SecureSend

Privacidad y Seguridad

Grupo: 82

ID de grupo de prácticas:

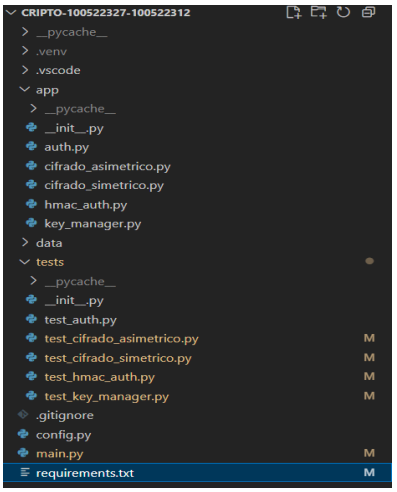
Nombre de todos los alumnos: Mario Agúndez Díaz / Jorge Condado Carballo

Correo de todos los alumnos:
100522312@alumnos.uc3m.es / 100522327@alumnos.uc3m.es

Repositorio de código (enlace) si lo hubiera:
https://github.com/100522327/Cripto-100522327-100522312.git

Índice

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



SecureSend es una aplicación de escritorio (o una aplicación web simple) que permite a profesionales (como abogados, consultores o médicos) almacenar y compartir documentos sensibles de forma segura con sus clientes. La aplicación garantiza la **confidencialidad**, **integridad** y el **no repudio** de los documentos compartidos.

Esta es nuestra estructura interna:

1. Directorio data/
Guarda todos los datos que la aplicación necesita persistir, separados del código fuente. Incluye pki/ (infraestructura de clave pública), ca_root/ y ca_sub/ (certificados y claves de las autoridades raíz y subordinada), user_certs/ (certificados de usuarios), users/ (datos y claves de cada usuario) y documents/ (documentos cifrados).

2. Directorio app/
Contiene la lógica de negocio en módulos. auth.py gestiona registro e inicio de sesión; cifrado_simetrico.py y cifrado_asimetrico.py manejan cifrado AES y RSA; digital_signature.py crea y verifica firmas digitales; hmac_auth.py controla la integridad con HMAC; pki_service.py administra certificados y autoridades; key_manager.py gestiona el almacenamiento seguro de claves; y ui.py maneja la interfaz de usuario.

3. Directorio tests/

En él se encuentran todos los test que verifican el correcto funcionamiento de los archivos de app/. Estos test son: tes_auth.py, test_cifrado_simetrico.py, test_cifrado_asimetrico.py, test_key_manager.py y test_hmac_auth.py; los cuales son expuestos más adelante en la declaración de pruebas

4. Archivos en la raíz
main.py es el punto de entrada que orquesta la aplicación. config.py define constantes y rutas de configuración. requirements.txt lista las dependencias necesarias para instalar con pip install -r requirements.txt.

¿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.

El sistema de autenticación usa el método clásico de **usuario y contraseña**.

En el **registro**, se genera un **salt único** que se combina con la contraseña mediante **PBKDF2-HMAC-SHA256**, obteniendo un **hash** que se almacena junto al nombre de usuario, pero **nunca la contraseña original**.

En el **inicio de sesión**, el sistema repite el proceso con la contraseña introducida y compara el resultado con el hash guardado, usando una **comparación segura** para evitar ataques de temporización.

El uso de **PBKDF2** con **600 000 iteraciones** y **salts únicos** protege contra ataques de fuerza bruta y tablas arcoíris. Además, la clave derivada puede emplearse para **cifrar la clave privada del usuario**, haciendo que la seguridad del sistema dependa directamente de la contraseña.

¿Para qué utiliza el cifrado simétrico/ asimétrico o ambos? ¿Qué algoritmos ha utilizado y por qué? ¿Cómo gestiona las claves? Explique los mismos aspectos si se utiliza cifrado asimétrico para este tipo de cifrado.

Hemos decidido emplear un cifrado híbrido para asegurar los documentos, aprovechando la velocidad del cifrado simétrico y la seguridad del intercambio de claves del cifrado asimétrico.

Cifrado Simétrico

Con el cifrado simétrico nos aseguramos de proteger la confidencialidad del documento completo por su alta eficiencia al manejar grandes volúmenes de datos, como archivos. El algoritmo empleado es AES-256 en modo GCM (Galois/Counter Mode). AES es el algoritmo de cifrado por bloque estándar actual, recomendado por no estar comprometido. El modo GCM es crucial, ya que, proporciona cifrado autenticado, lo que aborda simultáneamente los requisitos de confidencialidad e integridad/autenticidad.

Cifrado Asimétrico

El cifrado asimétrico se utiliza para la distribución segura de las claves simétricas (AES y HMAC) que se generan por cada documento. Es inviable que los usuarios introduzcan claves de gran longitud, por lo que la clave simétrica generada se cifra con la clave pública RSA del destinatario (encrypted_sym_key y encrypted_hmac_key).

Gestión de Claves

Claves Simétricas (AES/HMAC): Se genera una clave simétrica aleatoria única por cada documento subido. Esta clave no se almacena directamente, sino que se cifra con la clave pública RSA del usuario y el resultado cifrado se almacena en los metadatos del documento.

