {
    public static void main(String args[]) {
        try {
            //Creamos tres instancias del tipo MessageDigest para cada uno de los
            // algoritmos de hash de una via
            MessageDigest algMD5 = MessageDigest.getInstance("MD5");
            MessageDigest algSHA1 = MessageDigest.getInstance("SHA-1");
            MessageDigest algSHA256 = MessageDigest.getInstance("SHA-256");

            // Convertimos el mensaje original en una cadena de bytes
            byte[] mensajeOriginal = "Aula Campus".getBytes();

            // Pasamos a cada uno de los algoritmos el mensaje Original en bytes
            algMD5.update(mensajeOriginal);
            algSHA1.update(mensajeOriginal);
            algSHA256.update(mensajeOriginal);
```

```
            // Mediante digest obtenemos el hash o resumen del mensaje original
            // para cada uno de los algoritmos de hash de una via
            byte[] mensajeMD5 = algMD5.digest();
            byte[] mensajeSHA1 = algSHA1.digest();
            byte[] mensajeSHA256 = algSHA256.digest();

            // Mostramos en hexadecimal el hash o resumen del mensaje para cada uno
            // de los algoritmos de hash de una vía
            System.out.println("MD5    hash: " +
                  DatatypeConverter.printHexBinary(mensajeMD5));
            System.out.println("SHA1   hash: " +
                  DatatypeConverter.printHexBinary(mensajeSHA1));
            System.out.println("SHA256 hash: " +
                  DatatypeConverter.printHexBinary(mensajeSHA256));
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

Ejemplo: Comparar clave encriptada con clave introducida por teclado.

```
package ejemplohash2;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class EjemploHash2 {
    public static void main(String args[]) {
        try {
            // Instancia para pedir datos por teclado
            Scanner entrada = new Scanner(System.in);
            // Instancia para uso del algoritmo de una via SHA
            MessageDigest algSHA = MessageDigest.getInstance("SHA");
            // Clave
            String clave = "mario";
            // Convertirmos la clave en bytes.
            byte dataBytes[] = clave.getBytes();
            // Se pasa el texto en bytes al algoritmo
            algSHA.update(dataBytes);
            // Se obtiene el hash del texto
            byte[] claveEncriptada = algSHA.digest();

            // Introdcimos la clave por teclado
            System.out.print("Introduce clave: ");
            String miClave = entrada.nextLine();
            //Se pasa el texto introducido por teclado al algoritmo en bytes
            algSHA.update(miClave.getBytes());
            // Se obtiene el hash del texto introducido por teclado
            byte miClaveEncriptada[] = algSHA.digest();

            // Se comprueban si el hash de la clave y el texto introducido
            // por teclado son iguales
            if (MessageDigest.isEqual(miClaveEncriptada, claveEncriptada)) {
                System.out.println("La clave es correcta");
            } else {
                System.out.println("La clave no es correcta");
```

```
            }
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

Uso de Base64 para codificar/decodificar mensajes en SHA1, SHA256 o MD5.

Base64 es un sistema numérico que se emplea en muchos ámbitos de la informática para representar información binaria.

Todos los sistemas de numeración tienen una lista de símbolos que utilizan para representar valores, por ejemplo:

- **Binario**: '01'
- **Decimal**: '0123456789'
- **Hexadecimal**: '0123456789ABCDEF'
- **Base64**: 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'

Como vemos, es un subconjunto de ASCII, y tiene como particularidad que todos sus caracteres son imprimibles, por eso, si pasamos cualquier información a su representación de base64, tenemos la seguridad de que no tendremos problemas al transmitirla, almacenarla o leerla. Podríamos codificar un archivo ejecutable, una imagen, archivos de sonido o simplemente como en este caso, Texto Plano.

Por lo tanto, Base64:

- No es un método de encriptación. Si bien ofusca la información ante la simple vista no hace falta mucho conocimiento ni pericia para recuperar la información original.
- Pesa más que la información original.
- Se usa para guardar o transmitir información previamente cifrada en SHA1, SHA256 o MD5 por ejemplo, que suele presentar caracteres problemáticos para ser transmitidos o almacenados.

Codificar/decodificar bytes mediante la clase Base64 usando Java.

Para codificar bytes usando la codificación Base64 disponemos del método Base64.getEncoder().encodeToString que codifica un array de bytes y devuelve un String con la codificación.

Para decodificador bytes usando las codificación Base64 disponemos del método Base64.getDecoder().decode que decodifica un String y devuleve un array de bytes con la decodificación.

Ejemplo: Codificación y Decodificación en Base64.

