UNIVERSIDAD CARLOS III

CRIPTOGRAFÍA Y SEGURIDAD INFORMÁTICA

2022-2023 GRADO EN INGENIERÍA INFORMÁTICA

ENTREGABLE 2

DESARROLLO DE UNA APLICACIÓN QUE UTILICE CRIPTOGRAFÍA

GRUPO 81

ID GRUPO DE PRÁCTICAS 12

Fernando Galán Núñez NIA:100451280

100451280@alumnos.uc3m.es

Alvaro Bernal Torregrosa NIA:100451179

100451179@alumnos.uc3m.es

ÍNDICE

PARTE 1

1. Propósito de la aplicación

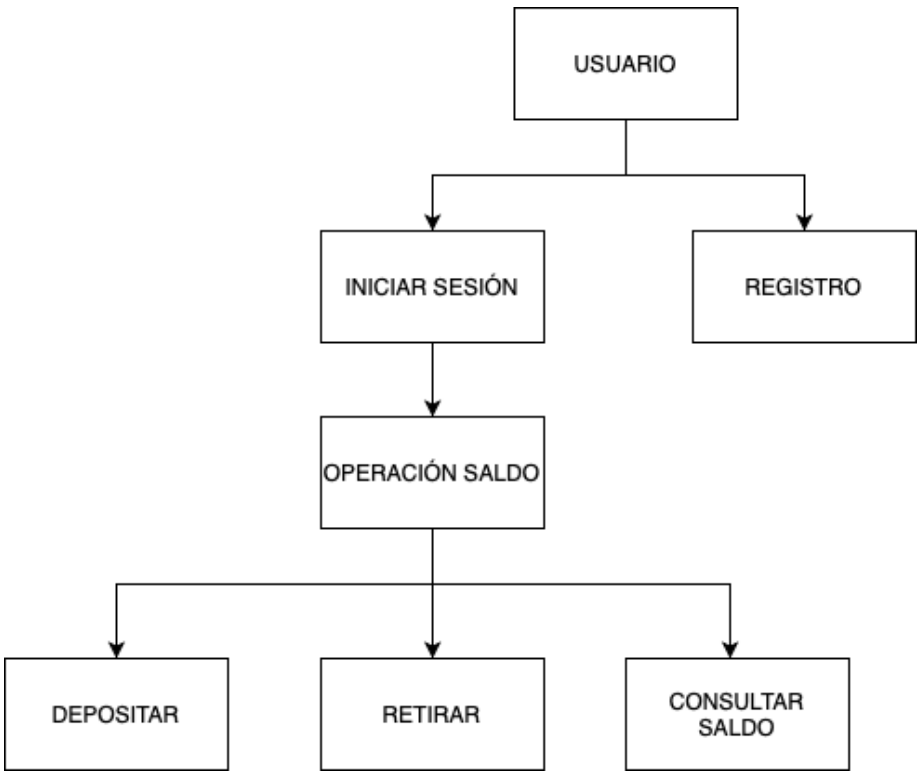
Nuestro propósito para la práctica va a tratar principalmente del funcionamiento de una aplicación para un banco con varios usuarios.

La aplicación te dejará iniciar sesión o registrarse, con tu usuario y con la contraseña, en caso de que el usuario ya exista te aparecerá un mensaje diciendo que el usuario que has escrito ya se encuentra registrado en la base de datos y tendrás que introducir uno nuevo en caso de que quieras registrarte.
Cuando accedas a tu cuenta tendrás la opción de seleccionar varias funcionalidades como consultar tu saldo, depositar saldo o bien retirar saldo si lo deseas, en el caso de que quieras introducir una cantidad negativa, el programa no te dejará y te saldrá un mensaje con ello.

Hemos elegido esta aplicación porque consideramos que a la hora de implementar la criptografía puede resultar muy útil, como por ejemplo la utilización del cifrado y descifrado simétrico, o mediante la utilización de funciones hash y HMAC.

Un ejemplo en el desarrollo de la aplicación sería el siguiente:

Para el uso de la aplicación hemos creado un diagrama que es el siguiente:



2. ¿Para qué utiliza el cifrado simétrico? ¿Qué algoritmos ha utilizado y por qué? ¿Cómo gestiona las claves?

Hemos utilizado el cifrado simétrico para cifrar el saldo de cada usuario. Para su implementación hemos utilizado Fernet que es una herramienta muy fácil de implementar y que nos proporciona un cifrado simétrico y la autentificación de los datos mediante la contraseña del usuario.

Su gestión es muy sencilla, cuando un nuevo usuario se registra se crea un salt único para cada persona. Con él obtenemos el resumen (token) de la contraseña que usaremos para crear la llave con la que podemos cifrar y descifrar el saldo. Por lo tanto, si el usuario no conoce la contraseña no se podrá acceder al saldo.

EJEMPLO DE LA EJECUCIÓN DEL PROGRAMA

Cuando ejecutamos el programa nos pedirá un usuario y contraseña (la cual posee un regex que tiene la condición de que tiene que ser entre 8 y 16 caracteres, al menos un dígito, una minúscula y una mayúscula).