Claves Asimétricas (RSA): El par de claves asimétricas (pública y privada) se genera por usuario. La clave pública se almacena en un archivo PEM sin cifrar, ya que es información pública. La clave privada se almacena en formato PKCS8 y está protegida por la contraseña de inicio de sesión del usuario, la cual se requiere cada vez que se carga la clave privada para descifrar un documento. Esta práctica asegura que el acceso a la clave privada esté protegido y se cumpla con el requisito de protección de claves.

¿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.

Hemos utilizado dos mecanismos para la autenticación: HMAC y Cifrado Autenticado (AES-GCM). El HMAC (Hash-based Message Authentication Code) se aplica al documento CIFRADO para generar una etiqueta (hmac_tag) que se almacena junto a los metadatos. Esto permite al receptor verificar si el documento cifrado ha sido modificado o si proviene de una fuente legítima con la clave HMAC correcta.

Los algoritmos que hemos utilizado son:

- HMAC-SHA256: Es un estándar para códigos de autenticación que utiliza una clave secreta y una función *hash* criptográfica. Se utiliza SHA-256 como función *hash* base por ser robusta y actual. La clave tiene una longitud de 32 bytes (256 bits) y la verificación del HMAC se realiza mediante una comparación segura de *digests* (secrets.compare_digest) para prevenir ataques de temporización.
- Cifrado Autenticado (AES-GCM): El cifrado simétrico ya utiliza AES-GCM. GCM es un modo de operación de cifrado que, por diseño, genera un *tag* de autenticación además del texto cifrado, lo que proporciona el servicio de autenticación directamente.

Gestión de Claves HMAC

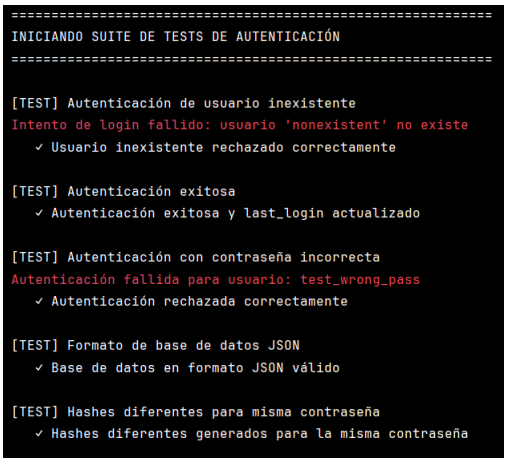
- Generación de Clave: Se genera una clave HMAC aleatoria única de 256 bits por documento, similar a la clave AES.
- Protección y Almacenamiento: Esta clave, que es secreta, tampoco se almacena directamente. En su lugar, se cifra con la clave pública RSA del usuario (encrypted_hmac_key) y se guarda en el archivo de metadatos del documento. Esta protección asimétrica asegura que solo el dueño de la clave privada pueda acceder a ella.

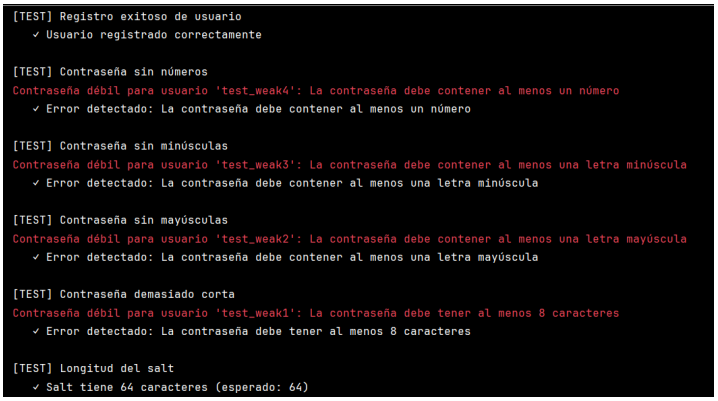
Además, el sistema utiliza algoritmos de cifrado autenticado mediante el modo AES-GCM.

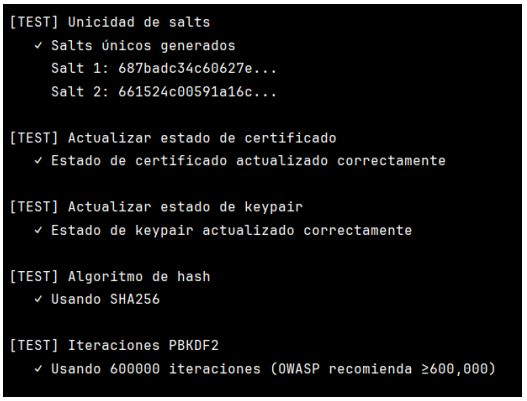
La ventaja principal es que garantiza tanto la confidencialidad (el contenido está cifrado) como la autenticidad y la integridad (cualquier modificación a los datos cifrados o al *tag* de autenticación hará que el descifrado falle) en una sola operación criptográfica. Esto elimina riesgos de seguridad y errores de implementación (como un HMAC aplicado incorrectamente)

