MEMORIA

Grupo: 84
ID de grupo de prácticas: 17
Nombre de todos los alumnos: Iker Mirón Blázquez, Hugo Oropesiano Valadés
Correo de todos los alumnos: 100522189@alumnos.uc3m.es , 100522384@alumnos.uc3m.es
Repositorio de código (enlace) si lo hubiera:
https://github.com/IkerMironBlazquez/criptografia_100522189_100522384.git

ÍNDICE

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

La aplicación que hemos desarrollado es una pequeña red social de mensajería y quedadas de perros. Principalmente está dirigida a usuarios que tengan perros y quieran buscar otras personas con perros para organizar quedadas y que tanto sus mascotas como ellos mismos puedan conocerse y hacerse amigos. Los usuarios de la aplicación podrán registrar tantos perros como deseen con los datos correspondientes y podrán ver los perros de otros usuarios. Si encuentran uno, por ejemplo, de la misma raza que el suyo, y les gustaría que se conociesen y jugasen juntos, pueden enviar un mensaje a su dueño para contactar con él y proponerlo.

La estructura interna de nuestra aplicación es muy sencilla: tenemos 4 scripts de Python:

main.py: este script es el núcleo de la interfaz y orquesta el resto de módulos. Muestra un menú inicial para registrar usuarios, iniciar sesión, ver información del sistema y salir. Tras iniciar sesión permite registrar perros, ver tus perros, borrar perros, explorar perros públicos para contactar con sus propietarios, enviar mensajes (cifrados automáticamente), ver mensajes, eliminar la cuenta (borrado en cascada de perros y mensajes asociados a dicha cuenta) y cerrar sesión.
main.py también configura el logging (ficheros en carpeta logs, limpieza al inicio y handlers) y crea las instancias de UsuarioManager, PerroManager y MensajeManager, los cuales se encargarán de la gestión de datos y el cifrado. Al arrancar comprueba que el sistema de cifrado (AES-256-GCM) funciona. El almacenamiento y manejo de datos se realiza mediante JSON (usuarios, perros, mensajes y claves de sistema) gestionado por los managers.

autenticación.py: este script se encarga de todo lo relativo a los usuarios de la aplicación: crear cuentas, comprobar que las credenciales sean correctas y eliminar cuentas.
Se compone de 2 clases:

Usuario (id, nombre, hash de la contraseña, email opcional y fecha de registro), que tiene los métodos to_dict, utilizado para convertir objetos Usuario a diccionarios y poder guardarlos en JSON, y from_dict, para sacar de JSON y pasar de diccionario a un objeto Usuario.

UsuarioManager, que gestiona el registro y la autenticación de usuarios de forma segura. Posee los métodos validar_contraseña_robusta, para verificar que la contraseña cumple con los criterios establecidos; cargar_usuarios para sacar de JSON; guardar_usuarios para meter a JSON; hash_contraseña para generar un hash seguro de la contraseña; registrar_usuario para registrar a un nuevo usuario (este método llama a los anteriormente descritos); y autenticar_usuario para verificar que las credenciales son correctas y guardar el usuario (este método llama a verificar_contraseña para comprobar que la contraseña coincide con el hash almacenado)

Además, UsuarioManager también posee otros 2 métodos extra: listar_usuarios, que muestra los nombres de los usuarios registrados (principalmente utilizado para realizar pruebas) y borrar_usuario, el cual elimina totalmente el usuario correspondiente, así como sus perros y sus mensajes (este método utiliza PerroManager y MensajeManager para eliminarlos)

clases.py: este script contiene el resto de clases:
Perro: guarda los datos de un perro (nombre, identificador como microchip, descripción, fecha) y tiene funciones para validar el microchip y generar un id público para mostrar en la app. También tiene los métodos to dict y from dict.
Mensaje: representa un mensaje entre dos usuarios con su texto (antes y después de ser cifrado), la fecha de envío y su estado de leído/no leído. Tiene los métodos to dict y from dict.
PerroManager posee algunos métodos similares a los de UsuarioManager: cargar y guardar perros, registrar perros y borrar perros (puede ser llamado desde eliminar usuario). En adición a estos métodos, tenemos otros para obtener los perros del usuario, los perros públicos de otros usuarios, y para obtener un perro mediante su ID.
MensajeManager también presenta una estructura equivalente. Posee sus correspondientes cargar y guardar mensajes, obtener mensajes de un usuario, borrar mensajes de un usuario, y otros nuevos como marcar como leído. Además, este manager se encarga de verificar que el cifrado y autenticado de mensajes con AES-256-GCM se ha realizado correctamente, y obtiene algunas estadísticas sobre el cifrado de los mensajes, como pueden ser el total de mensajes cifrados y el porcentaje de mensajes que han sido cifrados.

criptografia.py: este script gestiona la criptografía de la aplicación. La clase CriptografiaManager genera/lee la clave AES-256 del sistema, crea nonces únicos para GCM, cifra y autentica mensajes con AES-256-GCM y convierte los resultados a un formato adecuado para almacenamiento en JSON. También descifra mensajes desde ese formato y ofrece una comprobación de integridad para verificar que las operaciones de cifrado/descifrado realizadas funcionan correctamente.

Por último, tenemos 4 archivos JSON para el almacenamiento de datos: **usuarios.json, perros.json, mensajes.json** y **claves sistema.json.**

¿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

Para autenticar usuarios, comenzamos con el registro: la aplicación valida la robustez de la contraseña y, si es válida y el nombre de usuario no existe, la contraseña se transforma en un hash seguro utilizando **bcrypt.hashpw** y se guarda junto al resto de datos en usuarios.json. Posteriormente, a la hora de iniciar sesión, la aplicación busca el usuario por su nombre y compara la contraseña introducida con el hash que hay almacenado en JSON

utilizando **bcrypt.checkpw.** Si la comprobación es correcta, queda confirmado que el usuario está autenticado.

El algoritmo utilizado es **Bcrypt.** Lo implementamos mediante la librería del mismo nombre. Hemos decidido utilizar Bcrypt porque es un algoritmo diseñado específicamente para conseguir un almacenamiento de contraseñas totalmente seguro. Además, incorpora salt aleatorio e incorpora un factor de coste ajustable: esto significa que en base al coste, tendremos un crecimiento exponencial de las rondas, por lo que será muy resistente a ataques de fuerza bruta o ataques masivos.

En cuanto al almacenamiento, la contraseña de los usuarios nunca se guarda como texto en claro, siempre se almacena la contraseña hasheada por bcrypt.
No se genera ningún tipo de clave a partir de las contraseñas. Todas las claves de cifrado que se utilizan se generan independientemente en criptografia.py, por lo que funciona en paralelo a bcrypt y la seguridad de contraseñas.

¿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.

La aplicación utiliza únicamente el cifrado simétrico mediante el algoritmo AES-256-GCM, este algoritmo simétrico se utiliza para diferentes cosas en nuestra aplicación:

Se usa para la protección de mensajes privados: cada uno de los mensajes intercambiados entre los usuarios se cifra automáticamente, después estos mensajes se almacenan en mensajes.json ya cifrados.
Gracias a este cifrado se garantiza que solo los usuarios que hayan enviado o recibido un mensaje sean los únicos que pueden leer el contenido del mismo, protegiendo así la información real de los mensajes.
Los datos sensibles se guardan cifrados en el sistema de archivos, con esto conseguimos que aunque se acceda fácilmente a los archivos se siga protegiendo la información.

Hemos seleccionado este algoritmo atendiendo a diferentes razones:

Este algoritmo lleva mucho tiempo en funcionamiento resistiendo numerosos ataques y está optimizado perfectamente para procesadores modernos. Se usan 256 bits ya que así conseguimos mayor longitud de clave lo que resiste mejor los ataques a fuerza bruta.
Hemos seleccionado el modo de operación GCM ya que este permite cifrar los mensajes y al mismo tiempo generar una etiqueta de autenticación. Con esto además de proteger la integridad del mensaje el sistema puede detectar si el mensaje ha sido manipulado o modificado.
GCM es uno de los modos más seguros actualmente utilizados por diversos organismos internacionales (NIST, industria bancaria).

Aparte de esto, GCM es muy rápido y está optimizado para procesadores modernos, ya que permite cifrar y descifrar mensajes sin padding, lo que lo hace ideal para mensajes cortos como lo son los de nuestra aplicación. También al utilizar GCM el código es más sencillo y más difícil cometer errores ya que no hay que gestionar la autenticación por separado.
Con esto podemos ver que GCM cumple tanto la parte de cifrado simétrico como la de autenticación de mensajes y es ideal para nuestra aplicación. No se usa cifrado asimétrico ya que la aplicación no requiere intercambio de claves públicas ni comunicación entre sistemas externos por lo que el cifrado simétrico es suficiente y más eficiente para nuestro caso.

La aplicación gestiona las claves de la siguiente manera:

Al iniciar la aplicación por primera vez, se genera una clave AES-256 usando un generador de números aleatorios. También se genera un salt de 16 bytes para posibles futuras derivaciones de clave.
Estas claves se guardan en el archivo claves sistema.json en formato Base64 lo que permite almacenarlas como texto en JSON.
Cada vez que la aplicación se carga se comprueba si el archivo de claves está correcto y si existe, si está todo correcto se reutiliza la clave para cifrar y descifrar todos los mensajes. Si el archivo no es correcto la aplicación lo regenera automáticamente para garantizar la seguridad.
La clave permanecerá constante mientras el archivo exista, si se elimina el archivo se genera una nueva clave y los mensajes antiguos ya no podrán descifrarse, así protegemos la confidencialidad si el sistema se elimina.

¿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.

La aplicación utiliza AES-256-GCM lo que es un algoritmo de cifrado autenticado que tiene las siguientes ventajas:
Se protege el contenido del mensaje y se garantiza que el contenido del mismo no ha sido modificado, si el mensaje cifrado se altera, el sistema lo detecta automáticamente y rechaza el descifrado.
Se combina cifrado y autenticación en una sola operación con esto se elimina la utilización de HMAC lo que lo hace más rápido que usa otros tipos de modos de operación que habíamos pensado en utilizar como AES-CBC y autenticación (HMAC-SHA256) por separado.
Todo esto es idóneo para nuestra aplicación ya que al combinar el cifrado con la autenticación en una sola operación lo hace más rápido y reduce la complejidad del código y además protegemos nuestra aplicación de ataques de manipulación y padding.
La aplicación utiliza códigos de autenticación de mensajes MAC para garantizar la integridad y autenticación de los mensajes cifrados.

Con esto protegemos contra modificaciones ya que si un atacante intenta alterar el contenido cifrado el sistema detecta la manipulación y rechaza el mensaje. Esto está directamente relacionado con el modo de operación GCM que genera una etiqueta de autenticación (MAC) durante el cifrado, esta etiqueta se verifica automáticamente al descifrar así aseguramos que los mensajes no hayan sido manipulados.

La gestión de claves ha sido explicada en el apartado anterior ya que es un aspecto común tanto para el cifrado simétrico como para la autenticación de mensajes. En ambos casos, la clave AES-256 generada y almacenada en claves sistema.json es la misma y su gestión sigue igual.

Indique las pruebas realizadas para garantizar la calidad del código

Para garantizar la calidad del software, hemos realizado múltiples pruebas, que han sido las siguientes, todas ellas utilizando de forma normal la aplicación y explorando todas las opciones que ofrece:

-Inicio de sesión con un usuario no registrado.
-Inicio de sesión con contraseña incorrecta.
-Registro de un usuario ya existente.
-Registro con contraseña no robusta.
-Registro correcto con o sin email, ya que es opcional.
-Cerrar sesión
-Salir de la aplicación y volver a iniciar sesión con el usuario anteriormente registrado.
-Ver perros sin perros registrados.
-Registrar un perro y ver perros.
-Registrar un perro y eliminarlo.
-Ver perros públicos sin que haya.
-Ver perros públicos y que sí haya.
-Introducir un ID de perro incorrecto a la hora de contactar con su propietario.
-Introducir un ID correcto y enviar un mensaje al otro usuario.
-Revisar el estado de los mensajes.
-Cambiar un mensaje al estado leído.
-Registrar 2 perros, eli minar uno y después borrar la cuenta para ver que tanto el usuario como el perro restante y los mensajes que tiene ese usuario se han eliminado.
-Dejar campos vacíos (a excepción de email).
-Seleccionar una opción no disponible.
-Ver información del sistema.

Todas estas pruebas se han realizado revisando tanto logs como JSON para ver que la aplicación opera con éxito para cada caso.

Además, se han realizado pruebas enfocadas en el cifrado y la autenticación. Estas pruebas no aparecen en la versión final de la aplicación ya que en ellas se modifica el código y se fuerzan errores.