```
package base64cd;

import java.io.UnsupportedEncodingException;
import java.util.Base64;


public class Base64CD {
```

```
    public static void main(String[] args) throws UnsupportedEncodingException {
// Cadena Original a codificar
        String cadenaDeTexto = "Aula Campus Centro de Estudios";
// Cadena Codificada
        String cadenaCodificada;
// Muestra cadena original
        System.out.println("Cadena original > " + cadenaDeTexto);
// Codifica Cadena Original
        cadenaCodificada = codificar(cadenaDeTexto);
// Muestra Cadena Encriptada
        System.out.println("Cadena Codifcada > " + cadenaCodificada);
// Decodifica la cadena codificada
        byte[] cadenaDecodificadaBytes = decodificar(cadenaCodificada);
// Muestra la cadena desencriptada que debe coincidir con la original
        String cadenaDecodificada = new String(cadenaDecodificadaBytes);
        System.out.println("Cadena decodificada > " + cadenaDecodificada);
    }
// Método que codifica una cadena a Base64

    private static String codificar(String s) throws UnsupportedEncodingException {
        return Base64.getEncoder().encodeToString(s.getBytes("utf-8"));
    }
// Método que decodifica una cadena en Base64

    private static byte[] decodificar(String s) throws UnsupportedEncodingException {
        byte[] decode = Base64.getDecoder().decode(s.getBytes("utf-8"));
        return decode;
    }
}
```

Ejemplo: Codificación en Base64 de hash en SHA256.

Para encriptar las contraseñas gastaremos un **MessageDigest** con el algoritmo SHA-256. Esto crea una cadena ilegible e imposible de transferir por el buffer de conexión al api, por lo que deberemos codificarla en Base64.

```
package ejemplohashbase64;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

public class EjemploHashBase64 {
    public static void main(String args[]) {
        try {
            //Creamos tres instancias del tipo MessageDigest para el
            // algoritmo de hash de una via SHA256
            MessageDigest algSHA256 = MessageDigest.getInstance("SHA-256");
            // Convertimos el mensaje original en una cadena de bytes
            byte[] mensajeOriginal = "Aula Campus".getBytes();
            // Pasamos al algoritmo el mensaje Original en bytes
            algSHA256.update(mensajeOriginal);

            // Mediante digest obtenemos el hash o resumen del mensaje original
            // para el algoritmo de hash de una via SHA256
            byte[] mensajeSHA256 = algSHA256.digest();

            // Mostramos hash o resumen del mensaje que puede
```

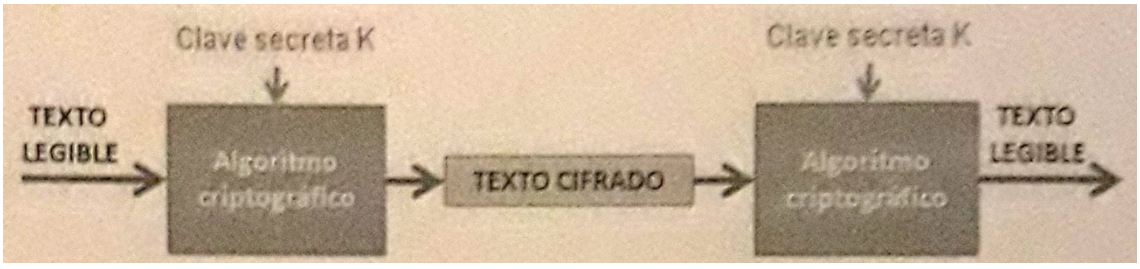
```
            // contener caracteres no imprimibles
            String mensajeHash = new String(mensajeSHA256);
            System.out.println("SHA256 hash: " + mensajeHash);
            // Codificamos a Base64 el hash SHA256
            String codificado = Base64.getEncoder().encodeToString(mensajeSHA256);

