

*Implementación de  
un servicio básico  
de autenticación  
con DNle*

***Práctica 3***

*Antonio Delgado Pamos*

*Cristian Maier*

---

# INDICE

<b>INTRODUCCIÓN.....</b>	<b>2</b>
<b>ESTRUCTURA DE LA APLICACIÓN.....</b>	<b>3</b>
Cliente.....	3
Servidor .....	4
Base de Datos .....	5
<b>CIFRADO SHA-1 .....</b>	<b>6</b>
Método Convencional (Sin Cifrado).....	7
Método con Cifrado en Cliente .....	8
Método con Cifrado en Servidor.....	11
Método Cifrado en Cliente / Descifrado en Servidor (Fuerza Bruta).....	13

## INTRODUCCIÓN

Esta práctica consiste en la realización de un programa con el que un usuario que disponga de un DNle pueda autenticarse en un servicio.

Para ello, hemos tenido que implementar una aplicación en Java, la cual, nos permite acceder a los datos del DNle a través del lector de tarjetas.

Una vez obtenidos los datos del usuario, generamos automáticamente un nombre de usuario, que junto con una contraseña pedida por teclado previamente y el NIF, enviaremos al servidor.

Este servidor, se pondrá en contacto con la base de datos y comprobará que el usuario se encuentra registrado. En caso afirmativo, nos devolverá un saludo verificando la autenticidad del usuario, en caso contrario, nos muestra un mensaje de error indicando que el usuario no existe o que algunos de los datos introducidos son incorrectos.

Para la implementación de esta práctica, hemos usado las siguientes herramientas y tecnologías:

- ➔ Eclipse Mars 2.0
- ➔ NetBeans IDE 8.1
- ➔ Dia Diagram Editor (diseño de diagramas de flujos)
- ➔ JDK 1.8 Java
- ➔ Notepad++
- ➔ Procesador de textos Microsoft Word 2013
- ➔ Google Chrome y Mozilla Firefox
- ➔ XAMPP (Servidor Local)
- ➔ PHP Versión 5.6.15
- ➔ MySQL a través del cliente MariaDB
- ➔ GitHub (Herramienta de control de versiones)
- ➔ Lector de tarjetas OMNIKEY 3121
- ➔ DNle 2.0
- ➔ Drivers DNle para Windows 7 y 10.

A continuación, vamos a proceder a explicar cada una de la funciones de nuestra aplicación.

## ESTRUCTURA DE LA APLICACIÓN

Nuestra aplicación se compone de tres partes principales:

- Un Cliente realizado en java.
- Un Servidor implementado con PHP.
- Una Base de Datos implementada en MySQL.

### CLIENTE

Nuestro cliente se compone de dos archivos, llamados: *ObtenerDatos.java* y *Main.java*.

- **ObtenerDatos.java**

En este archivo, venía implementado el método *ConexionTarjeta()*, que es el encargado de establecer la conexión entre el lector de tarjetas y nuestra aplicación.

Además, también venía implementado un método llamado *esDNle()*, que comprueba si la tarjeta introducida en el lector es un DNle o por el contrario, si es otro tipo de tarjeta. En caso de no ser un DNle, nos salta una excepción y no nos deja continuar con la ejecución de la aplicación.

Basándonos en el método original, *leerDeCertificado()*, hemos programado cuatro métodos distintos para obtener el nombre, apellidos y DNI del usuario a través del certificado interno de la tarjeta DNle.

Para la realización de estos métodos, hemos tenido que tener en cuenta la posición en la que se encuentra almacenados dichos datos dentro del certificado.

Estos métodos nos devuelven una cadena de caracteres que utilizaremos a continuación para enviar los datos a nuestro archivo principal (*main.java*) a través de los métodos *LeerAp1()*, *LeerAp2()*, *LeerNombre()* y *LeerDNI()*, que, a su vez, comprobarán si están extrayendo los datos de una tarjeta DNle llamando al método *esDNle()*, explicado anteriormente.

- **Main.java**

En este archivo, creamos un objeto de la clase *ObtenerDatos*, que pone en marcha todo lo explicado anteriormente.

