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

**ENTREGABLE 2**

**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](#_Toc152103624)

[**DISEÑO** 3](#_Toc152103625)

[**MEJORAS IMPLEMENTADAS** 4](#_Toc152103626)

[**FIRMA DIGITAL** 5](#_Toc152103627)

[**CERTIFICADOS DE CLAVE PÚBLICA** 5](#_Toc152103628)

[**COMPLEJIDAD DE LA APLICACIÓN** 5](#_Toc152103629)

# **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. Las funcionalidades principales que se pueden realizar en el sistema son las siguientes:

* Modo sistema o admin para la gestión de entradas, usuarios y peticiones almacenadas.
* Gestión de tarjetas (añadir o eliminar) para ser usadas en los pagos por parte del usuario.
* Adquisición de entradas mediante un sistema de peticiones, que tendrán que ser aceptadas por un Admin o el sistema.
* Cambiar tu contraseña en cualquier momento.
* Gestión de roles, con distintas opciones a realizar en algunos menús. Tenemos los roles “A” (Administrador) y “U” (Usuario), y por encima de estos está el sistema. Cada usuario al registrarse comienza con rol “U”, que podrá ser cambiado por alguien de rol superior.

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. Además, a la hora de realizar peticiones, estas irán firmadas y serán verificadas, al igual que en el caso de que el sistema le envíe una entrada que ha comprado el usuario respectivo (lo veremos más adelante). Junto a todo esto, se ha desplegado una PKI, que será comentada en el diseño, consiguiendo así autenticidad a la hora de realizar acciones en el sistema. Todos los procesos de cifrado y descifrado, y de firma y verificación de datos se pueden observar en sus respectivos logs: Log\_cifrado\_sim y log\_firma.

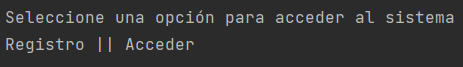
El diseño de la aplicación está comprendido por una serie de menús, cada uno con unas funcionalidades, que se manejan a través de la terminal (escribiendo inputs) y se puede navegar entre ellos. Aparte de la aplicación, se han añadido mejoras opcionales que se comentarán más adelante en profundidad, como implementación de base de datos, rotación de claves y acceso a un modo de sistema mediante reconocimiento facial.

## **DISEÑO**

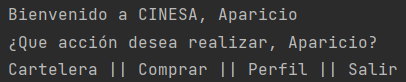
Como se ha mencionado antes, el diseño de la aplicación se basa en una serie de menús que están enlazados entre sí. Todos estos menús se generan en la clase Terminal, que representa a la terminal donde se va a mostrar todo, y se va a utilizar una variable global llamada ‘contrasena\_sys’, que almacenará la contraseña en claro del usuario actual. Como dato a mencionar, se crean datos para un usuario llamado “Sistema”, que no es almacenado en la tabla usuarios, pero si tiene claves asimétricas para poder realizar operaciones de gestión de datos, ya que es considerado el modo sistema que describiremos a continuación

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 tres cosas: registrarse, acceder al sistema con un usuario existente o acceder al menú sistema para gestionar datos mediante reconocimiento facial. La funcionalidad está recogida en sys\_inicio() {clase Terminal, línea 27}, y una vez accedido al sistema correctamente, irá al menú inicio del cine. Si el acceso al menú del sistema es permitido, entonces irá al menú de gestión del sistema.



1. **Menú inicio del cine** -> desde aquí podemos realizar 5 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, donde se enviará una petición con la entrada que se desea comprar), perfil (nos lleva al menú perfil), peticiones(nos lleva al menú de hacer petición) 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, ver las peticiones realizadas por el usuario y salir (volver al menú inicio del cine); y en el caso de que el usuario tenga rol “A”, se verá una opción adicional llamada admin, que permite acceder al menú gestión del sistema. 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.



1. **Menú hacer petición** -> en este menú se nos ofrecen dos opciones sobre que petición quieres realizar. En el caso de que el usuario accedido tenga rol “U”: hacer petición de rol (de “U” a “A”) o hacer petición de devolución de entrada, que en el caso de que no tenga ninguna entrada comprada, mostrará un mensaje de que el user no ha comprado ninguna entrada todavía. Si el usuario tiene rol “A”, este menú no se mostrará, porque irá directamente a hacer la petición de devolución de entrada, ya que no puede pedir petición de rol. La función que actúa como este menú es hacer\_peticion(user\_accedido){clase Terminal, línea 1144}, y necesita el user\_accedido para saber que usuario realiza la petición.
2. **Menú gestión del sistema** -> al acceder a este menú, se nos mostrarán las siguientes opciones que nos permiten gestionar una serie de datos específica: ‘Usuarios’, ‘Peticiones’ y ‘Entradas’. En cada uno de ellos, podemos ver todo lo que haya en el sistema registrado respecto a la opción elegida, y también podemos eliminar datos del sistema según se quiera. Como restricción para eliminar usuarios, se ha establecido que un usuario con rol “A” no puede eliminar otros usuarios con rol “A”, pero sí que pueden ser eliminados al entrar en este menú por el acceso al sistema. Todo esto está recogido en la función menú\_sistema(user\_accedido) {clase Terminal, línea 770}, que necesita saber el user\_accedido, por si se ha accedido en modo “Sistema”, o simplemente ha accedido un administrador.

