1. Desencriptación Simétrica de Fichero

Enunciado:

"Crea un programa que solicite al usuario el nombre del fichero encriptado (por ejemplo, `fichero.cifrado`) y la contraseña utilizada para generarlo. Utilizando el algoritmo **AES** (Rijndael) y la misma lógica de generación de clave basada en la semilla (usuario + password), descifra el contenido y muéstralo por consola. Debes usar `Cipher` en modo `DECRYPT_MODE`."

Código:

Java

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.MessageDigest;
import java.util.Arrays;
import java.util.Scanner;

public class DescifrarFichero {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);

            // 1. Datos de entrada
            System.out.println("Introduce ruta del fichero
cifrado:");
            String ruta = sc.nextLine();
            System.out.println("Introduce tu usuario:");
            String user = sc.nextLine();
            System.out.println("Introduce tu password:");
            String pass = sc.nextLine();

            // 2. Generar la clave (Igual que en tu ejercicio de
cifrado)
            // Usamos Hash para asegurar 128 bits a partir de la
semilla
            byte[] key = (user + pass).getBytes("UTF-8");
```

```
            MessageDigest sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16); // 16 bytes = 128 bits
            SecretKeySpec secretKey = new SecretKeySpec(key, "AES");

            // 3. Configurar Cipher para DESENCRIPTAR [cite: 714]
            Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);

            // 4. Leer fichero, descifrar y mostrar
            byte[] fileContent =
Files.readAllBytes(Paths.get(ruta));
            byte[] decipheredContent = cipher.doFinal(fileContent);

            System.out.println("--- Contenido Descifrado ---");
            System.out.println(new String(decipheredContent));

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

2. Integridad con Resumen de Mensajes (Hashing)

El PDF destaca que los resúmenes (`MessageDigest`) son unidireccionales y sirven para verificar que los datos no han sido alterados o para guardar contraseñas de forma segura.

Enunciado:

"Realiza un programa que simule un login. El programa tendrá una contraseña correcta almacenada en su código, pero **solo su HASH** (SHA-1). El usuario introducirá una contraseña por teclado; el programa calculará su resumen SHA-1 utilizando la clase `MessageDigest` y lo comparará con el almacenado. Si coinciden, acceso permitido."

Código:

Java

```
import java.security.MessageDigest;
import java.util.Scanner;

public class LoginHash {
    // Hash SHA-1 pre-calculado de la palabra "secreto"
    private static final String HASH_STORED =
"e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4";

    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            System.out.print("Introduce la contraseña: ");
            String password = sc.nextLine();

            // 1. Obtener instancia del algoritmo [cite: 638]
            MessageDigest md = MessageDigest.getInstance("SHA-1");

            // 2. Calcular el resumen (bytes) [cite: 639]
            byte[] digest = md.digest(password.getBytes());

            // 3. Convertir bytes a Hexadecimal para comparar
            StringBuilder sb = new StringBuilder();
            for (byte b : digest) {
                sb.append(String.format("%02x", b));
            }
            String inputHash = sb.toString();

            // 4. Verificar
            if (inputHash.equals(HASH_STORED)) {
                System.out.println("Acceso PERMITIDO.");
            } else {
                System.out.println("Acceso DENEGADO.");
            }

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

3. Cifrado Asimétrico (RSA)

Aquí se evalúa la distinción entre clave pública (para cifrar) y privada (para descifrar), gestionadas con `KeyPairGenerator`.

Enunciado:

"Crea un programa que genere un par de claves RSA de 1024 bits. El programa debe cifrar la frase 'Mensaje Secreto' utilizando la **Clave Pública** generada y mostrar los bytes cifrados. Inmediatamente después, debe descifrar esos bytes utilizando la **Clave Privada** y mostrar el mensaje original recuperado."

Código:

Java

```
import javax.crypto.Cipher;
import java.security.KeyPair;
import java.security.KeyPairGenerator;

public class AsimetricoRSA {
    public static void main(String[] args) {
        try {
            // 1. Generar par de claves RSA
            KeyPairGenerator keyGen =
KeyPairGenerator.getInstance("RSA");
            keyGen.initialize(1024);
            KeyPair pair = keyGen.generateKeyPair();

            String mensajeOriginal = "Mensaje Secreto";

            // 2. Cifrar con Clave PÚBLICA [cite: 259]
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, pair.getPublic());
            byte[] mensajeCifrado =
cipher.doFinal(mensajeOriginal.getBytes());

            System.out.println("Cifrado: " + new
String(mensajeCifrado)); // Saldrá basura ilegible

            // 3. Descifrar con Clave PRIVADA [cite: 259]
            cipher.init(Cipher.DECRYPT_MODE, pair.getPrivate());
            byte[] mensajeDescifrado =
cipher.doFinal(mensajeCifrado);
```

```
            System.out.println("Descifrado: " + new
String(mensajeDescifrado));

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

4. Firma Digital

Este ejercicio combina hashing y cifrado asimétrico para garantizar autenticidad. El PDF menciona explícitamente los pasos `initSign`, `update`, `sign` y `verify`.