Llamamos a los métodos de lectura para obtener los datos del usuario y los almacenamos en cadenas de caracteres para posteriormente construir el nombre del usuario, que se compone de la primera letra del nombre, primer apellido completo y primera letra del segundo apellido, en nuestro caso es: “**adelgadop**”.

Cabe destacar, que la obtención de datos del certificado, se realiza en mayúscula y por tanto, hemos tenido que hacer uso de un método de la clase *String*, llamado *toLowerCase()*, para convertir la cadena de texto a minúscula y poder componer el `username` correctamente, tal y como se pide en las especificaciones de la práctica.

Realizado este paso, pedimos la contraseña, con la que posteriormente nos autenticamos, al usuario, haciendo uso de la clase *Scanner*.

Para poder enviar el contenido de las variables al servidor, hacemos uso de la clase *URLEncoder*, para poder codificar los datos en UTF-8 (diccionario internacional).

Creamos un cliente *Http* mediante *URLConnection*, que utilizaremos para enviar los datos codificados anteriormente al servidor. Para ello utilizaremos un *StreamWriter*, que concatena todos los datos y los envía al servidor mediante el método `POST`.

Por último, leemos el resultado que nos devuelve el servidor mediante un *BufferedReader*. Este método, nos devuelve la página web entera, pero nosotros, la hemos descartado quedándonos, únicamente con un mensaje de saludo en caso de una correcta autenticación, o un error, en su defecto.

## SERVIDOR

Para la implementación de nuestro servicio hemos utilizado únicamente el archivo ***autentica.php***.

Para instalarlo en el servidor, lo hemos copiado en la carpeta de ***XAMPP/htdocs/dnie***.

En caso de utilizar el servidor WAMP, habría que copiarlo en la carpeta ***www***.

En este archivo, hemos tenido que modificar los parámetros de conexión con la base de datos ya que nosotros tenemos como usuario “*root*” y contraseña “*root*”. Por eso, en caso de que se quiera probar en otro servidor, hay que modificar estos parámetros de acuerdo a la configuración del mismo.

También, hemos realizado modificaciones en la recogida de los parámetros del cliente, añadiendo `$_POST['user']`, para poder recibir el nombre de usuario mediante el método POST desde nuestro cliente.

Por último, hemos realizado cambios para implementar el hash SHA-1. La explicación de nuestro algoritmo de cifrado se explicara más adelante, así como la codificación en base 64 para cada uno de los casos en que hemos realizado el algoritmo.

## BASE DE DATOS

Hemos utilizado la base de datos proporcionada en los recursos aportados para esta práctica.

Para importar la base de datos a nuestro Servidor MySQL, hemos utilizado el comando: ***source + ruta bd.sql***, a través de la consola de Windows, conectándonos a través del cliente MariaDB, que viene incorporado con el servidor XAMPP.

La base está formada por una tabla llamada *users* que tiene cuatro campos:

- ➔ ID: Es un número identificador de tuplas y pertenece a la clave primaria de esta tabla.
- ➔ Username: Es el nombre de usuario que tiene que coincidir con el generado automáticamente por nuestro cliente implementado en Java.
- ➔ Password: Es la contraseña que debe coincidir con la que el usuario introduce por teclado a través del cliente Java. No debe de coincidir exactamente en todos los casos, sino cuando no utilice algoritmo de cifrado, en caso de ir cifrado, debe coincidir con la contraseña cifrada, pero no con la “original” introducida por el usuario. Esto lo explicamos más adelante, cuando hablamos de los algoritmos de Hash y cifrado implementados.
- ➔ DNI: Es el NIF que debe coincidir con el recogido de la tarjeta DNle.

Nosotros, hemos introducido la siguiente tupla para comprobar nuestro servicio:

***Insert into users values (1,'adelgadop','123456','77368093a');***

```

MariaDB [dniauth]> insert into users values(1,'adelgadop','123456','77368093a');
Query OK, 1 row affected (0.00 sec)

MariaDB [dniauth]> select *from users;
+-----+-----+-----+-----+
| id | user      | password | dni      |
+-----+-----+-----+-----+
| 1  | adelgadop | 123456   | 77368093a |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

Aquí adjuntamos una captura de cómo queda nuestra base de datos con la tupla insertada.

Además, vamos a mostrar cómo quedaría con la contraseña cifrada en SHA-1, y codificada en Base64 a la misma vez, ya que hablaremos de esto en las siguientes páginas.

```

MariaDB [dniauth]> insert into users values(1,'adelgadop','YWFmMmNkMmU3YjU5MwZmN2NjMzNjZTczNDg0MmNlZjA0NTczODFhZg==','77368093a');
Query OK, 1 row affected (0.00 sec)

MariaDB [dniauth]> select *from users;
+-----+-----+-----+-----+
| id | user      | password                                     | dni      |
+-----+-----+-----+-----+
| 1  | adelgadop | YWFmMmNkMmU3YjU5MwZmN2NjMzNjZTczNDg0MmNlZjA0NTczODFhZg== | 77368093a |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

## CIFRADO SHA-1

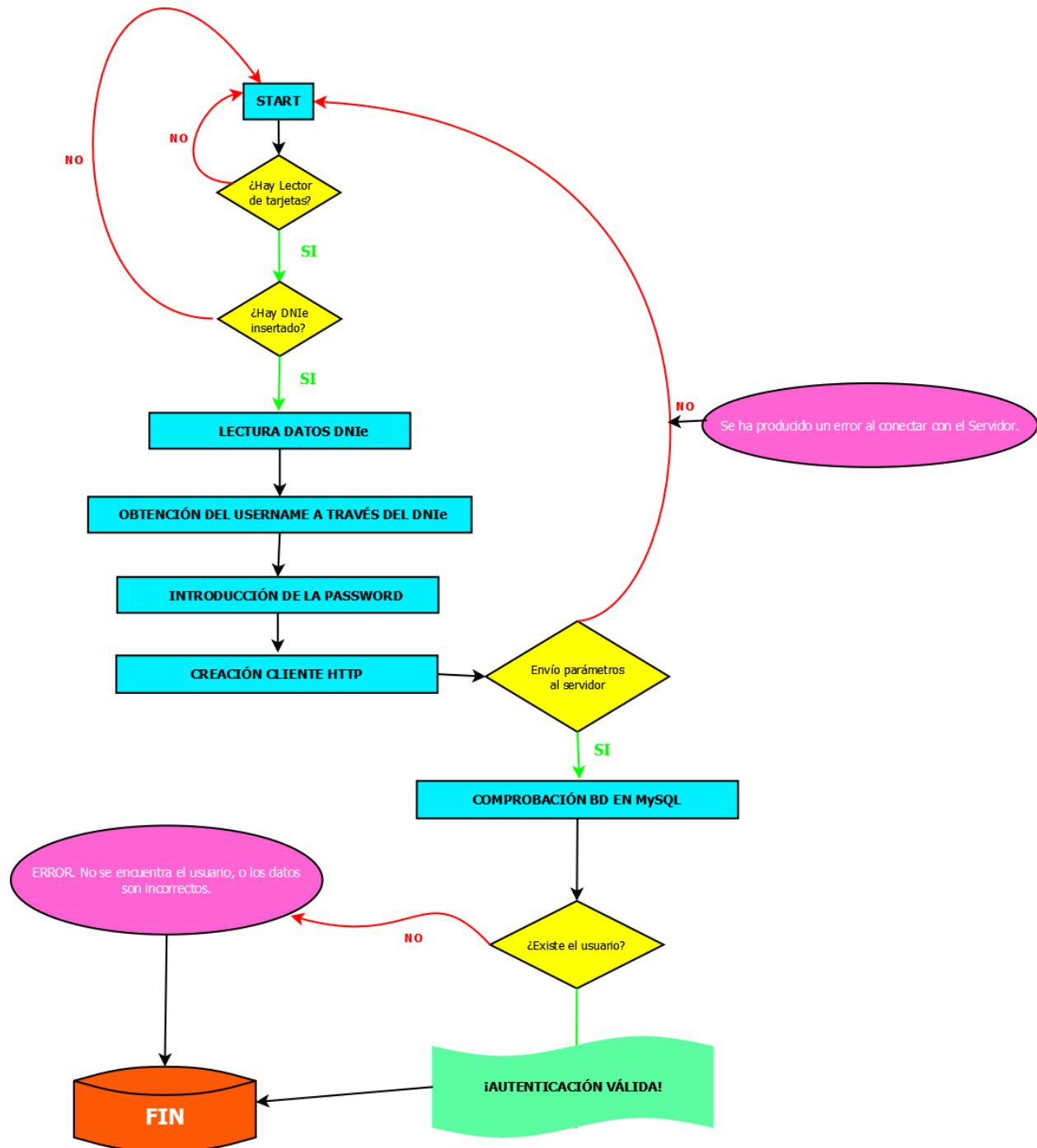
La práctica, requería de un cifrado de la contraseña mediante hash SHA-1. Para realizarlo, hemos utilizado tres formas distintas, además del método convencional sin cifrar que es el explicado anteriormente.