            // Mostramos en Base64 el hash o resumen del mensaje para
            // el algoritmo de hash de una vía SHA256
            System.out.println("SHA256 hash codificado: " + codificado);

            // Decodificamos el hash codificado en Base64
            String decodificado = new String(Base64.getDecoder().decode(codificado));

            // Mostramos decodificado el hash o resumen del mensaje que
            // puede contener caracteres no imprimibles
            System.out.println("SHA256 hash decodificado: " + decodificado);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

b) Algoritmos de Clave Secreta o Criptografía Simétrica (Clave privada)

El emisor y el receptor comparten el conocimiento de una clave que no debe ser revelada a ningún otro (clave secreta). Utilizan la misma clave para cifrar y descifrar el mensaje (simétrica). Principal problema de seguridad: Intercambio de claves entre el emisor y el receptor. La clave debe ser muy difícil de adivinar ya que hoy en día los ordenadores pueden adivinar claves muy rápidamente. Hoy por hoy se están utilizando ya claves de 128 bits que aumentan el "espectro" de claves posibles (2 elevado a 128) de forma que aunque se uniesen todos los ordenadores existentes en estos momentos no lo conseguirían en miles de millones de años. Los algoritmos más populares de cifrado simétrico son DES y AES.

Principales desventajas:

- Distribución de las claves: Peligro de que muchas personas deban conocer una misma clave y la dificultad de almacenar y proteger muchas claves diferentes.
- La seguridad en clave simétrica reside en la propia clave secreta, y por tanto el principal problema es la distribución de esta clave a los distintos usuarios para cifrar y descifrar la información.
- Si cae en manos equivocadas ya no podríamos considerar que la comunicación es segura y deberíamos generar una nueva clave.



AES(Advanced Encryption Standard) es un esquema de cifrado por bloques desarrollado por el gobierno de los Estados Unidos lanzado como estándar en el año 2002 y que se ha convertido en uno de los métodos de encriptación simétricos mas populares. La misma clave con la que se genera el mensaje codificado se puede desencriptar y recuperar el mensaje original, por lo que solo requiere almacenar una sola clave.

El primer paso para encriptar o desencriptar datos es convertir la clave de una cadena de texto a un objeto SecretKeySpec el cual es usado por las clases que incluye Java. Para esto obtendremos el SHA-1 del texto de la clave y lo pasaremos a un array de bytes para usarlo en el constructor de la clase SecretKeySpec.

La encriptación es realizada con un objeto Cypher el cual se inicializa indicando cual algoritmo de encriptación se usara, que operación encriptado o desencriptado se realizara y la clave que se usara. Hecho esto basta con pasarle los datos como un arreglo de bytes al método doFinal y le regresara el array de bytes del resultado de la encriptación.

Para recuperar los datos encriptados es un proceso muy similar al anterior, también se usa un objeto Cypher para hacer el trabajo y se inicializa con ayuda de la clave de encriptación, la principal diferencia es que se le indica que la operación será la desencriptación de datos. El objeto Cypher basta con pasarle los bytes de los datos encriptados al método doFinal y obtendrá los bytes del mensaje original, solo necesita convertirlos de byte[] a String.