Pruebas realizadas para garantizar la calidad del código

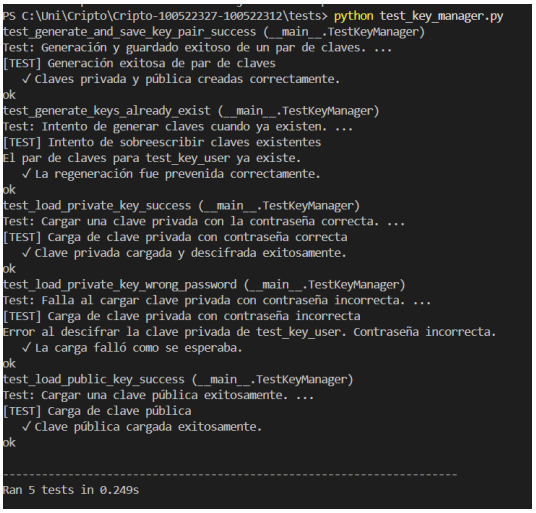
Para el registro y autenticación de usuario se han realizado las siguientes pruebas



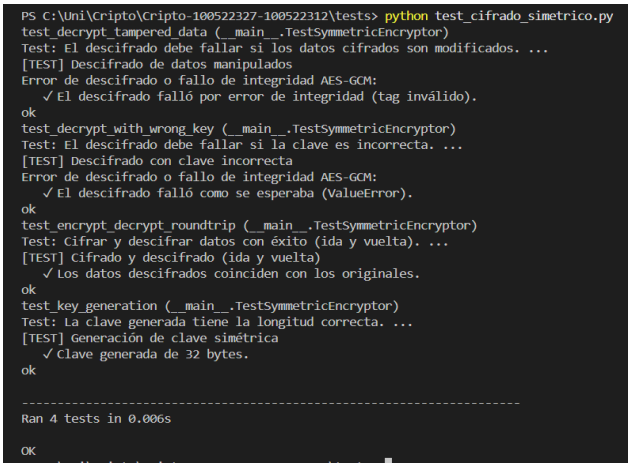


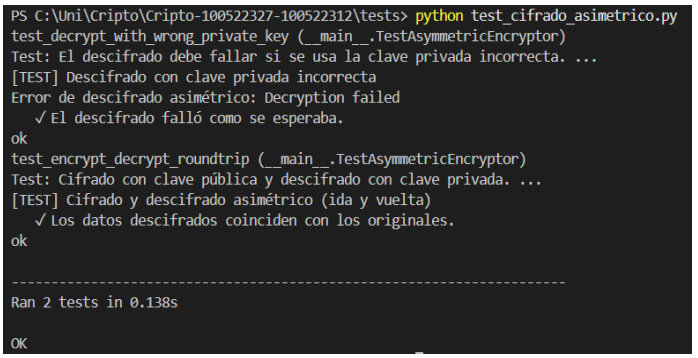


Para la generación y gestión de claves hemos realizado los siguientes tests (test_key_manager.py)

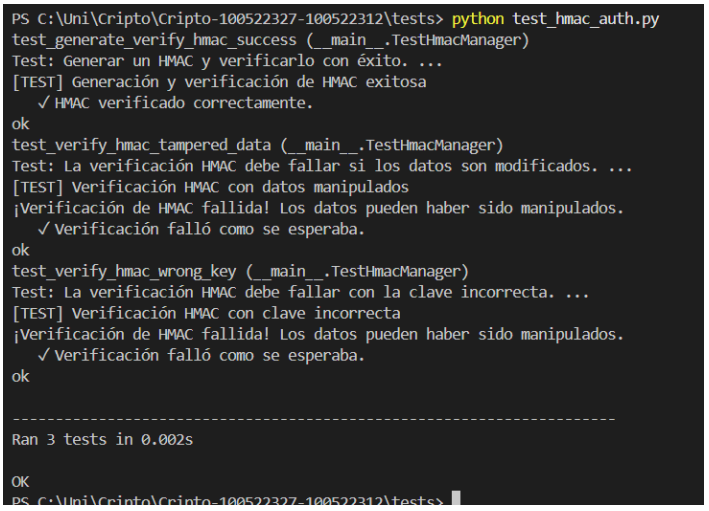


Para el cifrado simétrico hemos comprobado los siguientes 4 tests y para el asimétrico hemos comprobado qué pasa si se usa la clave privada incorrecta y si los datos descifrados coinciden con los originales:





Finalmente, para comprobar el buen funcionamiento de la creación de HMAC y su verificación, hemos realizado los siguientes tests:



Declaración de IA

Hemos utilizado la inteligencia artificial para ayudarnos en diversos aspectos del proyecto: nos recomendó la mejor estructura a seguir, explicó y sugirió librerías útiles para el desarrollo, asistió en la programación de los tests para no omitir ninguno, ayudó a corregir errores del código y fallos en las pruebas, colaboró en la configuración del archivo *config.py*, contribuyó a comentar el código y nos apoyó en la redacción y mejora del texto.