Computer Architecture and Technology Area (ARCOS)

Criptografía y Seguridad Informática

Grado en Ingeniería Informática

Miembros del grupo:

Carlos Martin Gallardo - 100522258@alumnos.uc3m.com

Alejandro Quirante Sanz - 100522183@alumnos.uc3m.com

Grupo 82 - Grado en Ingeniería Informática

Entregable 1

Curso 2025/2026

Índice

SecureShare

Propósito y estructura interna:

Propósito de la aplicación:
SecureShare es una aplicación en Python que permite a los usuarios almacenar, cifrar y compartir archivos de forma segura.

El objetivo es garantizar la confidencialidad, integridad y autenticidad de los datos mediante criptografía simétrica, funciones de autenticación de mensajes (MAC) y derivación segura de contraseñas.
Estructura interna:

La aplicación está organizada en módulos independientes que colaboran entre sí:

Archivo	Función principal
gestion_de_usuarios.py	Registro, autenticación y derivación de claves maestras con Scrypt.
cifrado_simetrico.py	Cifrado/descifrado de archivos (AES-GCM) y generación/verificación de etiquetas HMAC-SHA256.
storage.py	Gestión de almacenamiento local de paquetes cifrados (.pkg.json) y tokens de compartición (.share.json).
main.py	Interfaz de línea de comandos (CLI) con sesión persistente. Gestiona el flujo completo: registro, login, upload, share, receive, etc.
requirements.txt	Dependencias del proyecto (solo cryptography).

Autenticación de usuarios:

Método utilizado:
La autenticación se realiza mediante contraseñas derivadas con Scrypt, un *Key Derivation Function (KDF)* seguro frente a ataques de fuerza bruta o diccionario.
Cada usuario tiene su salt aleatorio (16 bytes) y una clave maestra derivada (64 bytes) que se almacena en users.json.

Proceso:

1. Al registrarse, se genera un salt aleatorio con os.urandom(16).
2. Se aplica Scrypt(salt, n=2**14, r=8, p=1, length=64) sobre la contraseña.
3. Se guarda solo el salt y el verifier (clave derivada en base64), nunca la contraseña original.
4. En el login, se vuelve a derivar la clave y se compara con el verifier.

Algoritmo:

- Scrypt (KDF), por su resistencia a hardware paralelo (GPU/ASIC).
- Ventaja: evita almacenar contraseñas planas y dificulta ataques de diccionario.

Gestión de claves derivadas:
La clave maestra (master_key) de 64 bytes se divide en dos mitades:
- Los primeros 32 bytes → clave de cifrado (owner_key)
- Los últimos 32 bytes → clave para HMAC (hmac_key)

Cifrado simétrico (uso y gestión de claves):

Flujo de cifrado:

1. El usuario cifra un archivo con una clave de archivo aleatoria (file_key).
2. AES-GCM cifra los datos → genera ciphertext + etiqueta de autenticación GCM.
3. La file_key se cifra de nuevo con la clave de propietario (owner_key) derivada del usuario.
4. Se almacena todo en un archivo .pkg.json.

Flujo de descifrado:

- Se descifra file_key con la master key del usuario.
- Se verifica el HMAC (ver siguiente punto).
- Se descifra el archivo original con AES-GCM.

Gestión de claves:

Clave	Longitud	Generación	Uso
master_key	64 bytes	Derivada con Scrypt	Base para las otras claves
owner_key	32 bytes	master_key[:32]	Cifrar y envolver file_key
hmac_key	32 bytes	master_key[32:64]	Crear/verificar etiquetas HMAC
file_key	32 bytes	Aleatoria por archivo	Cifrar el contenido del archivo

Autenticación de mensajes (MAC):

Propósito:
Garantizar la **integridad y autenticidad** de los datos cifrados.
Si alguien modifica el ciphertext o el nonce, la aplicación detecta el cambio.