Ejemplo:

```
package clavesimetrica;
import java.io.*;
import java.security.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class ClaveSimetrica {
    public static void main(String[] args) {
        try {
            // Clave en formato texto
            String claveEncriptacion = "mario";
            // Mensaje Original
            String datosOriginales = "AulaCampus";

            // Llamada al método encriptar que devuelve em mensaje original
            // encriptado usando la clave de encriptación/desencriptación
            String encriptado = encriptar(datosOriginales, claveEncriptacion);

            // Llamada al método desencriptar que devuelve em mensaje original
            // desencriptado usando la clave de encriptación/deesncriptación
            String desencriptado = desencriptar(encriptado, claveEncriptacion);

            // Muestra mensaje original
            System.out.println("Mensaje Original: " + datosOriginales);
            // Muestra mensaje encriptado
            System.out.println("Escriptado  : " + encriptado);
            // Muestra mensaje desencriptado
```

```
            System.out.println("Desencriptado  : " + desencriptado);
        } catch (UnsupportedEncodingException | NoSuchAlgorithmException |
                InvalidKeyException | NoSuchPaddingException |
                IllegalBlockSizeException | BadPaddingException ex) {
            System.out.println(ex.getMessage());
        }
    }

    // Metodo crearClave que genera clave para encriptar y desencriptar mensajes
    // La clave se pasa como un String y se encripta con el algoritmo SHA-1
    // para devolver una instancia de la clase SecretKeySpec que utilizaremos para
    // encriptar/desencriptar el mensaje
    public static SecretKeySpec crearClave(String clave)
            throws UnsupportedEncodingException, NoSuchAlgorithmException {
        // Convertimosla clave en bytes
        byte[] claveBytes = clave.getBytes("UTF-8");
        // Creamos una instancia de algoritmo SHA-1 para encriptar la clave
        MessageDigest algSHA = MessageDigest.getInstance("SHA-1");

        // Pasamos la clave en bytes a la instancia del algoritmo de encriptación
        algSHA.update(claveBytes);
        // Obtenemos el hash de la clave (clave encriptada)
        byte[] claveEncriptacion = algSHA.digest();
        // La clave AES deben tener un tamaño de 16 bytes por lo que extraemos
        // los primeros 16 bytes de la clave encriptada
        claveEncriptacion = Arrays.copyOf(claveEncriptacion, 16);
        // Creamos la clave simetrica del tipo SecretKeySpec para utilizar
        // en el proceso de encriptado/desencriptado del mensaje con algoritmo AES
        SecretKeySpec clavePrivada = new SecretKeySpec(claveEncriptacion, "AES");
        // Retornamos la clave Privada para encriptar/desencriptar mensajes
        return clavePrivada;
    }

    public static String encriptar(String datos, String claveSecreta)
            throws UnsupportedEncodingException, NoSuchAlgorithmException,
            InvalidKeyException, NoSuchPaddingException,
            IllegalBlockSizeException, BadPaddingException {
        // Genera la clave secreta encriptada
        SecretKeySpec secretKey = crearClave(claveSecreta);
        // Crea la instancia Cipher para cifrar mensaje con algoritmo AES
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        // Inicializa el cifrado con la clave secreta
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        // Convierte el mensaje a cifrar en bytes
        byte[] datosEncriptar = datos.getBytes("UTF-8");
        // Mediante el método doFinal cifra el mensaje obteniendo el
        // mensaje cifrado en bytes
        byte[] bytesEncriptados = cipher.doFinal(datosEncriptar);
        // codifica el mensaje en un String a Base64
        String encriptado = Base64.getEncoder().encodeToString(bytesEncriptados);
        // Retorna mensaje cifrado con algoritmo AES y clave simetrica
        // y codificado en base64
        return encriptado;
    }
```

```
    public static String desencriptar(String datosEncriptados, String claveSecreta)
```

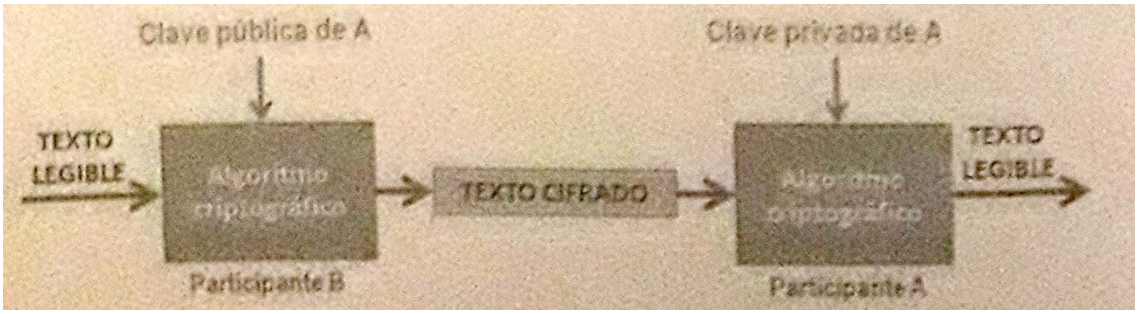
```
            throws UnsupportedEncodingException, NoSuchAlgorithmException,
            InvalidKeyException, NoSuchPaddingException,
            IllegalBlockSizeException, BadPaddingException {
        // Genera la clave secreta encriptada
        SecretKeySpec secretKey = crearClave(claveSecreta);
        // Crea la instancia Cipher para cifrar mensaje con algoritmo AES
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
        // Inicializa el descifrado con la clave secreta
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        // Decodifica el mensaje de Base64
        byte[] bytesEncriptados = Base64.getDecoder().decode(datosEncriptados);
        // Mediante el método doFinal cifra el mensaje obteniendo el
        // mensaje descifrado en bytes
        byte[] datosDesencriptados = cipher.doFinal(bytesEncriptados);
        // Convertimos bytes de mensaje descifrado a String
        String datos = new String(datosDesencriptados);
        // Retornamos el mensaje descifrado como una cadena de texto
        return datos;
    }
}
```