Si el usuario no está registrado nos pedirá que creemos uno nuevo:

```
Nombre Usuario: Juan
Contraseña Usuario: Juan1234
Usuario no encontrado, por favor registrate
Nombre nuevo de usuario: Juan
Introduce nueva contraseña usuario: Juan1234
Bienvenido Juan su saldo es de 2000
```

De esta forma el usuario se quedaría almacenado en un archivo csv llamado data.csv con sus datos personales cifrados.

Si una persona accediera al archivo donde se guardan los datos vería esto:

Usuario (Juan), **Contraseña** (3e1e0808236abbc40d5082fb84572665), **Saldo** (gAAAAABjYWX5XhTaTbrswiRjePJ4e6zHF3flfFF_twxiX5XpwR3jR9y2JQR-ikpuPgjaZZ04a_q _ytZjgSBL9U6S54gtTj68uA==), **Salt** (f0ce39c778d4bc79599d4bfad2dfc0d0), **Salt_saldo** (0691b85e0d1042adf7cb2071d9c5009d)

En caso de que el usuario esté ya registrado, y quiere depositar saldo, la salida del programa sería la siguiente:

```
Nombre Usuario: Juan
Contraseña Usuario: Juan1234
Usuario ya registrado
Que operación quieres realizar:
1 - Depositar | 2 - Retirar | 3 - Consultar Saldo | 4 - Salir
¿Qué desea hacer?: 1
Usted eligió Depositar
¿Cuánto desea depositar?: 700
3200.0
Bienvenido Juansu saldo es de 3200.0

Process finished with exit code 0
```

Y se modifican los datos en el archivo csv de la siguiente manera:

Usuario (Juan), **Contraseña** (3e1e0808236abbc40d5082fb84572665), **Saldo** (gAAAAABjYWluG-qxdRXZ7COF4B4qTNJb0bCSPSx8WFThk3T86_9hUAzALM8H2HTrSnJtM3nAxs-HAYkmZEZSPjfaWhtT6hAkQQ==), **Salt** (f0ce39c778d4bc79599d4bfad2dfc0d 0), **Salt_saldo** (0691b85e0d1042adf7cb2071d9c5009d)

Como se puede ver el único dato en claro que se puede consultar es el usuario. Todo lo demás son los datos personales del usuario cifrados (saldo y contraseña) y los códigos necesarios para poder consultarlos (salt de la contraseña y del saldo).

3. ¿Para qué utiliza las funciones hash o HMAC? ¿Qué algoritmos ha utilizado y por qué? En caso de HMAC, ¿cómo gestiona la clave/s?

La función hash la utilizamos para guardar la contraseña de una forma segura. Para crearla utilizamos el algoritmo PBKDF2 con el que creamos esa función resumen. Lo bueno de estos algoritmo es que es irreversible por lo que para saber si un usuario está introduciendo la contraseña bien usamos el algoritmo y si se crea la misma función resumen es que la contraseña que se ha introducido es la misma que hay guardada y, por lo tanto, es correcta.

Hemos utilizado el algoritmo PBKDF2 porque nos permitía elegir el algoritmo para crear la función resumen. Este ha sido el SHA256 que es conocido por ser muy seguro y tener una implementación sencilla y que no suele dar muchos problemas.

PARTE 2

1. Firma digital

Utilizamos el algoritmo RSA para cifrar el último acceso del usuario (fecha en la que el usuario se registra o inicia sesión), de esta forma podemos comprobar la autenticidad del usuario y que no se ha accedido de forma fraudulenta.

Nuestro RSA dispone de claves de 2048 bits que son creadas en la primera ejecución y las guardamos como dos archivos .pem (“pem_private.pem” y “pem_public.pem”) .

La generación de claves se hace en la función *“generar_claves”* que crea tanto la pública como la privada y las guarda cada una en un archivo *.pem*. Está función solo se ejecutó una vez, ya que sino, cada vez que iniciamos el programa se crearían nuevas claves.

Para poder firmar el mensaje debemos obtener la clave privada del sistema con la que lo firmaremos, para obtener la privada llamamos a la función *“obtener_clave_privada”* y es necesario conocer la contraseña del sistema que en este caso, por temas de simplicidad, está en la parte superior del código, pero en un entorno real esto no sería así.

Por otro lado, la clave pública se obtiene llamando a la función *“obtener_clave_publica”* que se puede acceder sin ningún tipo de seguridad ya que debe ser accesible a todo el mundo. En esta función obtenemos dicha clave pública a través de la clave privada.

Explicamos la verificación de la firma en el punto 2, ya que se verifica con la clave pública del sistema.

2. Certificados

Las claves públicas que usamos al ser de libre acceso para todos no podemos afirmar con seguridad que son del sistema. Para ello utilizamos una infraestructura de clave pública (PKI) con la que certificamos las claves.

Nuestra infraestructura está formada por el sistema (A), es decir la entidad firmante que va a firmar el mensaje, y por una autoridad de certificación (AC1) qué es la que nos asegurará que la clave pública que estamos usando es la del sistema. Para crear la PKI hemos usado las herramientas por línea de comandos ofrecidas por la librería OpenSSL.

Mediante la línea de comandos de OpenSSL creamos el par de claves de A (RSA), y le pedimos a la autoridad de certificación (AC1) que las certifique. En este proceso de generación de solicitudes de emisión de certificado, rellenenamos todos los campos que se nos solicitan, que en nuestro caso serian los siguientes: país “ES”, provincia “MADRID”, localidad “LEGANÉS”, organización es “BANCO”y el nombre común es PKICSR y por último el email “PKICSR@SPI.INF.UC3M.ES”.

Así, AC1 como es una autoridad de verificación genera los certificados tanto del sistema (A) como del propio AC1. Estos certificados nos permiten saber que la clave que usamos le pertenece verdaderamente a A y no a otro sistema.

Para comprobar que el certificado de firma asociado es el auténtico, es decir, el que es el certificado por AC1, utilizamos la función *"verificación_certificado"* que verifica el certificado del sistema (A) con la clave pública de AC1 y el certificado de AC1 que se verifica a sí mismo con su propia clave.
Para verificarlo comprobamos los campos *signature, tbs_certificate_bytes, padding.PKCS1v15() y signature_hash_algorithm* en los dos certificados.

Por último para verificar que nuestra firma, la firma el sistema, deberemos comprobar mediante la clave pública de nuestro sistema (A), que verdaderamente estamos firmando con nuestro sistema y no con otro. Por eso en nuestra función *"verificar_firma",* usamos dicha clave. En un principio utilizabamos la clave pública que generabamos con la clave privada del RSA, pero para comprobar realmente que la firma nuestro sistema deberemos realizar lo descrito anteriormente.

3. Mejoras implementadas

En cuanto a las mejoras implementadas en comparación con la parte 1 hemos realizado las siguientes:

Comprobación contraseña: en la código de la anterior práctica cuando el usuario no introducía bien su contraseña se generaba un bucle infinito hasta que esta fuese correcta, en esta práctica lo que realizamos es que le damos al usuario tres intentos para que introduzca su contraseña, en el caso de que en estos tres intentos no consiguiese introducir la correcta el usuario sería bloqueado y expulsado del sistema para evitar ataques de ciberseguridad.

Más opciones de menú: en cuanto a las nuevas opciones de menú que hemos generado son las siguientes: si pulsa 4 podrás firmar el mensaje que en nuestro caso es la fecha actual en la que el usuario inicia sesión o se registra de tu mensaje, y una opción 5 que va a verificar que la firma del mensaje es correcta mediante la clave pública del sistema (A).

Depositar dinero (opción 1): en caso de que el usuario intente introducir una cantidad negativa, el saldo no se modificaría ya que la cantidad a ingresar es ilógica.

Retirar dinero (opción 2): en caso de que el usuario intente retirar una cantidad negativa, el saldo no se modificaría ya que la cantidad a ingresar es ilógica.
Además si el usuario tiene una deuda mayor de 10000 al banco es decir que tiene un saldo de -10000, el sistema no le dejará retirar más dinero ya que tiene una deuda con el banco. También si por ejemplo tiene una deuda de 5000 con el banco y quiere retirar 6000, es decir que se le quedaría un saldo de -11000, no le dejaría ya que la deuda sería superior a

10000. El banco admite una deuda máxima de 10000, en caso de que esta deuda sea superior no le dejará retirar más dinero del indicado.

Modificación OpenSSL: cuando configuramos nuestro archivo con el ejemplo del OpenSSL, y generamos nuestras solicitudes de certificados modificamos el md por defecto de sha1 a sha256 y los bits por defecto de 1024 a 2048

Un ejemplo sería el siguiente:

```
"/Users/fernando19/Desktop/UNI/3 CURSO/1 CUATRIMESTRE/criptografia/venv/bin/python"
 /Users/fernando19/PycharmProjects/criptografia/main.py
Nombre Usuario: J
Contraseña Usuario: Jjjj1111
Usuario ya registrado
La fecha en la que usted ha iniciado sesión es el 07/12/22 a las 14:48:03
Que operación quieres realizar:
1 - Depositar | 2 - Retirar | 3 - Consultar Saldo | 4 - Verificación firma | 5 - Verificación certificados | 6 - Cerrar sesión
¿Qué desea hacer?: 2
Usted eligió Retirar
¿Cuánto desea retirar?: 11000
Usted tendría una deuda de -10714.0 con el banco, no puede retirar esa cantidad dinero
Saldo no modificado
Bienvenido J su saldo es de 286.0

Process finished with exit code 0
```

Donde el usuario J dispone de un saldo de 286 y quiere retirar 11000, el banco no le dejaría ya que la deuda sería superior a 10000.

El uso de la aplicación sería el siguiente: