Criptografía y seguridad informática

Entrega 1

GRUPO - 8305

Jordi Pérez Pastor - 100429127

Alberto Díaz-Pacheco Corrales - 100441158

Curso 2022/2023

Grado en Ingeniería Informática

Grupo - 8305
Jordi Pérez Pastor
Alberto Díaz-Pacheco Corrales

1. ¿Cuál es el propósito de su aplicación?

La aplicación **Theowall** es un **gestor de contraseñas**. Permite poder guardar las credenciales del usuario de cualquier red social, página web, aplicación de forma segura, facilitando que el usuario pueda poner contraseñas más y complejas seguras sin miedo a que se le olviden.

Por el momento el usuario puede **registrarse**, **acceder** a la aplicación, **crear** credenciales, **modificar** credenciales, **eliminar** credenciales y modificar los datos de **perfil del usuario**.

La aplicación funciona con una base de datos con archivo **.json**. En dicho archivo se guardan los datos del perfil de usuario junto con el hash de la contraseña de acceso a la app y el contenido (las credenciales) cifrado al cual solo el usuario puede acceder con su **contraseña maestra** (la contraseña de acceso a la aplicación).

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

En la aplicación, el cifrado simétrico se usa para guardar el contenido del usuario de forma segura. El archivo .json usado como base de datos tiene la siguiente estructura:

```
[
  {
    "nombre": "Alfredo",
    "apellidos": "Gonz\u00e1lez Justo",
    "usuario": "@alGJ_12",
    "salt": "ZY*HoF,zcAiX/-2h",
    "nonce": "AH~?!kN3]lPZ>@TX",
    "email": "agonzalezj@gmail.com",
    "padding": "80A4C638A5d3C91Ef1F7bFae49aEaED6bcBb3a",
    "password": "c83e1ad72443ab597288339721ec8496fcf4683c214826dd004300624143e87f",
    "contenido": [
      "b\"\\xf5\\xdb\\x1ft2\\xdd9\\x8f\\xe1\\x97\\xf0=9\\xd6\\xc4q\\xf5U\\xe0\\x03~\\xdd\\xec\\xed\\x13\\xfd\\xaa\\\\\\xd1\\x
    ]
  },
  {
    "nombre": "Jos\u00e9 Mar\u00eda",
    "apellidos": "Duro Medina",
    "usuario": "chema2000",
    "salt": "KoP@~vD*37Zn[\\W=",
    "nonce": "Io0Mm'@kpl>72;Z[",
    "email": "chemaduna@gmail.com",
    "padding": "7fB72bBEc69fCfdD1928FbB52F8FC5F1C9dC3B64E7Ca",
    "password": "fb5c4122c7069318efa86c369104ea555f58923ccffc34d5fd2ef503ef68f900",
    "contenido": [
      "b'B\\x8bJ\\x1a0\\x112\\x05\\x9a\\xa4\\xae\\x1c\\xc4W?\\x13\\xf5M4~\\xa2*J\\xa7\\xe5\\x8e\\xaf(\\x8d\\x8d|)\\x05\\x8f\\x
    ]
  }
]
```


Cada ítem en la lista es un usuario diferente. Dentro de cada usuario hay datos guardados en claro y datos cifrados u ocultos. En la base de datos se guarda la información del usuario como "`nombre`", "`apellidos`", nombre de "`usuario`", "`email`". También se guarda el "`salt`" para

la contraseña del usuario, generado automáticamente al registrarse en la aplicación; el "`nonce`" que se usa para el cifrado simétrico, también generado automáticamente en el registro de usuario; el "`padding`" para la contraseña maestra del usuario (en claro). La suma de el padding con la contraseña maestra en claro del usuario transformada a hexadecimal se usa como clave para el cifrado simétrico. También se guarda en la base de datos el resultado de la función resumen o *hash* de la contraseña maestra del usuario más el salt ("`password`").

En el apartado de "`contenido`" se guarda una lista de diccionarios, en la que cada diccionario tiene su "`id`" y la "`credencial`". Todo esto está cifrado simétricamente. Al resultado del cifrado se le suma el hash del contenido sin cifrar.

El cifrado simétrico se aplica al apartado de "`contenido`". Se ha decidido usar el algoritmo **AES256** ya que es es más seguro que AES128 al aplicar 14 número de rondas (AES128 tiene 10 rondas) y no exige un gasto de tiempo extra considerable. Además, la posibilidades de AES256 son de 2^{255}. Al no haber diferencia notable en el tiempo de procesamiento ni en consumo energético, hemos optado por una opción un poco más segura.

```python
def symmetric_encryption(data, usuario_log):
    data_bytes = bytes(data, 'utf-8')

    key = usuario_log.PASSWORD
    key_b = key.encode('utf-8')
    key_hex = key_b.hex()
    key_hex_p = key_hex + usuario_log.PADDING
    key_bytes = bytes.fromhex(key_hex_p)

    iv = bytes(usuario_log.NONCE, 'utf-8')

    cipher = Cipher(algorithms.AES256(key_bytes), modes.CTR(iv))
    encryptor = cipher.encryptor()
    ct = encryptor.update(data_bytes) + encryptor.finalize()

    return str(ct)
```

```python
def symmetric_decryption(data, usuario_log):
    data_mod = data[2:-1]
    data_bytes_x = bytes(data_mod, 'utf-8')
    data_bytes = data_bytes_x.decode('unicode_escape').encode('latin1')

    key = usuario_log.PASSWORD
    key_b = key.encode('utf-8')
    key_hex = key_b.hex()
    key_hex_p = key_hex + usuario_log.PADDING
    key_bytes = bytes.fromhex(key_hex_p)

    iv = bytes(usuario_log.NONCE, 'utf-8')

    cipher = Cipher(algorithms.AES256(key_bytes), modes.CTR(iv))

    decryptor = cipher.decryptor()
    decrypted = decryptor.update(data_bytes) + decryptor.finalize()

    decrypted_mod = eval(decrypted)

    return decrypted_mod
```

El modo de operador elegido para el cifrado simétrico es counter mode (**CTR**) ya que requiere un ***nonce*** (como vector de inicialización IV) el cual se va modificando con el *counter* y se aplica en cada ronda (no como en CBC que el vector de inicialización se aplica solo en la primera ronda). A parte, es el más recomendable hoy en día.

Como hemos explicado anteriormente, la clave para el cifrado simétrico se forma con la contraseña maestra en claro del usuario **transformada a hexadecimal** más un **"`padding`"** (número hexadecimal) personal del usuario que está guardado en la base de datos. El "`padding`" se usa para que la clave del cifrado tenga el tamaño adecuado (256 bits – 64 dígitos hexadecimales).

Dicha clave no se guarda en ningún sitio, sino que se calcula cada vez que es necesario cifrar o descifrar el "`contenido`" del usuario.

Para no estar pidiendo la contraseña maestra al usuario en cada operación, cuando el usuario inicia sesión se crea una instancia de una clase llamada *`UserVariables`* la cual contiene el nombre, los apellidos, el nombre de usuario, el número de usuario (el índice del ítem que contiene toda la información del usuario en la lista del archivo .json, el salt, el nonce, el email, el padding y la contraseña en claro para poder usarla en la clave del cifrado simétrico. Esta instancia se destruye cuando el usuario cierra la sesión, borrando así todos los datos del usuario guardados en memoria.

```python
class UserVariables:

    NOMBRE = ""
    APELLIDOS = ""
    USUARIO = ""
    NUM_USUARIO = 0
    SALT = ""
    NONCE = ""
    EMAIL = ""
    PADDING = ""
    PASSWORD = ""

    def __init__(self, data):
        self.NOMBRE = data["nombre"]
        self.APELLIDOS = data["apellidos"]
        self.USUARIO = data["usuario"]
        self.NUM_USUARIO = data["num_usuario"]
        self.SALT = data["salt"]
        self.NONCE = data["nonce"]
        self.EMAIL = data["email"]
        self.PADDING = data["padding"]
        self.PASSWORD = data["password"]
```

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

La función hash o función resumen se ha utilizado en dos contextos distintos.

En el primero, el hash se utiliza para guardar la contraseña maestra del usuario. Cuando un nuevo usuario se registra, se realiza la función hash de la contraseña y se guarda en la base de datos, de tal manera que siempre que quiera acceder a la aplicación con su usuario y contraseña se realiza la función resumen de la contraseña que el usuario teclea y se compara con el hash guardado en la base de datos. Si coinciden, el usuario accede, sino, se imprime en pantalla un error de "contraseña incorrecta".

```python
def hash_pwd(pwd):
    "Recibe la contraseña y devuelve el hash-SHA256"
    pwd_b = bytes(pwd, 'utf-8')
    hash = hashlib.sha256()
    hash.update(pwd_b)
    pwd_h = hash.hexdigest()
    return pwd_h
```

```python
def hash_msg(msg, usuario_log):
    "Recibe el mensaje y devuelve el HMAC-SHA256"
    msg_bytes = bytes(msg, 'utf-8')

    key = usuario_log.PASSWORD
    key_b = key.encode('utf-8')
    key_hex = key_b.hex()
    key_hex_p = key_hex + usuario_log.PADDING
    key_bytes = bytes.fromhex(key_hex_p)

    h = hmac.HMAC(key_bytes, hashes.SHA256())
    h.update(msg_bytes)
    msg_h = h.finalize()
    return msg_h.hex()
```

En el segundo caso, el hash (con clave) se usa para realizar un HMAC. A la hora de cifrar la información de las credenciales del usuario guardadas en el apartado de "`contenido`", se hace la función resumen del contenido con la misma clave que se usa para cifrar el contenido (contraseña maestra en claro del usuario en hexadecimal + padding personal del usuario). Así, en el apartado "`contenido`" se guarda el resultado del cifrado simétrico más el hash del contenido sin cifrar. Esto aporta **integridad** y **autenticación** a los datos cifrados, ya que al descifrar, se realiza un hash con clave de los datos descifrados y se compara con el hash con clave guardado. Si no son iguales significa que la base de datos ha sido dañada o modificada sin permiso.

4. Anexos

Además de todos los aspectos anteriores, también se ha tratado el tema de las validaciones de los parámetros editables por el usuario, en concreto sus datos personales en el perfil de usuario y a la hora de registrarse.

También hay un control de errores. En los errores previstos, se manejan imprimiendo un mensaje de error explicativo. Por ejemplo, si el usuario escribe una dirección de email inválida, se imprime en pantalla el siguiente error:

```
+ ERROR --> LA DIRRECIÓN EMAIL NO ES VÁLIDA
```

O si se ha hecho un ataque a la base de datos y se ha modificado:

```
+ ERROR --> La base de datos ha sido dañada
```

Para los errores que no están contemplados en el código, se imprime un mensaje de error en pantalla terminando el programa:

```
exceptions.Exceptions: ⚠ HA OCURRIDO UN ERROR ⚠
```

Como se puede ver en la captura, este error lo maneja la clase Exceptions. Esto permite avisar de que ha ocurrido un error, pero no da pistas sobre qué es lo que ha pasado. Así evitamos que falsos usuarios o usuarios fraudulentos encuentren fácilmente una vía de ataque.