Algoritmos utilizados:

1. HMAC-SHA256: etiqueta explícita (mac) sobre nonce + ciphertext.
2. AES-GCM: modo de cifrado autenticado, que añade autenticación integrada (AEAD).

Ventajas de AES-GCM (cifrado autenticado):

- No requiere una operación HMAC separada para detectar modificaciones.

- Permite verificar integridad y autenticidad durante el descifrado.

Gestión de la clave HMAC:

- La clave HMAC (últimos 32 bytes de la master key) se genera automáticamente por usuario.
- Durante la compartición, el receptor recalcula su propia MAC para poder verificar el archivo recibido.

Flujo al compartir (resumen):

1. User_1 cifra y genera MAC con su hmac_key.
2. User_2 recibe el archivo, lo “reenvuelve” con su propia owner_key y recalcula el MAC con su hmac_key.

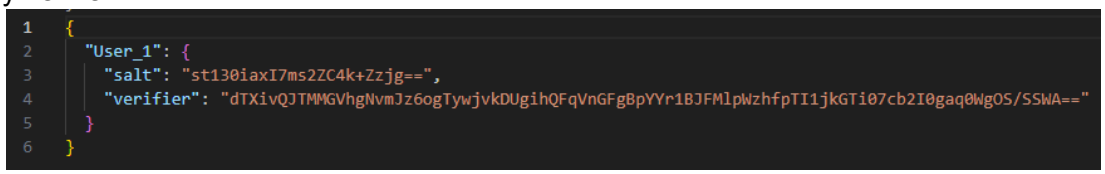
Pruebas:

1. Preparación inicial

Asegurarse que estamos en el archivo y activamos el entorno virtual para poder ejecutar bien el código. Ejecuta:
cd path\to\secure_share
.\venv\Scripts\activate

2. Registro y login:

Ejecuta:
python main.py register User_1 mi_contraseña_segura
El programa debe responder:
[Registro] Usuario 'User_1' registrado.
y crear un archivo llamado users.json (si no existia ya) y añadir el User_1 con su salt y verifier.



Si intentas registrar el mismo usuario otra vez:
python main.py register User_1 mi_contraseña_segura
El programa te responde:
Error registrando: Usuario ya existe
Para iniciar sesión ejecuta:
python main.py login User_1 mi_contraseña_segura
El programa te responde:
[Autenticación] Usuario 'User_1' autenticado correctamente.
[Sesión] Guardada sesión activa de 'User_1'.
Sesión iniciada como User_1
Y crea otro archivo session.json (si no existia ya) y añade el User_1 con su verifier.

Si intentas hacer el login con una contraseña incorrecta.
python main.py login User_1 clave_incorrecta
Te responde:
[Autenticación] Usuario no encontrado.

Login fallido: Autenticación fallida
Y si intentas iniciar sesion con un usuario no registrado:
python main.py login User_2 mi_contraseña_segura
Te responde:
[Autenticación] Usuario no encontrado.
Login fallido: Autenticación fallida

3. Cifrado (upload)

Deberia venir el archivo mensaje.txt (pero sino crearlo con un mensaje cualquiera dentro com esto:

```
Este es un mensaje secreto que solo User_1 puede leer.
```

Ejecutar:
python main.py upload mensaje.txt
Te responde:
[Storage] Archivo guardado como storage\User_1\mensaje.txt.pkg.json
Y crea una carpeta llamada storage (si no esta creada ya) dentro una carpeta llamada user_1 y dentro un archivo llamado mensaje.txt.pkg.json que contiene lo siguiente:


```
{
  "enc_nonce": "OiI+sXePKRTgSxrK",
  "ciphertext": "LoFxZRMzxsAcrQBJL8xjUWQ6WJHOYPyDcVqE4nU3wsy/Qn7dvhuDXBfPiBt87IVVtqHBznHK4AElk/moc6GozJGFl5HSKFuK",
  "wrap_nonce": "unlkNr+btcLKHESE",
  "wrapped_filekey": "QjPEEeAUUwwfTrvAIKgCv6Wb6fTgfS7lJV4dTUMyII/WTnlwhRBMkNXFkgnYe82s",
  "mac": "7XF9kK27J/ZF4IgwIaLY3Ck514c0NvZZzNZvhCyz2aI="
}
```


