PROYECTO CRIPTOGRAFÍA

Alberto Menchen Montero 100495692

Javier Moyano San Bruno 100495884

ÍNDICE

1º ¿Cuál es el propósito de su aplicación? ¿Cuál es su estructura interna?
¿Cuál es el propósito de su aplicación?

El propósito de la aplicación es proporcionar un sistema que simule las funciones de un gestor de contraseñas. Cuenta con un sistema de registro y autenticación seguro para los usuarios mediante el cual pueden almacenar cualquier tipo de contraseña y asegurarse de que va a estar encriptada de buena manera y bajo un tratamiento de sus datos personales seguro y eficaz.
Incluye funcionalidades como el cifrado de datos personales, la gestión de contraseñas y restauración de las mismas mediante envío de PINs al correo vinculado a la cuenta. Como estas un amplio abanico de funciones capaces de hacer intuitiva, sencilla y segura la utilización de la aplicación.También, cuenta con técnicas de cifrado y hashing para asegurar la integridad y confidencialidad

¿Cuál es su estructura interna?

La aplicación cuenta con una estructura modular, dividiendo los diferentes métodos en archivos y módulos:

- data_validation.py: Contiene funciones para validar datos de entrada como nombres de usuario, contraseñas, correos electrónicos y números de teléfono. Se siguen los patrones más clásicos de depuración para los campos que se especifican. Estas funciones de depuración sirven para delimitar las fronteras de nuestra aplicación y saber con qué tipo de información vamos a tratar.
  Por ejemplo, algunos de los patrones que se siguen para el email y el teléfono viene delimitados por lo siguiente:
  email:
  ```
  patron = r'^[a-zA-Z0-9._%+-]+@gmail\.(com|es)$'
  ```
  Teléfono:
  ```
  patron = r'^[6-7]\d{8}$'
  ```
  Y algunos ejemplos de depuración de la contraseña y el nombre de usuario es que tenga un segmento limitado del número de caracteres y longitud.
- encryption.py: Este código implementa un esquema de cifrado y descifrado usando AES en modo GCM. La clave de cifrado se deriva de una contraseña mediante PBKDF2-HMAC con SHA-256 y una sal, generando una clave segura de 256 bits. Durante el cifrado, se usa un nonce único y opcionalmente datos adicionales de autenticación (AAD) para mayor seguridad. El texto cifrado y un tag de autenticación se generan y almacenan junto al nonce. Para descifrar, el código utiliza el nonce, el tag, y la clave para restaurar el mensaje original, validando su integridad mediante el tag.
- exceptions.py: Define excepciones personalizadas para manejar errores de validación.
- json_management.py: Maneja la carga y el guardado de datos de usuarios en un archivo JSON. Se usará como base de datos para la información almacenada en nuestra app. Son los datos con los que trabajaremos.
- password_hashing.py: Contiene funciones para generar sal y hash de contraseñas utilizando PBKDF2-HMAC.
- password_restoration.py: Implementa la funcionalidad de cambios de contraseña mediante el envío de correos electrónicos registrados.
- profile_management.py: Gestiona las operaciones relacionadas con el perfil del usuario, como la consulta y el almacenamiento de contraseñas. Aquí se desarrollan

las funcionalidades principales de nuestra aplicación, que tratan de consultar las contraseñas almacenadas y agregar algunas nuevas.

- register_login.py: Maneja el registro y la autenticación de usuarios. Funciones que recogen todos los datos iniciales del usuario y manejan su inicio de sesión.
- app.py: Implementa la interfaz gráfica de usuario utilizando Tkinter. Aprovechando que estamos trabajando en otra asignatura de la carrera con archivos CSS encontramos esta forma de interfaz que seguía una estructura muy parecida a la que ya conocíamos.

2º ¿Cómo se realiza la autenticación de usuarios? ¿Qué algoritmos ha utilizado y por qué? Detalle cómo se gestionan las contraseñas de los usuarios y si se generan claves a partir de éstas.

register_login.py

Registro de Usuario:

- Validación de Datos: Antes de almacenar los datos de usuario, se validan el nombre de usuario, la contraseña, el email y el teléfono para garantizar que cumplen con los requisitos.
- Generación de Sal y Hashing de Contraseña: Para asegurar las contraseñas, se genera una sal aleatoria única por usuario, y la contraseña se procesa con PBKDF2-HMAC usando SHA-256. Este proceso produce un hash seguro de 256 bits, que se almacena en lugar de la contraseña original.
- Cifrado de Datos Sensibles: El email y el teléfono se cifran utilizando AES en modo GCM, que proporciona confidencialidad y autenticación de integridad. Para el cifrado, se deriva una clave segura a partir de la contraseña y una segunda sal mediante el mismo algoritmo PBKDF2-HMAC.
- Almacenamiento Seguro: Finalmente, la sal, el hash de la contraseña, y los datos cifrados (email y teléfono, junto con el nonce y el tag de autenticación) se almacenan en un archivo JSON, protegiendo los datos personales de accesos no autorizados.

Autenticación de Usuario:

- Verificación de la Contraseña: En el proceso de autenticación, la contraseña proporcionada por el usuario se transforma nuevamente en un hash con la misma sal almacenada y se compara con el hash guardado. Si coinciden, la autenticación es válida.
- Derivación de Clave de Sesión: Si la contraseña es correcta, se deriva una clave de cifrado a partir de la contraseña y la sal de cifrado. Esta clave se utiliza para descifrar los datos personales cifrados y manejar las futuras solicitudes del usuario de manera segura.

Algoritmos Utilizados:

- PBKDF2-HMAC con SHA-256: Este algoritmo se usa tanto para el hashing de la contraseña como para la derivación de claves de cifrado. Su capacidad para realizar múltiples iteraciones y el uso de una sal aleatoria lo hace resistente a ataques de fuerza bruta y ataques de diccionario.
- AES-GCM: AES en modo GCM cifra el email y el teléfono, garantizando tanto la confidencialidad como la autenticidad de los datos mediante un tag de autenticación. Este modo es eficiente y se considera seguro para la protección de datos sensibles, especialmente en combinación con claves generadas mediante PBKDF2-HMAC.

password_restoration.py

Método de recuperación de contraseñas:

La función enviar_correo_aviso_cambio_contraseña envía un correo de notificación cuando se cambia la contraseña de un usuario. Primero, recupera el email cifrado del usuario desde json_management y lo descifra con AES-GCM usando la clave de sesión (clave_sesion), un nonce, y un tag para asegurar autenticidad. Luego, configura un servidor SMTP de Gmail con las credenciales del remitente. Crea un mensaje MIME con el asunto y el cuerpo personalizado que alerta sobre el cambio de contraseña.

Para su implementación, tuvimos que generar una "contraseña de aplicación" para uno de los correos electrónicos que iba a ser el encargado de administrar todas estas funciones. De esta manera, conseguimos automatizar el envío instantáneo de esos correos para todo tipo de correos válidos.

Además, descubrimos que no podíamos ser capaces de identificar si un correo existía o no ya que la política de privacidad de GMAIL "anti spam" no te averiguar si un correo existe o no,pero si si tiene un formato válido.

Teníamos implementada otra función de "Olvide mi contraseña" que generaba un PIN y te lo enviaba por correo electrónico siguiendo la misma lógica, pero tuvimos varios problemas con el encriptado de la contraseña. Si no iniciaste sesión con la contraseña no se generaban las claves de encriptado y, por lo tanto, era ineficiente el sistema.

El código era el siguiente:

```python
def generar_pin():
    return ''.join([str(random.randint(0, 9)) for _ in range(6)])


# Método para restaurar la contraseña
1 usage  ▲ ppppp
def restaurar_contraseña(nombre_usuario, email_proporcionado, clave_sesion):
    usuarios = json_management.cargar_usuarios()
    if nombre_usuario not in usuarios:
        raise ValidationError("Usuario no encontrado.")

    user_data = usuarios[nombre_usuario]
    email_cifrado = base64.urlsafe_b64decode(user_data['email']['cifrado'])
    email_nonce = base64.urlsafe_b64decode(user_data['email']['nonce'])
    email_tag = base64.urlsafe_b64decode(user_data['email']['tag'])

    email_descifrado = encryption.descifrar_aes_gcm(email_cifrado, clave_sesion, email_nonce, email_tag)

    if email_descifrado != email_proporcionado:
        raise ValidationError("El correo electrónico no coincide con el registrado.")

    pin = generar_pin()

    # Configuración del servidor de correo
    servidor_correo = "smtp.gmail.com"
    puerto = 587
    correo_envio = "100495692@alumnos.uc3m.es"
    contraseña_correo = "miep iewr zlmc ycfp"

    # Crear el mensaje de correo
    mensaje = MIMEMultipart()
    mensaje['From'] = correo_envio
    mensaje['To'] = email_descifrado
    mensaje['Subject'] = "Restauración de contraseña"
    cuerpo_mensaje = f"Hola {nombre_usuario},\n\nTu PIN de restauración de contraseña es: {pin}\n\nSi no has solicitado un restablecimiento de contraseña, por favor ignora este correo."
    mensaje.attach(MIMEText(cuerpo_mensaje, 'plain'))

    # Enviar correo
    try:
        servidor = smtplib.SMTP(servidor_correo, puerto)
        servidor.starttls()
        servidor.login(correo_envio, contraseña_correo)
        servidor.send_message(mensaje)
        servidor.quit()
    except smtplib.SMTPException as e:
        raise ValidationError(f"Error al enviar el correo electrónico: {e}")

    return pin
```

profile_managment.py

- **consultar_usuario**: Recupera y descifra el email y el teléfono de un usuario usando claves derivadas para el acceso seguro a estos datos personales.
- **guardar_contraseña**: Cifra y almacena una nueva contraseña en el perfil del usuario, junto con un asunto descriptivo, y guarda los datos cifrados en el almacenamiento.
- **obtener_contraseñas**: Descifra y devuelve todas las contraseñas almacenadas para un usuario, utilizando una clave de sesión derivada para validar la autenticidad.
- **eliminar_contraseña**: Elimina una contraseña específica del usuario, identificada por el asunto, y actualiza la información en el almacenamiento JSON.

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

El cifrado simétrico en el proyecto se utiliza para proteger datos sensibles de los usuarios, específicamente el correo electrónico, el número de teléfono y las contraseñas guardadas. La clave de cifrado se deriva de la contraseña del usuario, proporcionando un método seguro y personalizado para cada usuario, ya que los datos cifrados solo pueden ser descifrados con la clave derivada de su propia contraseña.

```
clave = encryption.derivar_clave_cifrado(password, salt_cifrado)
email_cifrado = encryption.cifrar_aes_gcm(email, clave)
telefono_cifrado = encryption.cifrar_aes_gcm(telefono, clave)
```

Este código, en register_login.py, muestra cómo se cifra el correo electrónico y el teléfono de cada usuario tras derivar la clave del cifrado simétrico con la contraseña y el salt generado.

Para el cifrado simétrico se ha utilizado AES en modo GCM. Este modo fue elegido por sus propiedades de autenticación y encriptación en un solo paso, ya que GCM (Galois/Counter Mode) es un modo de operación de AES que ofrece autenticación de mensajes mediante un tag y asegura la integridad del mensaje. AES-GCM se ajusta bien al sistema porque:

1. Cifra y auténtica simultáneamente, proporcionando confidencialidad e integridad.
2. Eficiencia y rendimiento para manejar datos como correos electrónicos y contraseñas.

```
def cifrar_aes_gcm(mensaje: str, clave: bytes, aad: bytes = None) -> dict:
    nonce = os.urandom(12)  # Genera un nonce de 12 bytes para AES-GCM
    cipher = Cipher(algorithms.AES(clave), modes.GCM(nonce), backend=default_backend())
    encryptor = cipher.encryptor()
    if aad:
        encryptor.authenticate_additional_data(aad)

    cifrado = encryptor.update(mensaje.encode()) + encryptor.finalize()
    tag = encryptor.tag  # El tag de autenticación de GCM

    logging.debug(f"Cifrado AES realizado. Algoritmo: AES, Longitud de clave: {len(clave)*8} bits")
    logging.debug(f"Nonce utilizado: {base64.urlsafe_b64encode(nonce).decode('utf-8')}")
    logging.debug(f"Etiqueta de autenticación (tag): {base64.urlsafe_b64encode(tag).decode('utf-8')}")

    return {
        'nonce': nonce,
        'cifrado': cifrado,
        'tag': tag
    }
```

Las claves de cifrado se derivan de la contraseña del usuario junto con un salt mediante PBKDF2-HMAC con SHA-256 como función de derivación de clave, realizando 100,000 iteraciones. Este método asegura que cada usuario tenga una clave única, aunque puedan tener contraseñas similares.

1. **Generación de Salt:** Cada usuario tiene su propio salt para la contraseña y otro para el cifrado, lo cual se almacena en el archivo de datos del usuario.
2. **Almacenamiento Seguro:** El salt y la clave derivada no se almacenan directamente; en su lugar, se almacena el hash resultante de la combinación de la contraseña y el salt.

```python
def derivar_clave_cifrado(password: str, salt: bytes) -> bytes:
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,  # Longitud de la clave AES de 256 bits
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    clave = kdf.derive(password.encode())
    logging.debug(f"Clave derivada con KDF. Algoritmo: SHA256, Longitud de clave: {len(clave)*8} bits")
    return clave
```

4. ¿Para qué utiliza las funciones de códigos de autenticación de mensajes (MAC)? ¿Qué algoritmos ha utilizado y por qué? ¿Cómo gestiona la clave/s? Explicite si utiliza algoritmos de cifrado autenticado y las ventajas que esto ofrece.

En este sistema, el tag que se genera al cifrado con AES-GCM actúa como un código de autenticación de mensajes (MAC). Este tag permite verificar la integridad y autenticidad del mensaje cifrado, asegurándose de que no se haya alterado durante la transmisión o almacenamiento. En caso de que el tag no coincida al intentar descifrar, el sistema rechaza el mensaje, protegiéndolo contra manipulaciones de datos.

```python
tag = encryptor.tag  # El tag de autenticación de GCM
logging.debug(f"Etiqueta de autenticación (tag): {base64.urlsafe_b64encode(tag).decode('utf-8')}")

return {
    'nonce': nonce,
    'cifrado': cifrado,
    'tag': tag
}
```

El sistema usa AES-GCM para el cifrado simétrico y la autenticación de mensajes. Este algoritmo fue elegido por su capacidad para proporcionar autenticación y cifrado en un solo paso. AES-GCM es eficiente y se adapta a necesidades de cifrado que requieren verificación de integridad, lo cual es esencial para los datos sensibles.

La clave del MAC no es gestionada de forma independiente, ya que el tag se deriva automáticamente como parte del proceso de cifrado con AES-GCM usando la misma clave de cifrado del usuario. Esto simplifica la gestión de claves al unificar el cifrado y la autenticación en un solo flujo de trabajo, donde la misma clave cifrada sirve tanto para proteger el mensaje como para generar el tag.

Sí, AES-GCM es un algoritmo de cifrado autenticado. Las ventajas de usar este algoritmo incluyen:

- Confidencialidad y autenticidad: Garantiza que sólo el destinatario con la clave correcta puede leer el mensaje y verificar su integridad.
- Protección contra ataques de manipulación: Cualquier intento de modificar el mensaje altera el tag y fallará al momento de descifrar.
- Eficiencia: AES-GCM es un algoritmo rápido y seguro, ampliamente utilizado en sistemas de alto rendimiento.

En resumen, el uso de AES-GCM permite a este sistema proteger los datos de los usuarios con un enfoque completo y seguro.