- **Algoritmos de Clave Publica o Criptografía Asimétrica.**

El emisor del mensaje (Participante B) emplea una clave pública para encriptar el mensaje, difundida previamente por el receptor (Participante A). El receptor emplea la clave privada correspondiente para desencriptar el mensaje. En general, estos algoritmos se basan en que cada participante genera una pareja de claves relacionadas entre sí. La clave pública la puede conocer todo el mundo para enviar mensajes cifrados al Participante A, pero sólo este puede desencriptar el mensaje pues conoce su clave privada. El Algoritmo asimétrico más popular es el RSA. Resumiendo, el emisor usa la clave pública del destinatario para cifrar el mensaje. Solo con la clave privada del destinatario se puede descifrar.

La criptografía asimétrica resuelve la desventaja principal de la simétrica ya que no se necesita canales seguros para mandar la clave. La distribución de claves es más fácil y segura ya que la clave que se distribuye es la pública manteniéndose la privada para el uso exclusivo del propietario. Pero este tipo de algoritmos también tiene desventajas:

- Son poco eficientes. Las claves deben de ser largas y se tarda bastante tiempo en aplicarlas.
- Utilizar las claves privadas repetidamente puede hacer que reciban ataques criptográficos que se basan en analizar paquetes cifrados.



Ejemplo: Encriptado/Desencriptado con algoritmo RSA de clave asimétrica pública.

```
/* El receptor del mensaje genera las claves Pública y Privada.
Envía la clave Publica al emisor para encriptar el mensaje.
El emisor envía el mensaje encriptado en RSA a receptor que
lo desencripta con la clave privada*/
package clavepublicareceptor;
import java.io.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import java.util.*;
public class ClavePublicaReceptor {
    public static void main(String[] args) throws InvalidKeyException,
        IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException,
        NoSuchPaddingException,  UnsupportedEncodingException,  InvalidKeySpecException
{
        // Generamos instancia q permite generar par de claves publica y privada
        KeyPairGenerator generadorClaves = KeyPairGenerator.getInstance("RSA");
        // Usaremos una longitud de clave de 1024 bits
        generadorClaves.initialize(1024);
        // generamos en claves el par de claves
        KeyPair claves = generadorClaves.generateKeyPair();

        // Mensaje Original
        String mensajeOriginal = "AulaCampus";
        // Obtenemos la clave publica
        Key clavePublica = claves.getPublic();
        // Convertimos es String para poder enviar la clave de tipo Key al emisor
        String clavePublicaTexto = keyToString(clavePublica);
        System.out.println("Clave Publica: " + clavePublicaTexto);

        // Instancia de Scanner para introducir por teclado el mensaje encriptado
        // del emisor
        Scanner teclado = new Scanner(System.in);
        System.out.print("Mensaje Encriptado RSA: ");
        String mensajeEncriptado = teclado.nextLine();

        // Obtenemos la clave privada
        Key clavePrivada = claves.getPrivate();

        // Desencriptamos el mensaje a partir de la clave Privada
        String mensajeDesencriptado = descifrar(mensajeEncriptado, clavePrivada);

        // Mostramos mensaje descifrado
        System.out.println("El mensaje descifrado es: " + mensajeDesencriptado);


    }

    public static String keyToString(Key clavePublica) {
        // Convertimos la clave pública a bytes
        byte[] clavePublicaBytes = clavePublica.getEncoded();
        // Codificamos en Base64 la clave en bytes para obetener el String
        String clave = Base64.getEncoder().encodeToString(clavePublicaBytes);
        // Retornamos la clave
        return clave;
    }
```

```
    public static String descifrar(String paraDescifrar, Key claveDescifrado)
            throws InvalidKeyException, IllegalBlockSizeException,
            BadPaddingException, NoSuchAlgorithmException,
            NoSuchPaddingException {
        // Decodifica el mensaje de Base64
        byte[] paraDescifrarBytes = Base64.getDecoder().decode(paraDescifrar);

        // Creamos una instancia Cipher para cifrar mensaje con algoritmo RSA
        Cipher cifrador = Cipher.getInstance("RSA");
        // Iniciamos instancia del cifrador en modo descifrado con la clave privada
        cifrador.init(Cipher.DECRYPT_MODE, claveDescifrado);
        // Mediante el método doFinal descifra el mensaje obteniendo el
        // mensaje descifrado en bytes
        byte[] mensajeDescifradoBytes = cifrador.doFinal(paraDescifrarBytes);
        // Convertimos bytes de mensaje descifrado a String
        String mensajeDescifrado = new String(mensajeDescifradoBytes);
        // Retornamos el mensaje cifrado
        return mensajeDescifrado;
    }
}
```

```
/* El emisor recibe la clave public del receptor y
envía al receptor mensaje codificado en Base64*/
package clavepublicaemisor;
import java.io.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import java.util.*;
public class ClavePublicaEmisor {
    public static void main(String[] args) throws InvalidKeyException,
        IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException,
        NoSuchPaddingException,                          UnsupportedEncodingException,
InvalidKeySpecException{
        // Instancia de Scanner para introducir clave Publica y mensaje por teclado
        Scanner teclado = new Scanner(System.in);
        System.out.print("Clave Publica: ");
        String clave = teclado.nextLine();
        System.out.print("Mensaje: ");
        String mensajeOriginal = teclado.nextLine();

        // Convertimos clave String en Key
        Key clavePublica = stringToKey(clave);

        // Encriptamos el mensaje en RSA pasandole el mensaje y la clave
        String mensajeCifrado = cifrar(mensajeOriginal, clavePublica);

        // Mostramos el mensaje encriptado RSA en Base64
        System.out.println("Mensaje Encriptado RSA:  " + mensajeCifrado);
    }

    public static Key stringToKey(String clave)
            throws NoSuchAlgorithmException, InvalidKeySpecException {
```

```
        // Decodificamos la clave de Base64
        byte[] claveBytes = Base64.getDecoder().decode(clave);
        //Indicamos que se trata de una clave de algoritmo RSA
        KeyFactory algRSA = KeyFactory.getInstance("RSA");
        // Codificamos la clave en bytes para convertirla en Key
        X509EncodedKeySpec claveBytesCodificada = new X509EncodedKeySpec(claveBytes);
        // Convertimos la clave en bytes codificada a tipo Key
        Key clavePublica = algRSA.generatePublic(claveBytesCodificada);
        // Retornamos la clave publica generada a partir del String
        return clavePublica;
    }

    public static String  cifrar(String paraCifrar, Key claveCifrado)
            throws InvalidKeyException, IllegalBlockSizeException,
            BadPaddingException, NoSuchAlgorithmException,
            NoSuchPaddingException {
        // Creamos una instancia Cipher para cifrar mensaje con algoritmo RSA
        Cipher cifrador = Cipher.getInstance("RSA");
        // Iniciamos la instancia del cifrador en modo cifrado con la clave pública
        cifrador.init(Cipher.ENCRYPT_MODE, claveCifrado);
        // Mediante el método doFinal cifra el mensaje obteniendo el
        // mensaje cifrado en bytes
        byte[] resultado = cifrador.doFinal(paraCifrar.getBytes());
        // Convertimos el mensaje cifrado a String codificado en Base64
        String                          mensajeCifradoCodificado                          =
Base64.getEncoder().encodeToString(resultado);
        // Retornamos el mensaje cifrado
        return mensajeCifradoCodificado;
    }
}
```