Enunciado:

"Implementa una clase que firme digitalmente el texto 'Documento Importante' utilizando el algoritmo **DSA**.

1. Genera las claves necesarias.
2. Crea la firma usando la clave privada y muéstrala.
3. Verifica la firma usando la clave pública e imprime si es válida o no."

Código:

Java

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.Signature;

public class FirmaDigital {
    public static void main(String[] args) {
        try {
            // 1. Generar claves DSA (Estándar para firmas) [cite:
669]
            KeyPairGenerator keyGen =
KeyPairGenerator.getInstance("DSA");
            keyGen.initialize(1024);
            KeyPair pair = keyGen.generateKeyPair();
```

```
            String datos = "Documento Importante";

            // --- PROCESO DE FIRMA (Emisor) ---
            Signature dsa = Signature.getInstance("SHA1withDSA");
            dsa.initSign(pair.getPrivate()); // Firmar con Privada
[cite: 662]
            dsa.update(datos.getBytes());    // Cargar datos [cite:
663]
            byte[] firma = dsa.sign();       // Obtener firma [cite:
664]

            System.out.println("Firma generada (hex): " +
bytesToHex(firma));

            // --- PROCESO DE VERIFICACIÓN (Receptor) ---
            Signature dsaVerify =
Signature.getInstance("SHA1withDSA");
            dsaVerify.initVerify(pair.getPublic()); // Verificar con
Pública [cite: 666]
            dsaVerify.update(datos.getBytes());     // Cargar mismos
datos [cite: 667]

            boolean check = dsaVerify.verify(firma); // Validar
[cite: 668]
            System.out.println("¿Firma válida?: " + check);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Método auxiliar para ver la firma bonita
    private static String bytesToHex(byte[] hash) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : hash) hexString.append(String.format("%02x",
b));
        return hexString.toString();
    }
}
```

5. Sockets Seguros (Servidor SSL)

Para este ejercicio, el PDF indica que necesitas un certificado. En el examen, o bien te dan el almacén (`keystore.jks`) o te piden asumir que ya existe.

Requisito previo (Terminal): Debes crear un certificado con `keytool` como indica el PDF: `keytool -genkey -keyalg RSA -alias examen -keystore almacenExamen.jks -storepass 123456`

Enunciado:

"Crea un **Servidor SSL** que escuche en el puerto 5555. El servidor debe cargar el KeyStore llamado `almacenExamen.jks` con contraseña `123456`. Al recibir un cliente, le enviará el mensaje 'Conexión Segura Establecida' y cerrará la conexión."

Código:

Java

```
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import java.io.OutputStream;
import java.io.PrintWriter;

public class ServidorSSL {
    public static void main(String[] args) {
        // Definir propiedades del KeyStore (Obligatorio para el
servidor)
        System.setProperty("javax.net.ssl.keyStore",
"almacenExamen.jks");
        System.setProperty("javax.net.ssl.keyStorePassword",
"123456");

        try {
            // 1. Crear factoría y socket servidor seguro [cite:
821]
            SSLServerSocketFactory sslf = (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();
            SSLServerSocket servidor = (SSLServerSocket)
sslf.createServerSocket(5555);

            System.out.println("Servidor SSL esperando
conexiones...");
```

```
            // 2. Aceptar conexión (Se crea un socket seguro) [cite:
823]
            SSLSocket cliente = (SSLSocket) servidor.accept();

            // 3. Enviar mensaje simple
            OutputStream out = cliente.getOutputStream();
            PrintWriter writer = new PrintWriter(out, true);
            writer.println("Conexión Segura Establecida");

            // Cerrar
            writer.close();
            cliente.close();
            servidor.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class ClienteSSL {
    public static void main(String[] args) {
        // IMPORTANTE: El cliente debe confiar en el certificado del
servidor.
        // En un entorno real usaríamos un TrustStore, aquí usamos
el mismo almacén para simplificar.
        System.setProperty("javax.net.ssl.trustStore",
"almacenExamen.jks");
        System.setProperty("javax.net.ssl.trustStorePassword",
"123456");

        try {
            [cite_start]// 1. Obtener un objeto SSLSocketFactory
[cite: 849]
            SSLSocketFactory factory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
```

```
            [cite_start]// 2. Crear un objeto SSLSocket indicando
servidor y puerto [cite: 850]
            System.out.println("Cliente intentando conectar...");
            SSLSocket socket = (SSLSocket)
factory.createSocket("localhost", 5555);

            [cite_start]// 3. Crear canal seguro de comunicación
(Lectura de datos) [cite: 851]
            BufferedReader entrada = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            // Leer el mensaje que envía el servidor
            String mensajeServidor = entrada.readLine();
            System.out.println("Recibido del servidor: " +
mensajeServidor);

            // Cerrar flujos y socket
            entrada.close();
            socket.close();

        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Error: Asegúrate de que el servidor
está encendido y el almacén existe.");
        }
    }
}
```