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:

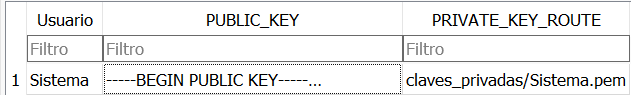
* **Implementación de base de datos:** para guardar los datos, hemos usado sqlite3 en Python, que nos permite crear bases de datos y gestionarlas para añadir los datos que querramos. Para ver la información de la base de datos, hemos usado DB Browser. La base de datos se corresponde con la clase Database, 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.

# **FIRMA DIGITAL**

La firma digital nos va a aportar dentro de nuestra aplicación autenticidad respecto a un mensaje. De esta manera, conseguimos verificar que un mensaje proviene de una persona concreta correctamente. Para realizar todo esto, primero hemos tenido que decidir dos cosas importantes: cómo gestionar las claves asimétricas que se utilizarán en este proceso, y cómo implementar la firma de manera que tenga utilidad. Como la firma y verificación va a ser transparente al usuario, hemos decidido implementar un log para este proceso, que nos proporcione información útil cuando se produzca firma o verificación. Además, es necesario decidir el algoritmo que se va a usar. En nuestro caso, el algoritmo usado es **RSA**, en el que se necesitará una clave privada y pública por cada usuario para que cada uno pueda firmar (con clave privada del emisor) y verificar (con clave pública del emisor) en su respectivo momento. A continuación, se van a explicar ambas cosas detalladamente.

* Gestión de claves asimétricas:

Por cada usuario que acceda por primera vez al sistema se creará un par de claves asimétricas (pública y privada) que se guardarán de la siguiente manera: en la base de datos, guardaremos la clave pública codificada en PEM y la ruta del fichero .pem donde estará la clave privada encriptada. A su vez, se creará un par de claves para un usuario llamado “Sistema”, que corresponde con el acceso en modo System. A continuación, vamos a mostrar fotos sobre cómo se ven estas claves en nuestra aplicación.



La función en la que se crean las claves se llama generar\_asymethric\_keys(user\_accedido) {clase Terminal, línea 1268}, que se invoca en el primer acceso de un usuario al sistema (en el futuro se ha cambiado la generación de una clave pública por un certificado de clave pública). Esta función genera mediante rsa una clave privada de longitud 2048 y con exponente público 65537 (valor recomendado). Esta clave es encriptada con la contraseña del usuario correspondiente para que sea privada suya, y es guardada en la ruta “claves\_privadas/”+<nombre\_user>+”.pem”. En cuanto al certificado de clave pública, que nos va a proporcionar la clave pública del usuario, se crea uno para el usuario que es firmado por la AC (“Sistema”), con una validez de 30 días. Este cambio se ve de la siguiente manera en la base de datos:

El formato de la firma se hablará más adelante, pues es el mismo para todos.

* Uso de firma digital:

Dentro de nuestra aplicación, hemos diseñado 4 usos de firma y verificación que se van a detallar a continuación. Para estas dos acciones, hemos creado una función llamada firmar\_datos(datos, user\_accedido) {clase Terminal, línea 1325} y otra función llamada verificacion\_firma(datos, firma\_bin, user\_firmante, user\_accedido) {clase Terminal, línea 1343}. El proceso de cada una de estas funciones es el siguiente:

* Función firmar\_datos(datos, user\_accedido): esta función recibe unos datos de un user que es el que ha accedido al sistema y devuelve la firma obtenida. “Datos” es un str que es lo que se va a firmar, y el user\_accedido corresponde con el usuario que va a firmar esos datos. El procedimiento que sigue esta función es el siguiente: coge las claves asimétricas de la base de datos del user\_accedido, carga la kv en formato correcto, obtiene la firma mediante la kv usando el algoritmo hash: SHA256, la codifica en base64 para poder leerla y guardarla, añade un log en el Log\_Firma (con información que vamos a ver en una foto a continuación) y devuelve la firma ya codificada.
* Función verificacion\_firma(datos, firma\_bin, user\_accedido, user\_firmante): esta función recibe una firma y el user que ha la ha firmado y comprueba que la firma es correcta. Si no es correcta, se obtiene una excepción de tipo **InvalidSignature**. “Datos” son los mismos datos que en la función anterior, y lo mismo pasa con user\_accedido. User\_firmante es el user que ha firmado anteriormente, y la firma es lo que se ha obtenido anteriormente en firmar\_datos(). El procedimiento de esta función es el siguiente: carga las claves asimétricas del user\_firmante, comprueba que el certificado de clave pública es válido para usar posteriormente la clave pública de ese certificado en la verificación de la firma usando el mismo algoritmo hash:SHA256 (todo esto en el caso de que el certificado sea válido, si no se devuelve que la firma no es válida porque el certificado no lo era tampoco), y crea un log en Log\_Firma con información del proceso (con información que vamos a ver en una foto a continuación). En el caso de que la firma sea válida, se devuelve True, y si no devuelve False.