Todos los pasos anteriores nos sirven para realizar el cifrado, por eso, ahora vamos a centrarnos en la parte de la encriptación de la contraseña.

Vamos a realizar la explicación de los tres métodos y además adjuntaremos el diagrama de flujo de cada uno de ellos.

## MÉTODO CONVENCIONAL (SIN CIFRADO)

Debido a que todos los pasos explicados anteriormente pertenecen al método sin cifrado, adjuntamos directamente el diagrama de flujo.





## MÉTODO CON CIFRADO EN CLIENTE

En este método, realizamos el cifrado de la contraseña directamente en el **Main.java (Cliente)**.

Cuando solicitamos por teclado la contraseña al usuario, la ciframos haciendo uso del método que se encuentra en la librería “org.apache.commons” llamado *DigestUtiles.sha1Hex()*, que se encarga de cifrar la contraseña en SHA-1.

Además de este método, nosotros hemos incluido una cadena de texto aleatorio que concatenamos junto con la contraseña cifrada para hacerla más segura y que un usuario externo no pueda descifrarla, haciendo uso de cualquier aplicación creada para este fin.

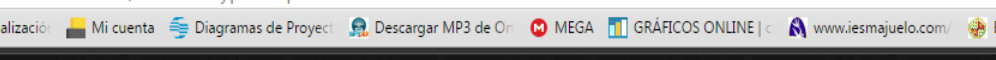
También, tal y como nos indican las especificaciones, hemos tenido que codificarla en Base64. Para ello, hemos hecho uso de la clase **Base64**, llamando al método *encodeToString()*.

Una vez cifrada y codificada la contraseña, la enviamos al servidor de la misma forma que hemos mencionado anteriormente y la comparamos con la almacenada en la base de datos, que tiene que estar cifrada y codificada previamente.

Con respecto a la base de datos original, hemos tenido que modificar la longitud del campo password, ya que solo admitía un máximo de 32 caracteres y nuestra contraseña tiene una longitud de 64, al estar codificada en esta base. Para evitar problemas con futuras contraseñas, hemos asignado una longitud de 100 caracteres, pudiendo así, aprovechar la Base de Datos para otro tipo de codificaciones y cifrados.

Este método, tiene la **ventaja** de que al estar la contraseña cifrada y codificada en la base de datos, en caso de que nos atacasen mediante inyección SQL, obtendrían la contraseña cifrada directamente y al tener implementado el sha1 y nuestra propia cadena de caracteres no podrán averiguar la contraseña original.

A continuación, hacemos un inciso para exponer un ejemplo de aplicación diseñada para desvelar contraseñas cifradas en sha1: <https://hashkiller.co.uk/sha1-decrypter.aspx>



https://hashkiller.co.uk/sha1-decrypter.aspx

Actualización Mi cuenta Diagramas de Proyecto Descargar MP3 de On MEGA GRÁFICOS ONLINE Espacios Virtuales UJA

Status: We found 1 hashes! [Timer: 90 m/s] Please find them below...

