***PRÁCTICA: CRIPTOGRAFÍA Y SEGURIDAD INFORMÁTICA***

**ENTREGABLE 1**

**Grado en Ingeniería en Informática, Grupo 82, 07/10/2023**

Realizado por el grupo de prácticas 8, formado por:

* Raúl Sanz Belmar, 100472372@alumnos.uc3m.es
* Mario Rodríguez Román, 100472194@alumnos.uc3m.es

Enlace repositorio GitHub: <https://github.com/Rawwl03/Criptografia_Uc3m>

Índice de Contenidos

[**PROPÓSITO DE LA APLICACIÓN** 3](#_Toc149264504)

[**DISEÑO** 3](#_Toc149264505)

[**MEJORAS IMPLEMENTADAS** 4](#_Toc149264506)

[**AUTENTICACIÓN DE USUARIOS** 5](#_Toc149264507)

[**CIFRADO SIMÉTRICO Y ASIMÉTRICO** 6](#_Toc149264510)

[**AUTENTICACIÓN DE MENSAJES: MAC** 7](#_Toc149264511)

# **PROPÓSITO DE LA APLICACIÓN**

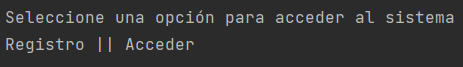
El proyecto va a estar basado en la creación de una aplicación de servicios que te permita comprar entradas para el cine y gestionar tus entradas asociadas a tu cuenta personal. Además, se podrán gestionar tarjetas asociadas a tu usuario para realizar los pagos, y cambiar la contraseña desde dentro de la aplicación. Todo esto contendrá una seguridad frente ataques proporcionada por la encriptación, de manera que vamos a guardar datos como contraseñas y tarjetas en una base de datos, pero estos estarán cifrados (y autenticados en el caso de las tarjetas) gracias a la librería cryptography. El diseño de la aplicación está comprendido por una serie de menús, cada uno con unas funcionalidades, que se manejan a través de la terminal (escribiendo inputs) y se puede navegar entre ellos. Aparte de la aplicación, se han añadido mejoras opcionales que se comentarán más adelante en profundidad, como implementación de base de datos y rotación de claves.

## **DISEÑO**

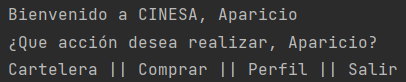
Como se ha mencionado antes, el diseño de la aplicación se basa en una serie de menús que están enlazados entre sí. Todos estos menús se generan en la clase Terminal, que representa a la terminal donde se va a mostrar todo, y se va a utilizar una variable global llamada ‘contrasena\_sys’, que almacenará la contraseña en claro del usuario actual.

Los menús principales que podemos encontrar son los siguientes:

1. **Menú acceso al sistema** -> este menú se trata del menú inicial que se muestra al iniciar la aplicación. Se pueden realizar dos cosas: registrarse o acceder al sistema con un usuario existente. La funcionalidad está recogida en sys\_inicio() {clase Terminal, línea 27}, y una vez accedido al sistema correctamente, irá al menú inicio del cine.



1. **Menú inicio del cine** -> desde aquí podemos realizar 4 acciones: cartelera (podemos ver las películas disponibles y seleccionar una de ellas para ver más información), comprar (nos lleva al proceso de compra, que nos hace seleccionar película, seleccionar un asiento disponible y seleccionar una tarjeta válida para proceder al pago), perfil (nos lleva al menú perfil) y salir del sistema. Corresponde a la función accion\_cine(user\_accedido) {clase Terminal, línea 159}, que recibe el usuario que accedido al sistema, y proporciona las funcionalidades descritas anteriormente.



1. **Menú perfil** -> las funcionalidades que recoge este menú van asociadas a la interacción con la información de un perfil, que en este caso puede ser guardar una tarjeta, borrar una tarjeta, cambiar la contraseña del usuario, ver las entradas compradas, ver las tarjetas guardadas del usuario y salir (volver al menú inicio del cine). La función que se encarga de realizar las funcionalidades de este menú es acc\_perfil(user\_accedido) {clase Terminal, línea 325}, que también necesita el user\_accedido.



Por lo tanto, estos son los 3 menús principales de la aplicación, el primero por ser el menú que sirve de entrada a la aplicación, y los otros 2 porque después de realizar cada acción se va a regresar a uno de ellos para seguir con la ejecución. Cada acción de las expresadas en cada menú va a llevar a cabo la creación de un menú de selección o de visualización, dependiendo de la acción escogida. Además, cada acción solicitada por input o valor dado tiene que ser válido, o de lo contrario el sistema no avanzará hasta que se haya introducido un input correcto.

## **MEJORAS IMPLEMENTADAS**

Entre todas las mejoras opcionales que se proponían en el enunciado, nosotros hemos decidido implementar 2 de ellas. Estas son las siguientes:

* **Implementación de base de datos:** para guardar los datos, hemos usado sqlite3 en Python, que nos permite crear bases de datos y gestionarlas para añadir los datos que querramos. Para ver la información de la base de datos, hemos usado DB Browser. La base de datos se corresponde con la clase Database, que contiene todas las acciones con querys SQL para realizar funcionalidades en cada caso. De esta manera, podemos tener los datos mejor organizados y con un formato adecuado, además de poder acceder fácilmente a nuestros datos mediante sentencias SQL.
* **Rotación de claves:** se ha decidido implementar la rotación de claves de dos maneras. La primera consiste en que cada vez que se realiza un acceso, se va a generar el cifrado de la contraseña con un salt diferente, de manera que el cifrado va cambiando (función rotación\_claves(user\_accedido, False, None) y así solo cambia el salt). La segunda manera consiste en la funcionalidad del menú perfil “cambiar contraseña”, que pedirá una nueva contraseña y generará un hash nuevo con un salt nuevo (función rotación\_claves(user\_accedido, True, contraseña\_antigua)). En estos dos casos, en rotación\_claves() {clase Terminal, línea 574}, se tendrán que descifrar las tarjetas con la clave antigua y cifrarlas de nuevo con la nueva clave generada, para que así los datos se conserven correctamente.

# **AUTENTICACIÓN DE USUARIOS**

Dentro de nuestra aplicación, lo primero que se ejecuta es la autenticación de usuarios registrados o el registro de usuarios válidos, que es donde entra el proceso de autenticación de usuarios en el sistema. En este proyecto hemos escogido la opción de autenticar a los usuarios mediante algo que sabemos, en este caso, contraseñas secretas que elegirá el mismo usuario. Estas contraseñas serán robustas, ya que deben de cumplir las siguientes características:

* Longitud mínima de 10 caracteres
* Mínimo una mayúscula
* Mínimo un número

Para que se produzca esta autenticación, primero el usuario deberá de registrarse, y el nombre de usuario con la contraseña encriptada del usuario y el salt utilizado en ese proceso se guardará en la base de datos. Para ello, necesitamos utilizar una función de derivación de claves segura. Entre todos los algoritmos que existen de cifrado de contraseñas, hemos utilizado el siguiente con las siguientes características:

**PBKDF2HMAC** con algoritmo **hashes.sha256()**

Longitud 32

Uso de salt

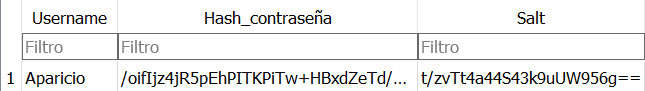
Iteraciones 30000

***CÓDIGO UTILIZADO:***

kdf = PBKDF2HMAC(algorithm=hashes.SHA256(), length=32, salt=salt, iterations=30000)

# Este algoritmo ha sido elegido debido a que para guardar claves en una base de datos, es una función que nos sirve de ayuda, y los parámetros de este algoritmo eran los que queríamos para poder elegir la salida deseada de la clave.

# En kdf tenemos nuestro algoritmo que vamos a utilizar para encriptar las contraseñas introducidas (con la función kdf.derive(clave)), y para la verificación de que una contraseña introducida por input es la misma que la que está guardada (con la función kdf.verify(hash, clave)). En la siguiente imagen, podemos ver el proceso realizado de haber guardado la información de acceso de un usuario en nuestra base de datos:



De esta forma, guardamos en la base de datos lo necesario para poder realizar la autenticación de usuarios sin que comprometa a la seguridad de la aplicación, ya que la contraseña secreta no se puede obtener mediante el conocimiento del hash.

Para derivar la clave, usamos la función *encriptar\_clave()* {clase Terminal, línea 138}, que tiene como parámetros la clave introducida, un bool indicando si la clave es para guardarla posteriormente o es para comprobar, y en el caso de que new = False también se necesitará introducir el salt de la contraseña utilizada en el proceso de registro respectivo. La ejecución de la función genera el algoritmo mencionado anteriormente (en kdf) y luego encripta la clave, y devuelve el hash junto al salt.

En cuanto al proceso de comprobación de claves y autenticación, se usa la función *validate\_contrs()* {clase Terminal, línea 148}, que recibe el hash de la contraseña, la clave introducida en el input y el salt del hash recibido. Esta función se llama en el proceso de acceso, para autenticar la clave introducida. Para ello, se genera el mismo algoritmo con el salt respectivo del hash, y se verifica que el resultado de encriptar la clave con ese salt es el hash. La función devuelve True o False, dependiendo de si son iguales o no.

Además de estos procesos con este algoritmo, también se va a realizar derivación de claves para cifrar las tarjetas, es decir, la contraseña secreta del usuario va a ser derivada con otro salt distinto para generar una clave que se usará como input de un algoritmo de cifrado simétrico (AESGCM en nuestro caso).