4. Listado de archivos

Si ejecutas:
python main.py list
Te responde:
Archivos de User_1
 - mensaje.txt

5. Descifrado (download)

Si ejecutas:
python main.py download mensaje.txt
Te responde:
Archivo descifrado guardado en downloads\mensaje.txt
Y crea la carpeta downloads (si no existe) y dentro mete el archivo mensaje.txt ya desencriptado como el original.

6. Compartir (share)

Si ejecutas:
python main.py share mensaje.txt "clave_compartida" --token-name token_mensaje
Te responde:
[Storage] Token de compartición guardado en storage\User_1\token_mensaje.share.json
Token creado. Entrega token + passphrase al destinatario.
Y añade el token_mensaje.share.json a storage de User_1 que contendrá el “token” cifrado con la passphrase "clave_compartida":

```
{
  "salt": "e8yl/POkBY/a+J1t6LAfig==",
  "nonce": "r2oIsl/V8iDvtClD",
  "wrapped_filekey": "wV2qkWzKmkJ5OwLSSne/fZoCLtuakKk5AvJdUYZoDl9/UOwgd4EHlzus+p4gRRGX"
}
```


Ahora habría que registrar a otro usuario e iniciar sesión como el:
python main.py register User_2 su_contraseña_segura
python main.py login User_2 su_contraseña_segura
Y simulamos que User_2 tiene el token del mensaje porque User_1 se lo ha dado.
Ejecutamos:
python main.py receive "clave_compartida" --token-path storage\User_1\token_mensaje.share.json --pkg-path storage\User_1\mensaje.txt.pkg.json --save-as mensaje_de_User_1
Te responde:
[Storage] Archivo guardado como storage\User_2\mensaje_de_User_1.pkg.json
[Receive] Archivo recibido y guardado como 'mensaje_de_User_1'.
[Receive] Nuevo HMAC recalculado correctamente para el receptor.
Y se añade el mensaje_de_User_1.pkg.json en la carpeta User_2 de storage (si no existía esa carpeta se crea).


```
{
  "enc_nonce": "OiI+sXePKRTgSxrK",
  "ciphertext": "LoFxZRMzxsAcrQBJL8xjUWQ6WJHOYPyDcVqE4nU3wsy/Qn7dvhuDXBfPiBt87IVVtqHBznHK4AElk/moc6GozJGFl5HSKFuK",
  "wrap_nonce": "5gaW5qGTkOiNYjAI",
  "wrapped_filekey": "UZxJ98wJV13cwgeA2Bqb10D2DVGtpUUTDo5Ipgiy4KalgcHMB5vRxXmebso5dH6M",
  "mac": "52l9PoQptsD9YXv0Ks+aP8esc3VqD36JwQ13W8A0MMI="
}
```


Ahora User_2 puede desencriptar el mensaje.
Ejecutar:
python main.py download mensaje_de_User_1
Te responde:
Archivo descifrado guardado en downloads\mensaje_de_User_1
Y mensaje_de_User_1 se crea en downloads con el mensaje desencriptado exactamente igual que mensaje.txt.

```
Este es un mensaje secreto que solo User_1 puede leer.
```

7. Pruebas de integridad y seguridad

Si cambias lo que sea del mac de cualquier mensaje encriptado al hacer el download este deberia fallar.
Cambiar en storage\User_2\mensaje_de_User_1.pkg.json el mac, por una mac distinta como por ejemplo la de mensaje.txt.pkg.json.
Ejecutar:
python main.py download mensaje_de_User_1
Te responde:
Error descargando: MAC inválida o datos alterados.

8. Logout

Para cerrar sesion simplemente.
Ejecutar:
python main.py logout
Te responde:
[Sesión] Cerrada y eliminada.
Y elimina el session.json.