SHA1 Hashes:

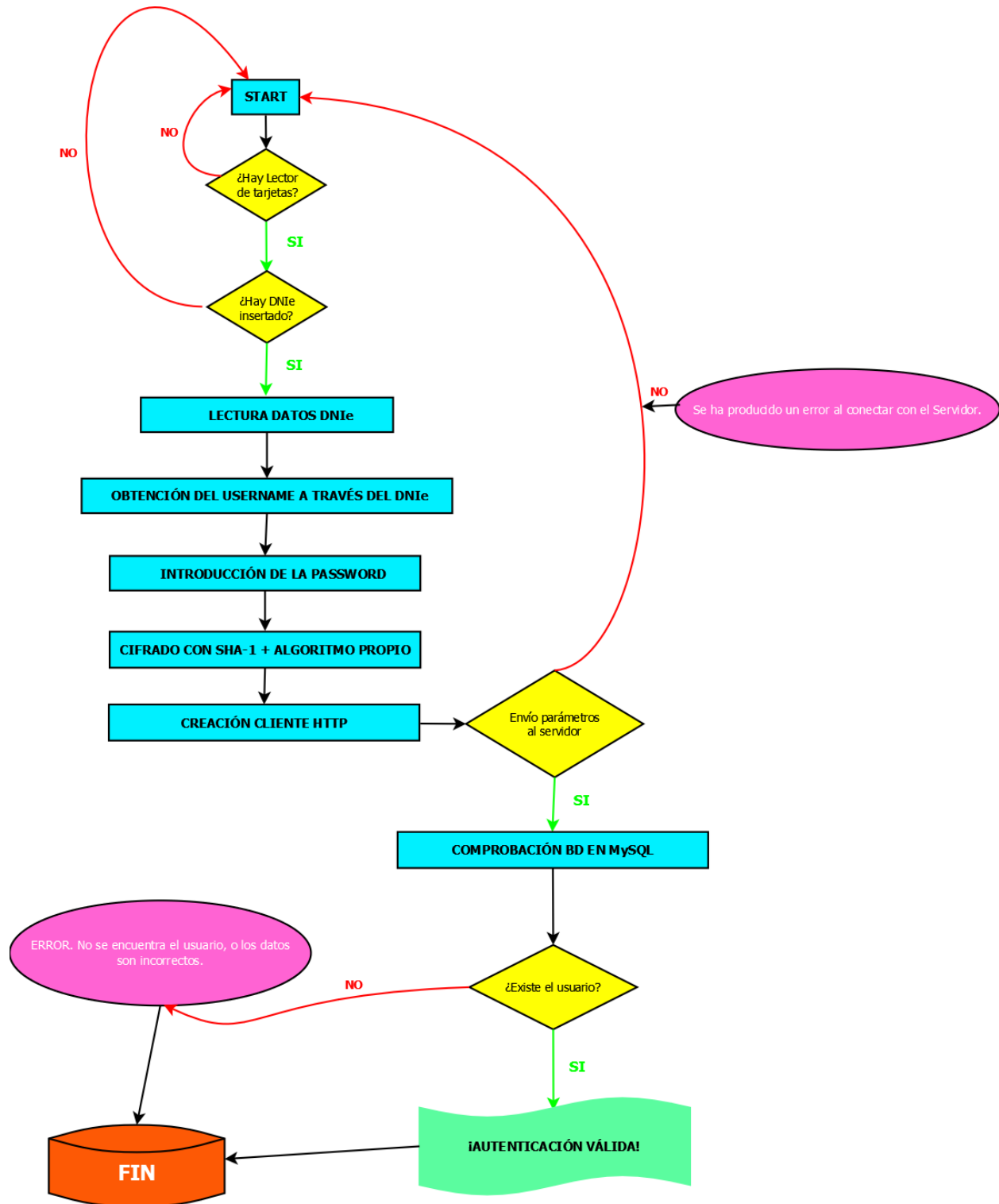
Max: 64

Please use a standard list format

7c4a8d09ca3762af61e59520943dc26494f8941b

7c4a8d09ca3762af61e59520943dc26494f8941b SHA1 : 123456

Adjuntamos el diagrama de flujo para este método:



## MÉTODO CON CIFRADO EN SERVIDOR

En este método, el hash y la codificación, la realizamos de la misma forma que en el paso anterior, es decir, utilizamos el cifrado sha1 concatenándolo con una cadena de caracteres aleatorios y además codificamos en base64, tal y como nos indican las especificaciones.

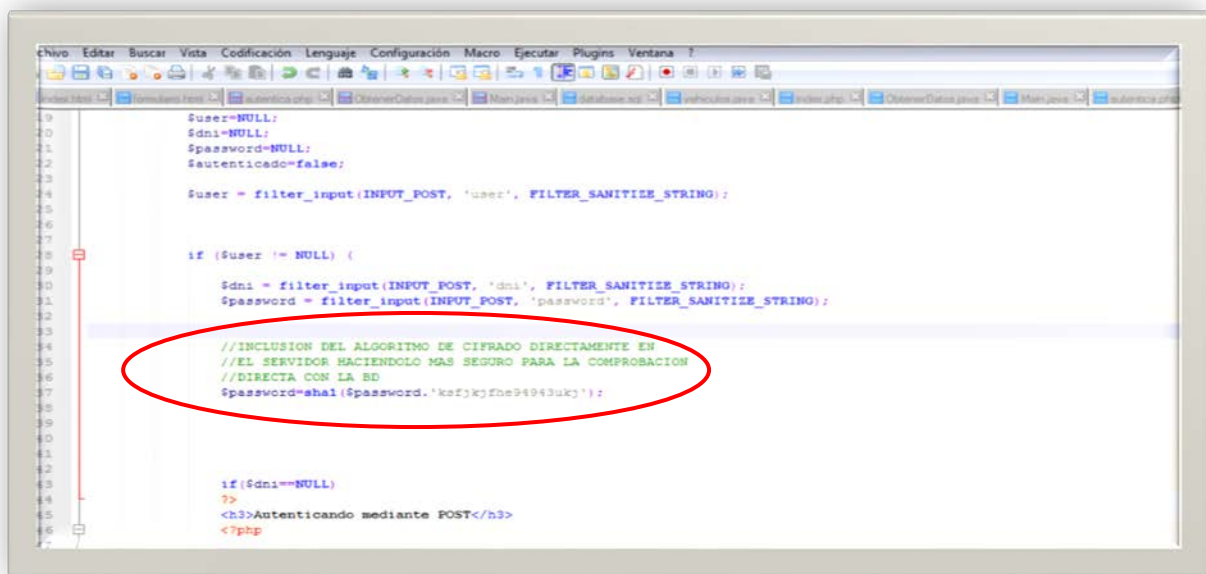
La diferencia con el método anterior, es que en este caso, la contraseña viaja desde el cliente hacia el servidor sin cifrar; Una vez llega a nuestro servidor, realizamos el cifrado con la instrucción ***sha1(\$password+\$texto\_aleatorio)***, así como la codificación en base64 de la misma con el comando ***base64\_encode(\$password)***.

Este paso lo realizamos en nuestro archivo *autentica\_Mac.php*.

