Criptografía y Seguridad Informática

Práctica: Programación Lineal

Covid-19 vaccine certificate generator Grupo 82, Grupo de prácticas 23 Isabell Helling 100467780, Alejandro Gonzalez Nuñez 100429135

1. Propósito de la aplicación:

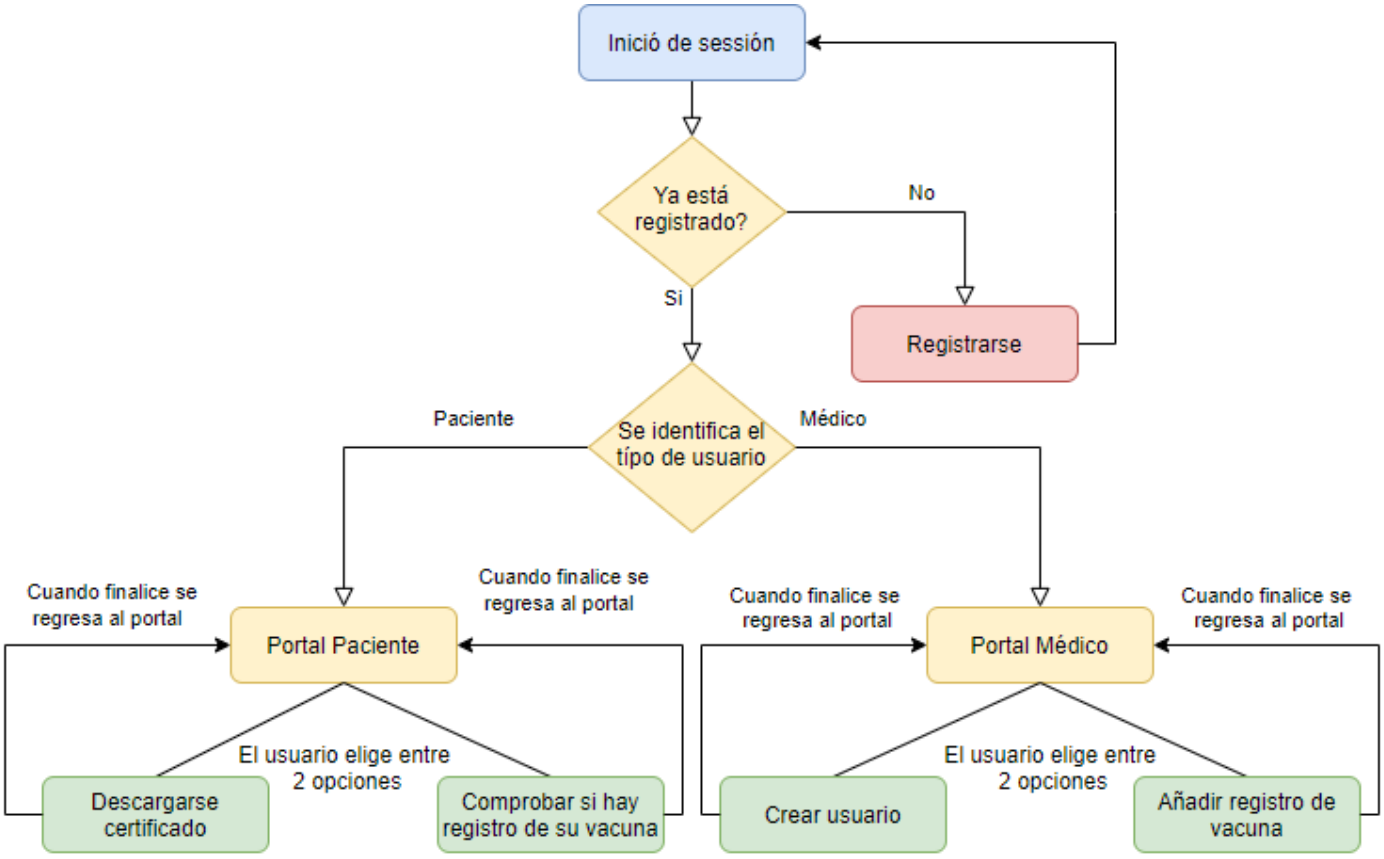
1.1 Propósito App:

La finalidad de esta aplicación es poder implementar una librería de criptografía de python a un programa que hemos creado usando python y SQLite.

Teniendo en cuenta lo anteriormente descrito, el propósito de la app es que un paciente pueda descargarse un documento PDF donde indique, con exactitud, sus datos y si está completamente vacunado de covid-19 (solo si tiene completa su vacuna). Este Pdf debe estar validado y firmado por un médico.

Ahora bien, para lograr dicha meta es fundamental que este procedimiento se lleve a cabo de una forma: confidencial, íntegra y que esté disponible para el usuario cuando lo requiera. Por ello es importante implementar una serie de cifrados y métodos criptográficos para que los datos de los usuarios estén seguros y no se encuentren comprometidos en caso de un intento de fraude o hackeo de la base de datos.

1.2 Flujo App:



1.2 Diseño App:

Nuestra aplicación ha sido diseñada empleando buenas prácticas para que el código sea entendible y organizado. Se hace uso de 4 clases y cada una con una serie de métodos/funciones. Cabe resaltar que todos los datos son almacenados en una base de datos local usando “DB Browser for SQLite“ lo cual facilita que se puedan agregar usuarios y que sus datos y claves queden almacenados

Es por ello que dividiremos el diseño en cómo hemos ido organizado nuestro código y dando una explicación de cada una de nuestras clases y cuales son los objetivos:

1.2.1 Generador de Pdf: Creamos una clase llamada “Pdf Generator” la cual utiliza una librería de python llamada “Free Pdf (FPDF)” dicha librería nos permite crear en tiempo de ejecución un PDF con los datos de los pacientes que tenemos almacenado en la base de datos. A su vez permite agregarle estilo a dicho pdf y diseñarlo como deseas, sin embargo teniendo en cuenta el propósito de esta práctica hemos hecho que sea un pdf sencillo.

1.2.2 Criptografía: con el fin de adaptar la librería de “Cryptography.io” a las funcionalidades deseadas en nuestra aplicación, decidimos crear una clase llamada “Cryptography”. La clase tiene la misma funcionalidad que la librería pero organizada a lo que deseamos implementar permitiendo: generar claves, autenticar encriptar y desencriptar.

1.2.3 App: Esta clase es la encargada de toda la funcionalidad de nuestro programa, es decir, en dicha clase se ejecutan las distintas acciones que llevan a cabo los usuarios en la aplicación haciendo llamadas a las demás clases y funciones.

1.2.4 main(): Hemos hecho de él main nuestra interfaz por líneas de comando en consola, es la raíz de nuestra app y se encarga de hacer las llamadas a todas las demás clases y funciones, para que se lleve a cabo todos los procedimientos adecuados en busca de cumplir con el propósito establecido por los usuarios.

1.2.5 Base de datos en “DB Browser for SQLite”: Todos los datos de los usuarios son guardados en una base de datos la cual contiene 3 tablas: usuarios, salto y vacuna covid. cabe resaltar que en el salt solo se almacena un dato que es el salt de la clave maestra. Esto permite almacenar correctamente los datos de los usuarios y así insertar y obtener los datos de un paciente cuando se desea. Es importante destacar que en dicha base de datos se almacenan tanto los “Salt” como las claves simétricas con las que se cifran los datos personales de los usuarios.

2. Cifrado asimétrico:

El cifrado simétrico no ha sido aplicado en esta aplicación al no considerarse requerida ya que no hay una comunicación directa entre los usuario de la aplicación sino que más bien es algo totalmente local. En su defecto se implementó el cifrado simétrico detallado en la primera parte de esta práctica.

3. Firma Digital:

3.1 Uso de la firma digital:

En nuestro programa hemos hecho uso de la firma digital para firmar los certificados covid generados por los médicos. Para esto utilizamos una Autoridad de certificación raíz que llamamos "The Covid Certificate Organization", Se genera una clave RSA pública y privada para esta entidad y se hace uso de ella para firmar el certificado covid en el momento que es creado. el resultado es un archivo certificate.pem en cual es anexado al pdf para que quede firmado.

3.2 Algoritmos utilizados para la firma digital:

Para firmar hemos usado la librería de "Cryptograpy.io", de ahi usamos los siguientes alñgoritmos/funciones:

- RSA: Usamos RSA para generar un par de claves para la AC1, se usa un exponente de 65537 y un tamaño de clave de 2048, decidimos usar rsa por su simpleza a la hora de ser aplicado en python y que está bastante estandarizado en la actualidad, lo importante es usar un longitud de clave que sea razonable.

```
# Generate our key
key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)
```

- X.509 en su versión 3: Usamos X.509 ya que nos permite crear certificados y establecer un emisor, periodo de validez, sujeto, su clave pública, extensiones. Tienen un formato estándar para certificados de clave pública, es decir documentos digitales que asocian de forma segura pares de claves criptográficas. en nuestro caso.

```
cert = x509.CertificateBuilder().subject_name(
    subject
).issuer_name(
    issuer
).public_key(
    key.public_key()
).serial_number(
    x509.random_serial_number()
).not_valid_before(
    datetime.datetime.utcnow()
).not_valid_after(
    # Our certificate will be valid for 90 days
    datetime.datetime.utcnow() + datetime.timedelta(days=90)
).add_extension(
    x509.SubjectAlternativeName([x509.DNSName(u"localhost")]),
    critical=False,
# Sign our certificate with our private key
).sign(key, hashes.SHA256())
```

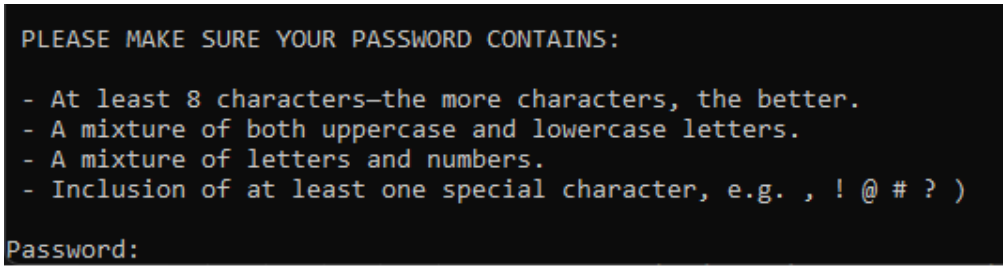
3.3 Gestión de claves:

Las claves son almacenadas en archivos locales de tipo .pem, en nuestro caso no estan organizados de una manera específica sino que se van almacenando 1 a uno en una carpeta en el orden que son creados.

4. Tipo de Autenticación:

4.1 Autenticación basada en secretos:

Para autenticar a los usuarios hemos escogido hacerlo mediante las contraseñas de los usuarios, ya que es mucho más práctico y sencillo de implementar para el alcance de esta práctica. sin embargo nos aseguramos de cumplir con una serie de normas que deben de cumplir las contraseñas de los usuarios.



4.2 Implementación de la autenticación de usuario:

Para implementar el login de los usuarios se hace uso de las claves simétricas almacenadas de los usuarios y se realiza una comprobación de si se genera la misma clave simétrica con el salt almacenado, es decir, Desencriptamos la clave simétrica del usuario con la clave maestra del sistema, generamos una nueva clave simétrica con el salt que tenemos almacenada en la base de datos junto con la contraseña del usuario y luego comprobamos si son la misma clave simétrica, si es correcto entonces el usuario es autenticado.

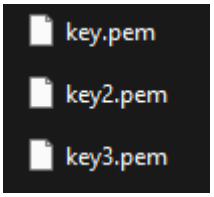
```python
def login(self, dni: str, u_key):
    e_key = self.get_data('users',dni)[0][7]
    d_key = self.crypt.keyDecryptionMasterPassword(self.master_salt, e_key)
    if d_key == u_key:
        return True
    else:
        return False
```

e_key= clave simétrica del usuario generada con el salt almacenado en el sistema

d_key= clave simetrica del usuario que ha sido desencriptada con la clave maestra

4.3 Almacenamiento de contraseñas de usuarios:

Las contraseñas de los usuarios no son almacenadas en el sistema, es decir, almacenamos los salt utilizados para la autenticación, estos son almacenados en la base de datos, así también las claves simétricas derivadas de las contraseñas aunque estás están encriptadas por una clave maestra.



5. Mejoras

- Almacenamiento de claves en base de datos sqllite3: las claves se almacenan convenientemente en un fichero .dat , esto realmente facilita la manipulación de los datos y hace que sea más práctico, sin embargo tuvimos que agregar muchas funciones para ajustarlo todo a nuestra aplicación.

- Validación de los datos que introducen los usuarios: aunque nuestra aplicación tiene una interfaz de comando de usuarios nos encargamos de manejar todo tipo de errores que un usuario puede cometer usando nuestra app. Sobre todo a la hora de introducir datos, es por ello que usamos expresiones regulares (RE) para dar dominios de los posibles valores que un usuario puede introducir. En esta imagen se puede ver un poco como lo hacemos con las contraseñas

```
while True:
    schema = PasswordValidator()
    schema.has().no().spaces()
    schema.has().symbols()
    password = input("Password: ")
    if len(password) < 8:
        print("Make sure your password is at lest 8 letters")
    elif re.search('[0-9]',password) is None:
        print("Make sure your password has a number in it")
    elif re.search('[a-z]',password) is None:
        print("Make sure your password has a lower case letter in it")
    elif re.search('[A-Z]',password) is None:
        print("Make sure your password has a capital letter in it")
    elif schema.validate(password) is False:
        print("Make sure your password has a symbol, no spaces allowed")
    else:
        break
```

- Generar PDFs: con la finalidad de hacer nuestra app más completa y actualizada decidimos que el resultado final sea un pdf con el certificado del usuario el cual indica los datos de los usuarios y la información relacionada con su vacuna covid. En la carpeta de entrega he dejado un certificado pdf pero se pueden crear con la aplicación siguiendo los pasos.