# **CIFRADO SIMÉTRICO Y ASIMÉTRICO**

En cuanto al cifrado simétrico y asimétrico, en nuestra aplicación hemos optado por el cifrado simétrico, ya que se adapta mejor a nuestra aplicación y el objetivo de uso del cifrado simétrico. La idea es que en la aplicación se tengan que tener tarjetas guardadas asignadas al usuario para poder proceder al pago, por lo que es importante guardar las tarjetas en la base de datos cifradas para cuando se vaya a proceder a la selección de tarjeta y al pago, se obtenga la tarjeta descifrándola con la misma clave con la que fue cifrada. En concreto, lo que se va a cifrar son los datos de la tarjeta, es decir, el número de la tarjeta, la fecha de caducidad y el cvv. Estarán unidos en un string separados por “-“, y todo eso conforma lo que se llama ‘datos’, y esto se va a guardar junto al salt utilizado (para generar esta clave con PBKDF2HMAC), el nonce usado (único para esta tarjeta), el username y el saldo de la tarjeta (inicialmente es 30). Cada parte de los datos tiene unos requisitos para ser un dato válido, y son los siguientes:

* El número de la tarjeta debe de estar conformado por dígitos, y tiene que tener una longitud de 16 dígitos.
* La fecha de caducidad debe de tener el formato mes/año, validando mes como números desde 01 hasta 12, y año desde 2024 hasta 2031 (longitud obligatoria de 7)
* El cvv debe estar formado por 3 dígitos.

Para realizar todo esto, hemos usado la siguiente función de cifrado simétrico:

**AESGCM: cifrado simétrico autenticado**

Necesita **key** derivada a partir de la clave secreta

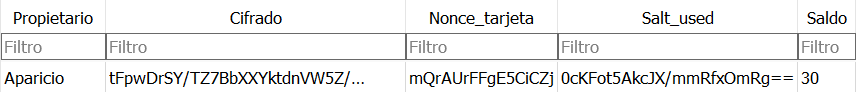
Uso de nonce

***CÓDIGO UTILIZADO:***

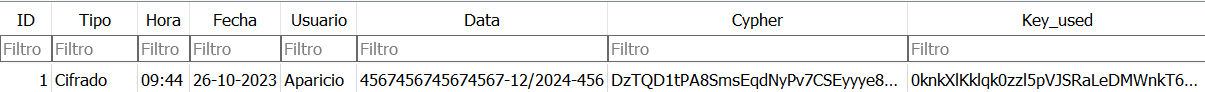
aes = AESGCM(key)

Hemos decidido usar AESGCM porque para cifrado simétrico necesitábamos un algoritmo seguro como AES, además que AESGCM contiene autenticación MAC (se va a profundizar en el apartado de MAC), lo que nos supone hacer cifrado autenticado directamente.

A partir de la función generada, podemos mediante aes.encrypt(nonce, datos, aad) y aes.decrypt(nonce, datos, aad) cifrar y descifrar simétricamente los datos de las tarjetas y así poder usarlos. El nonce representa a un número pseudoaleatorio único tanto para cifrar la tarjeta como para descifrarla. En la siguiente imagen se puede observar una tarjeta que ha sido cifrada y guardada en la base de datos:



Para el cifrado de los datos de la tarjeta, usamos la función cifrado\_tarjeta(user\_accedido) {clase Terminal, línea 431}. Esta función necesita como parámetro el user que va a cifrar una tarjeta para guardarla en la base de datos para indicar que esa tarjeta es suya. Dentro de la función, obtendremos los datos de la tarjeta a guardar, generaremos una clave derivada de la contraseña del user (derivación de contrasena\_sys, como habíamos mencionado en el apartado anterior, mediante PBKDF2HMAC), y luego cifraremos los datos de la tarjeta introduciendo el nonce generado, los datos, y aad = None (no queremos guardar información sin cifrar). Esto nos generará el cifrado de los datos de la tarjeta. Lo último que hace esta función es guardar en la base de datos la tarjeta con sus respectivos datos, y un log de que se ha completado un proceso de cifrado, que en la base de datos se ve de la siguiente manera:



Por otra parte, en el descifrado de los datos de la tarjeta se usa la función descifrar\_tarj(tarj\_guardada, selected\_key){clase Terminal, línea 447}, siendo tarj\_guardada una tarjeta de la base de datos y selected\_key un boolean utilizado para indicar si se está produciendo rotación de claves de salt o de contraseña en claro. Lo que realiza esta función es lo siguiente: primero obtiene el nonce utilizado en el cifrado de la tarjeta junto al criptograma de la tarjeta; luego dependiendo del uso que le queramos dar (con selected\_key) se generará una clave derivada nueva o no, que se utilizará para aplicar la función AESGCM; y por último, se descifrará la contraseña, obteniendo los datos originales de la tarjeta y haciendo return de estos, y, además, en la base de datos se guarda un log especificando que se ha realizado un proceso de descifrado. Este log es igual que el anterior mostrado, pero de tipo “Descifrado”.

# **AUTENTICACIÓN DE MENSAJES: MAC**

Respecto a autenticación de mensajes MAC, en nuestra aplicación se ha desarrollado con el objetivo de cifrar(o descifrar) y autenticar que el criptograma es correcto. Cómo se ha mencionado antes en el apartado de cifrado simétrico, el MAC se realiza a la vez que el cifrado simétrico, pues AESGCM proporciona cifrado simétrico autenticado a la vez. De esta manera, se consigue cifrar y descifrar los datos de la tarjeta y, además, se comprueba que son datos autenticados. El funcionamiento es el mismo que el cifrado simétrico, ya que dicho funcionamiento también se corresponde con el uso de AESGCM.