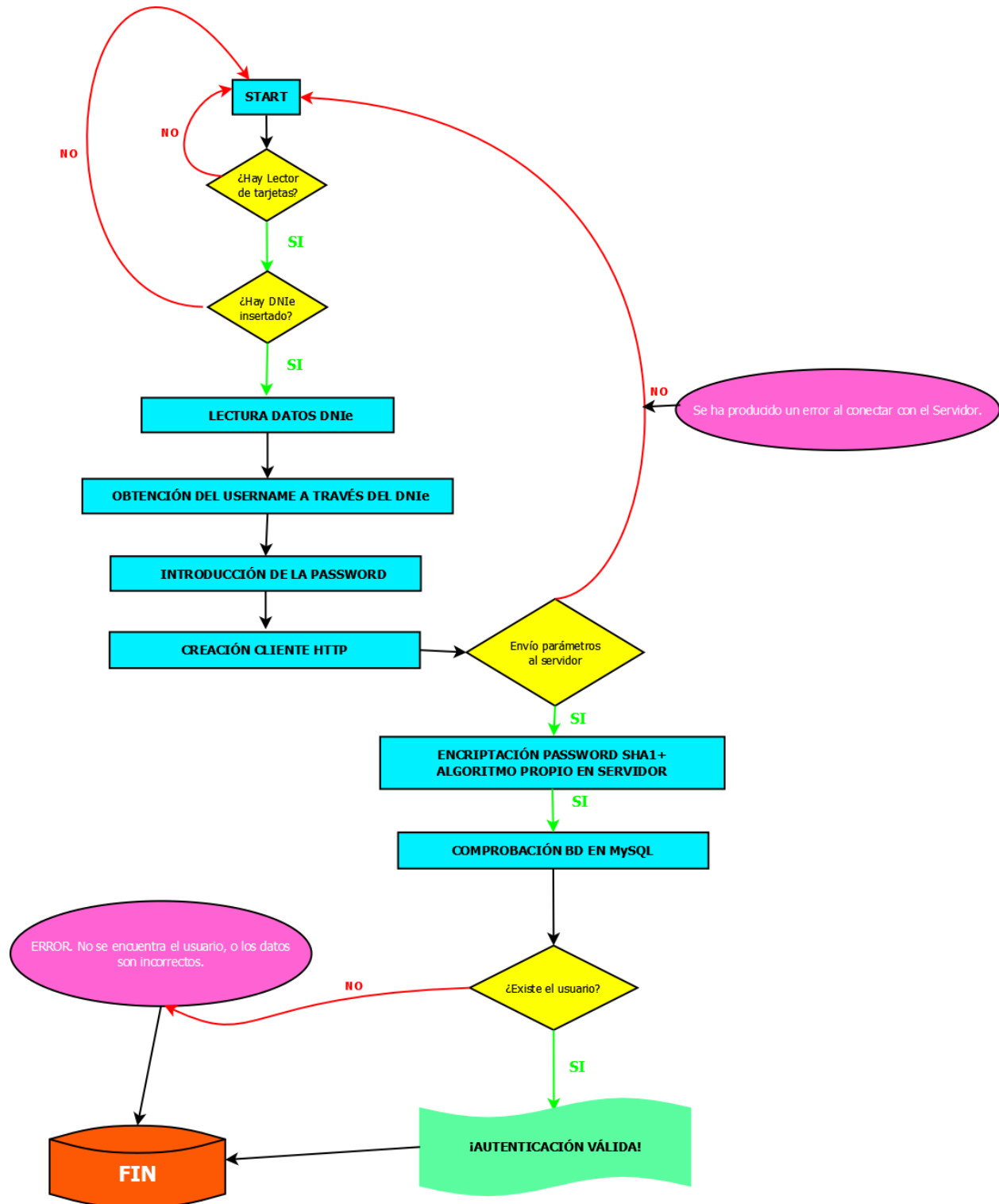
Una vez aquí, se comprueba la contraseña con la base de datos en la que también se encuentra cifrada, y, para ello, la base de datos utilizada es la misma que en el método anterior, es decir, hemos tenido que ampliar la longitud del campo password.

Este diseño Mac, tiene la **ventaja** de que el cifrado se realiza en el servidor, y, de esta manera, nadie puede acceder al archivo *autentica\_Mac.php*, por eso, no podrán conocer nuestra cadena de caracteres aleatoria, que concatenamos con nuestra contraseña cifrada en sha1.

Por otra parte, tiene la **desventaja** de que la contraseña viaja sin cifrar hasta el servidor y por tanto, puede ser interceptada en el trayecto.



Adjuntamos diagrama de flujo:



## MÉTODO CIFRADO EN CLIENTE / DESCIFRADO EN SERVIDOR (FUERZA BRUTA)

En este método, realizamos el cifrado de la contraseña en el cliente al igual que lo hacíamos en el método “CIFRADO EN CLIENTE” que hemos explicado anteriormente.

Una vez llega la contraseña cifrada y codificada al servidor, realizamos el descifrado y el decodificado de la misma.

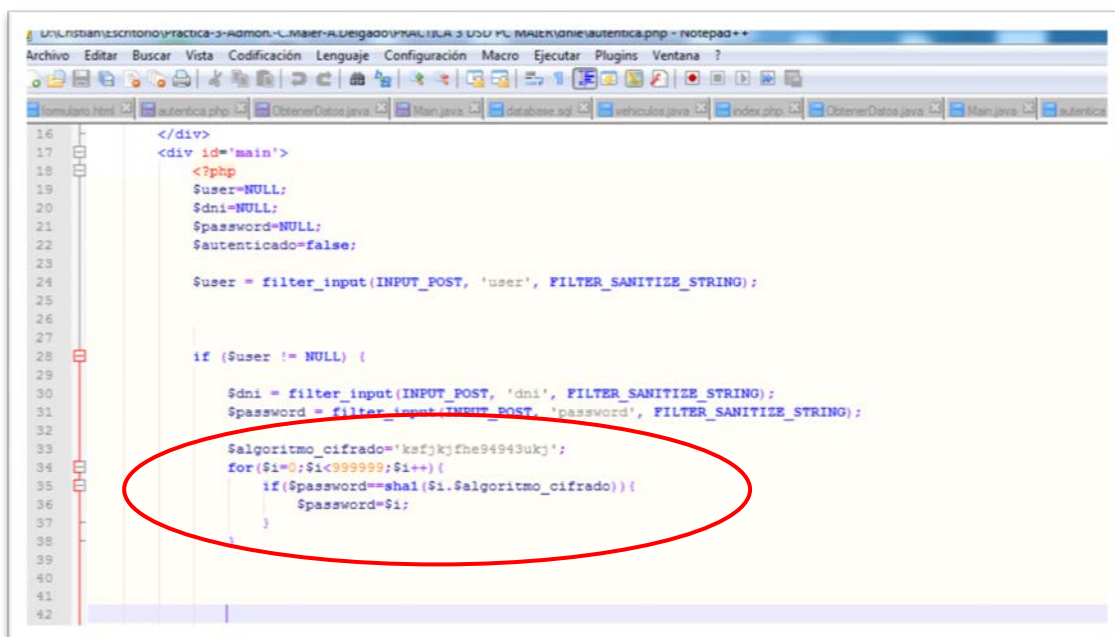
Para realizar la decodificación, utilizamos la instrucción ***base64\_decode(\$password)*** y una vez obtenida, procedemos al descifrado del SHA-1, a través de un bucle, que nos recorre un “diccionario numérico” desde el 0 hasta el 999999 (Nuestra password “original” es de 6 caracteres numéricos).

Si mediante el recorrido del bucle, la contraseña enviada por el cliente, concatenada con nuestra cadena aleatoria, coincide con algún número mencionado anteriormente cifrado con la cadena aleatoria, la contraseña original será el número coincidente.

Una vez descifrada la contraseña, la comparamos con la almacenada en nuestra base de datos, que en este caso, almacenará la contraseña sin cifrar.

Como **ventaja destacable** de este método, estamos incluyendo un método privado de descifrado y decodificado, que se almacena en el servidor y por lo tanto, nadie puede acceder a él.

También tiene una **desventaja** muy clara, y es que, almacenamos la contraseña en la base de datos sin cifrar, y, queda expuesta a cualquier ataque externo.



```
16 </div>
17 <div id='main'>
18 <?php
19 $user=NULL;
20 $dni=NULL;
21 $password=NULL;
22 $autenticado=false;
23
24 $user = filter_input(INPUT_POST, 'user', FILTER_SANITIZE_STRING);
25
26
27
28 if ($user != NULL) {
29
30     $dni = filter_input(INPUT_POST, 'dni', FILTER_SANITIZE_STRING);
31     $password = filter_input(INPUT_POST, 'password', FILTER_SANITIZE_STRING);
32
33     $algoritmo_cifrado='ksfj;kjThe94943ukj';
34     for($i=0;$i<999999;$i++){
35         if($password==sha1($i.$algoritmo_cifrado)){
36             $password=$i;
37         }
38     }
39
40
41
42
```



Adjuntamos el diagrama de flujo de este método:

