Criptografía y Seguridad Informática

PRÁCTICA DE CRIPTOGRAFÍA ENTREGA 1

Curso 2024/2025

Nombre	NIA	Correo Electrónico	Grupo
Javier Rosales Lozano	100495802	100495802@alumnos.uc3m.es	81
Manuel Roldán Matea	100500450	100500450@alumnos.uc3m.es	81

Fecha de presentación: 31/10/2024

Contenido

1. Introducción

El objetivo de la práctica de Criptografía y Seguridad Informática se centra en el conocimiento y aprendizaje del uso de librerías criptográficas, proponiendo como enunciado el desarrollo de una aplicación cuya funcionalidad es elegida por los alumnos, pero que haga uso de operaciones criptográficas en algún momento de su ejecución.

El siguiente documento presenta toda la información relacionada con el desarrollo de la primera entrega de la práctica, atendiendo a los requerimientos que se piden y las funcionalidades solicitadas en el enunciado de ésta.

Primeramente, se analizará la idea base de la aplicación, explicaremos el funcionamiento básico de la interfaz a nivel de usuario, e identificaremos qué es lo que se va a cifrar. Seguidamente, explicaremos el algoritmo usado para el registro y autenticado de usuarios y contraseñas, y tras eso proporcionaremos la información y explicaciones necesarias para entender el proceso de cifrado de la aplicación. Finalmente, daremos nuestras conclusiones acerca del proyecto, el trabajo realizado, y las posibles mejoras a futuro.

2. Idea del proyecto

La aplicación propone una idea sencilla: el guardado de notas de texto de un usuario. El usuario, una vez inicie sesión, accederá a una pantalla dónde se mostrarán las notas de texto que ha creado. Esta nota creada almacenará un título y el contenido textual proporcionado por el usuario, la fecha y la hora de creación de la nota. Todo se centrará en una única interfaz en la que el usuario podrá únicamente visualizar sus notas creadas. Obviamente, las notas que se muestran para un usuario no serán las mismas que se muestran para otro usuario registrado. Es por ello por lo que nuestra aplicación otorgará un **servicio de "bloc de notas" privado** a cada usuario que se registre.

El proyecto se ha realizado usando el lenguaje de scripting Python, importando diferentes librerías para cumplir el objetivo que se tenía para el funcionamiento básico de la aplicación. Con la idea de segmentar el código, el código se ha dividido en diferentes módulos que otorgan las funcionalidades por separado de la aplicación, aislando las funcionalidades que no dependen una de otra:

app.py	Lógica general de la interfaz de Tkinter
screens.py	Maneja la lógica de las pantallas mostradas por la interfaz general. Cada clase declarada en este módulo representa un tipo de pantalla que se puede mostrar al usuario.
note.py	Módulo que almacena la declaración de los objetos "nota".
register_authenticate.py	Registro y autenticado de usuario

key.py	Creación de “password token” y clave de cifrado.
aes.py	Maneja las operaciones de cifrado y autenticado de datos del usuario (algoritmo de cifrado simétrico AES-GCM).
constants.py	Importación de constantes
main.py	Ejecución del programa

3. Registro-Autenticado

La primera parte de la aplicación consiste en un **registro de usuarios**, en el cual cada usuario tendrá que introducir un nombre de usuario y una contraseña con unas restricciones específicas; la contraseña deberá ser mínimo de 8 caracteres entre los cuales se incluya, al menos, un número, una mayúscula y un carácter especial. Después del registro de un usuario, se almacenarán sus credenciales.

Para almacenar las contraseñas de los usuarios de manera segura se ha usado un algoritmo de derivación a partir de una contraseña que dará como resultado un **token de contraseña**, que se referencia como `pwd_token` en el código. El algoritmo usado es `Scrypt`, de la librería `Cryptography` de Python; este algoritmo genera el `pwd_token` a partir de un `salt` (número aleatorio de 16 bytes). Cuando el algoritmo genera el token a través de una función denominada .derive(), se almacena en un JSON en base64 (`usuarios.json`) junto a un `salt` por cada usuario. De esta manera, seremos capaces de autenticarlo posteriormente. Usando este algoritmo se consigue almacenar la contraseña de cada usuario de manera segura y privada para los agentes externos.

```python
def derivate_pwdtoken(self):
        """Genera una clave a partir de la contraseña"""
        kdf = Scrypt(salt=self.salt_pwd, length=32, n=2 ** 14, r=8, p=1)
        # Derivamos la clave usando la contraseña en formato bytes
        pwd_token = kdf.derive(self.password)
        return self.salt_pwd, pwd_token
```

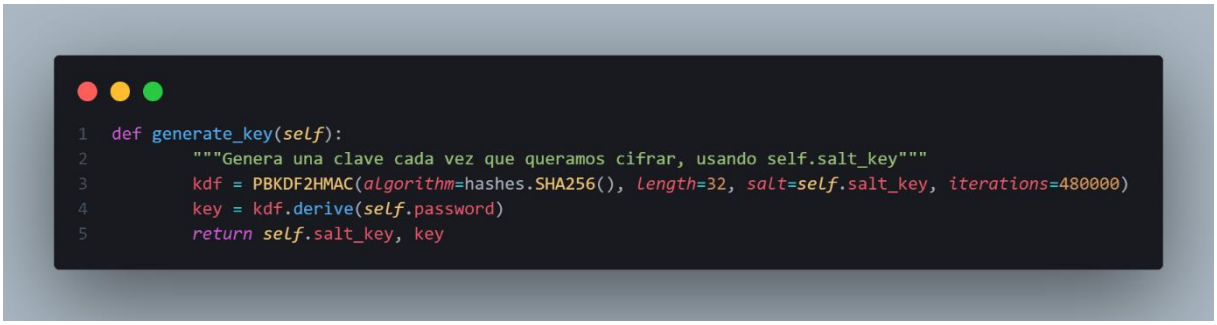
Para el autenticado del usuario, el cual se produce cuando se pulsa la opción de iniciar sesión, se reutilizan los datos del `salt` y del `pwd_token`; primeramente, se decodifican (ya que están almacenados en base64), después se volverá a generar el objeto `Scrypt` en una variable con el `salt`, y posteriormente se usará la función `.verify()`, que recibe por parámetro la contraseña y el `pwd_token`, de manera que pueda verificarlo. Si no salta ninguna excepción, la función encargada de la verificación devolverá TRUE, y el usuario podrá entrar a ver sus notas personales.

```python
def authenticate(self, user: dict):
        """Verifica la contraseña del usuario"""
        salt_json = user["salt_pwd"]
        key_json = user["pwd_token"]
        # Convertimos el salt y la clave desde base64 si son cadenas
        if isinstance(salt_json, str):
            salt = base64.b64decode(salt_json)
        else:
            salt = salt_json
        if isinstance(key_json, str):
            key_token = base64.b64decode(key_json)
        else:
            key_token = key_json
        # Preparamos el KDF con el salt almacenado (longitud de 32 bytes)
        kdf = Scrypt(salt=salt, length=32, n=2 ** 14, r=8, p=1)
        # Verificamos la contraseña, que debe estar en bytes
        try:
            # Verificamos si la clave derivada coincide (kdf.verify())
            kdf.verify(self.password, key_token)
            return True
        except Exception as e:
            return False
```

4. Cifrado de datos

En lo referente al cifrado-autenticado de datos correspondiente a las partes 2 y 3 del enunciado, se ha decidido usar el **algoritmo de cifrado AES-GCM**. En este caso, al ser una aplicación de notas personales, se deben cifrar y autenticar esas notas; concretamente, el cifrado se realiza sobre el conjunto de notas al completo. En el diseño del programa se crea una clase AES que recibirá por parámetro el nombre de usuario y su respectiva contraseña. Este objeto creará un `nonce` distinto para cada cifrado de notas, es decir, si se añaden notas de algún usuario o de cualquier otro, se instanciará un nuevo `nonce` para el cifrado; y por último, recibirá la clave derivada de la contraseña para el cifrado.

La clave derivada de la contraseña que se usa para el cifrado se generará con el **algoritmo PBKDF2**, que a través de un `salt` y la contraseña del usuario, deriva una clave que se usará para el cifrado con AES. Estos dos elementos, el `salt` y la clave de cifrado derivada, se pasarán al objeto AES para llevar a cabo el cifrado.



```python
def generate_key(self):
        """Genera una clave cada vez que queramos cifrar, usando self.salt_key"""
        kdf = PBKDF2HMAC(algorithm=hashes.SHA256(), length=32, salt=self.salt_key, iterations=480000)
        key = kdf.derive(self.password)
        return self.salt_key, key
```

Una vez se instancia el AES con su clave, `salt`, nonce, contraseña y nombre de usuario, se lleva a cabo la función **`encrypt()`**, que cifrará los datos usando la clave derivada anteriormente; en nuestro caso, **se cifra todo el listado de notas personales de un usuario**. La salida de este cifrado se almacena en un archivo JSON (concretamente el `salt` para la clave, el texto cifrado, el `nonce` y el `tag`, que es la etiqueta de autenticado de los datos; todos estos datos se almacenan en base64 utf-8 para que se soporte en formato JSON).

```python
def encrypt(self, lista_notas):
        """Método de cifrado: devuelve un diccionario serializable a JSON que contiene
        el mensaje cifrado, el nonce, el salt y el tag"""
        cifrado = Cipher(algorithms.AES(self.key), modes.GCM(self.nonce))
        cifrador = cifrado.encryptor()
        contenido_str = json.dumps(lista_notas)
        texto_cifrado = cifrador.update(contenido_str.encode()) + cifrador.finalize()
        return {
            "texto_cifrado": base64.b64encode(texto_cifrado).decode('utf-8'),
            "nonce": base64.b64encode(self.nonce).decode('utf-8'),
            "salt": base64.b64encode(self.salt).decode('utf-8'),
            "tag": base64.b64encode(cifrador.tag).decode('utf-8')
        }
```

En el caso del autenticado y descifrado se recurre a la función **`decrypt()`**. Esta función, en primer lugar, generará la clave otra vez con el `salt` y la contraseña para poder usar la función .`decryptor()` que recibirá la clave usada para cifrar, el `nonce` y la etiqueta de autenticado generada anteriormente. Considerando los datos del JSON, la función volverá a generarlos para comprobar que no se han modificado, y una vez se han autenticado, los descifra. Una vez descifrados y autenticados los datos, se usarán para mostrarlos por pantalla, de modo que para el programa el almacenamiento se encuentre cifrado y seguro, y para el usuario registrado se muestre su repertorio de notas creadas. Al usar cifrado-autenticado, nos aseguramos de que las notas de cada usuario están seguras, y además en ningún momento son modificadas o eliminadas del JSON, que es la función del autenticado. De esta manera, sólo podrá actuar sobre las notas el creador de éstas (que es el usuario registrado).

```python
def decrypt(self):
        """Método de descifrado: obtiene el diccionario de cifrado y devuelve los datos
        en claro (la lista con las notas)"""
        datos_decrypt = self.extraer_datos_json()
        if isinstance(datos_decrypt, dict):
            texto_cifrado = base64.b64decode(datos_decrypt['texto_cifrado'])
            nonce = base64.b64decode(datos_decrypt['nonce'])
            salt = base64.b64decode(datos_decrypt['salt'])
            tag = base64.b64decode(datos_decrypt['tag'])
            # Se vuelve a generar la clave con la contraseña
            kdf = PBKDF2HMAC(algorithm=hashes.SHA256(), length=32, salt=salt, iterations=480000)
            key_descifrar = kdf.derive(self.password.encode('utf-8'))
            descifrador = Cipher(algorithms.AES(key_descifrar), modes.GCM(nonce, tag)).decryptor()
            texto_bytes = descifrador.update(texto_cifrado) + descifrador.finalize()
            return json.loads(texto_bytes.decode('utf-8'))
        return datos_decrypt
```

5. Conclusión

En primer lugar, consideramos que la idea de la aplicación, a pesar de no ser la más compleja ni tampoco la más original, cumple con las expectativas de cifrado y autenticado de datos que propone la práctica, además de satisfacer el funcionamiento correcto de ésta. Como futuras mejoras a implementar de cara a la segunda entrega de la práctica podríamos considerar el cifrado de cada nota del usuario por separado, en vez de cifrar la lista de notas como conjunto. Por otro lado, podríamos añadir aspectos como la edición y eliminación de notas, incluyendo nuevas pantallas u otras funcionalidades que la aplicación no presenta para esta primera entrega. Sin embargo, consideramos que el trabajo entregado cumple con lo que queríamos hacer, y nos sentimos gratificados con el resultado obtenido.

Por lo que respecta a nuestra opinión, creemos que el desarrollo de la práctica ha sido productivo en cuanto al aprendizaje del funcionamiento de métodos criptográficos. La libertad del diseño de la aplicación a la hora del desarrollo de los algoritmos criptográficos complementa a la sensación de amenidad que hemos sentido con el avance del trabajo semanal, lo que nos ha servido como motivación para realizarla. La propuesta de la práctica ha resultado ser una fuente de inspiración para futuros proyectos personales y/o académicos.