Tras explicar cómo firmamos los datos y verificamos que el proceso es correcto, se va a detallar en que partes hemos implementado esto, otorgando al sistema autenticación en estos procesos. La aplicación de firma digital se produce en los siguientes casos:

|  |  |  |  |
| --- | --- | --- | --- |
| Acción | Descripción | Firmante | Quien verifica |
| User hace petición de rol | Petición tipo “Rol”, se firman tipo+user\_accedido (concatenado). Se guarda una petición de tipo “Rol”, que si es verificada correctamente y aceptada, se genera una petición\_confirmada de tipo “Rol” que cambiará el rol del usuario. | User | Sistema o admin |
| User hace petición de devolución |  | User | Sistema o admin |
| User hace petición de compra de entrada |  | User | Sistema o admin |
| Sistema o un admin acepta la petición de compra de entrada de un usuario |  | Sistema o admin | User que solicitó la compra de la entrada |

# **CERTIFICADOS DE CLAVE PÚBLICA**

En cuanto al uso de certificados en nuestra aplicación, hemos diseñado una estructura de clave pública (PKI) en la cual simulamos que el cine (es decir, el usuario “Sistema”) es una autoridad de certificación raíz. De esta forma, cuando un usuario se da de alta en la aplicación se añade una petición de certificado de clave pública, a una tabla de la base de datos, llamada *csr*, donde se almacenarán todas las peticiones de certificado.  De esta forma el AC “Sistema” verificará la autenticidad de cada solicitud de certificado de la tabla *csr*, y en caso de que la verificación sea correcta, devolverá al usuario correspondiente un certificado de su clave pública, firmado por el propio “Sistema”.

Respecto a la implementación de los certificados en nuestro programa, hemos usado los certificados X.509 de la librería *cryptography* de Python. A continuación, se mostrarán los diferentes fragmentos de código que implementan el uso de certificados en nuestro programa:

* Creación del certificado autofirmado del AC, es decir, el usuario “Sistema”:

Para la creación del certificado autofirmado del “Sistema” primero se crea una clave privada para el sistema, que se pasa posteriormente al método *certificado\_propio()*. En *certificado\_propio()* se genera un certificado con el método x509.CertificateBuilder. Este certificado contiene información acerca del usuario al que se certifica (subject), el usuario que emite el certificado (issuer), que en este caso son ambos el “Sistema”; la clave pública asociada al subject y el periodo de validez del certificado. Este certificado se firma con la clave privada del “Sistema” y se guarda codificado, como un PEM, en la columna “Certificado” del registro sistema, dentro de la tabla ASYMETHRIC\_KEYS de SQL.

* Creación de una solicitud de certificado.

Para crear una solicitud de certificado invocamos al método *crear\_csr()* que genera un objeto CertificateSigningRequestBuilder donde establecemos el subject al usuario pasado por parámetro. La solicitud es firmada con la clave privada del usuario y se guarda la solicitud codificada como un PEM en la tabla CSR de la base de datos.

Esta solicitud se genera cada vez que se crean las claves de un nuevo usuario:

Como se puede observar, al generar las claves de un usuario nuevo, se guardará una clave pública, de forma provisional, en el campo de “certificado” de la tabla ASYMETHRIC\_KEYS hasta que la solicitud del certificado sea validada por el sistema.

* Validación y creación de un certificado firmado por el AC(“Sistema”):

Para validar una solicitud de certificado se invoca al método *verificacion\_CertificateRequest()* que comprueba que los datos a firmar de la petición del certificado(csr.tbs\_certificate\_bytes) son los mismos que los que se obtienen al aplicar la clave pública ,asociada a la solicitud del certificado, sobre la firma de la solicitud.

El método encargado de crear un certificado en base a una solicitud de certificado es *crear\_cert()*. Este método primero verifica la solicitud llamando a *verificacion\_CertificateRequest(*). Si la solicitud es válida este método devuelve un certificado, asociado a la clave pública de la solicitud, firmado por el sistema y válido por 90 dias.

 (Falta la parte de creación de verificado implementada)

* Verificación de un certificado y uso del certificado para obtener la clave pública de un usuario:

# **COMPLEJIDAD DE LA APLICACIÓN**