Entrega 1

Práctica Criptografía y Seguridad Informática

Miguel Jimeno Casas - 100495932 - 100495932@alumnos.uc3m.es

Hector Herráiz Diez - 100499734 - 100499734@alumnos.uc3m.es

Grupo 82 - Ingeniería Informática

ÍNDICE

PROPÓSITO DE LA APLICACIÓN

Propósito de PadelApp

Nuestra aplicación PadelApp tiene como propósito ofrecer una plataforma diseñada para gestionar reservas de pistas de pádel de manera segura y sencilla. Permite a los usuarios autenticarse mediante un sistema de inicio de sesión, registrarse si son nuevos o recuperar su cuenta en caso de haber olvidado la contraseña. Incluye funcionalidades para seleccionar una hora y pista disponibles para jugar, proporcionando una interfaz intuitiva para facilitar las reservas. También cuenta con opciones de seguridad, como la verificación por PIN para el restablecimiento de contraseñas, y una función para enviar mensajes a otros usuarios dentro de la app, lo que promueve la comunicación y organización entre los jugadores. La aplicación garantiza la seguridad de la información con medidas de cifrado para proteger los datos de los usuarios.

Estructura Interna

- `user.py` crea clase `User` facilita la gestión de usuarios en la aplicación, conectándose a una base de datos SQLite y utilizando funciones de cifrado para asegurar contraseñas. Su método `register_user` verifica la seguridad de la contraseña y guarda la información del usuario en la base de datos junto con una clave privada y pública para el cifrado. El método `sign_in` permite el inicio de sesión, utilizando `authenticate_user` para verificar la contraseña comparando su hash con el almacenado. `check_username` confirma si un nombre de usuario existe en la base de datos, mientras que `hash_password` y `generate_salt` se encargan de cifrar la contraseña de forma segura.

- `recuperar_contraseña.py` define la clase `RecuperarContrasena`, que facilita la recuperación de contraseñas de usuarios en una aplicación. Esta clase permite enviar un código PIN de verificación mediante SMS al número de teléfono registrado del usuario, utilizando la API de Twilio para asegurar que el número de teléfono está vinculado al usuario adecuado. También verifica si un número de teléfono está registrado para un usuario específico (`is_phone_registered`) y permite actualizar la contraseña en la base de datos. Para el cambio de contraseña, `update_password` obtiene la "salt" original de la contraseña guardada, hashea la nueva contraseña y la almacena en la base de datos, garantizando que solo los usuarios autenticados puedan recuperar su acceso de manera segura.

- `reserve_court.py` define la clase `ReserveCourt`, que permite gestionar la reserva de pistas en una aplicación deportiva. Su método principal, `reserve_court`, permite a los usuarios reservar una pista en una hora específica tras varias validaciones. Primero, verifica que el usuario no haya realizado una reserva previa y que la combinación de hora y pista esté disponible. Además,

confirma la validez de la tarjeta de crédito proporcionada usando el algoritmo de Luhn (`algoritmo_luhn`) para evitar datos incorrectos. Si las verificaciones son exitosas, registra la reserva en la base de datos.

- `message.py` implementa la clase `Message`, destinada a manejar el envío y recuperación de mensajes cifrados en la aplicación. A través de `send_message`, los mensajes se cifran con claves RSA y se almacenan en la base de datos de manera segura, impidiendo que el contenido sea legible sin descifrarlo. El método asegura que el remitente y el destinatario sean diferentes y guarda el mensaje cifrado para ambos usuarios. Además, la clase ofrece métodos como `get_messages_receiver` y `get_messages_sender` para recuperar mensajes cifrados enviados o recibidos por un usuario específico.

- `dataBase.py` define la clase `dataBase`, que administra la creación y manipulación de las tablas de la base de datos para usuarios, reservas y mensajes en la aplicación. En su método `create_table`, esta clase configura las tablas `users`, `reservations`, y `messages`, asegurando que cada una tenga restricciones para mantener la integridad de los datos. Además, incluye métodos genéricos como `execute` y `fetchone` para ejecutar consultas SQL parametrizadas y obtener resultados de una sola fila, respectivamente. Finalmente, el método `close` permite cerrar la conexión de la base de datos de forma segura al finalizar la sesión.

- `crypto.py` ofrece la clase `Crypto` que se compone de funciones criptográficas para cifrar y descifrar mensajes, así como para la autenticación de los mismos. Se incluyen métodos para generar y manejar claves RSA, cifrar y descifrar claves privadas, y verificar la autenticidad de los mensajes mediante HMAC.

  Métodos Clave:
  `generate_rsa_keys`: Genera un par de claves RSA, devolviendo la clave pública en formato PEM y la clave privada sin cifrar.
  `cipher_private_key`: Cifra la clave privada utilizando la contraseña del usuario.
  `decipher_private_key`: Descifra la clave privada utilizando la contraseña correspondiente.
  `generate_hmac_key`: Genera una clave HMAC utilizando PBKDF2 con una sal aleatoria.
  `create_hmac` y `verify_hmac`: Permiten crear y verificar la autenticación de mensajes usando HMAC.
  `encrypt_message_rsa`: Cifra un mensaje para un usuario específico utilizando su clave pública y genera un HMAC para autenticación.

`decrypt_rsa`: Descifra un mensaje cifrado, valida su autenticidad con HMAC y utiliza la clave privada del usuario.
`insert_keys`: actualiza la base de datos con la clave pública y la clave privada cifrada del usuario.

- `main.py` actúa como el punto de entrada principal de la aplicación, donde se inicializan las conexiones a la base de datos y se instancian las clases principales para gestionar usuarios, reservas de pistas, recuperación de contraseñas, y mensajería. Su flujo general incluye:
    1. **Inicialización de la Base de Datos:** Se crea o conecta a una base de datos SQLite y se instancian las tablas mediante la clase `dataBase.`
    2. **Registro y Autenticación de Usuarios:** Permite a los usuarios registrarse, iniciar sesión y gestionar sus datos de autenticación a través de la clase `User`.
    3. **Recuperación de Contraseñas:** Facilita la recuperación de contraseñas mediante el envío de códigos PIN por SMS y actualización de la contraseña usando la clase `RecuperarContrasena`.
    4. **Reserva de Pistas de Pádel:** Gestiona la reserva de pistas asegurando disponibilidad y validación de pago usando la clase `ReserveCourt`.
    5. **Mensajería entre Usuarios:** Permite enviar y recibir mensajes cifrados entre usuarios utilizando la clase `Message` y las funciones de cifrado de `Crypto`.

    Finalmente, `main.py` coordina estas operaciones para permitir una experiencia completa en la aplicación, e inicializa la interfaz gráfica de usuario para el sistema de registros de usuarios, reservas y mensajería. La interfaz, que utiliza es Tkinter, permite la interacción directa del usuario con el sistema, facilitando la autenticación, la gestión de reservas y el envío de mensajes.

REGISTRO Y AUTENTICACIÓN DE USUARIOS

Autenticación del usuario

La interacción con un usuario tiene lugar en el método `authenticate_user,`(el cual se llama a la hora de iniciar sesión) de la clase `User`. El método en cuestión realiza las acciones que se enumeran a continuación

Búsqueda de Salt de Usuario: Toma un nombre de usuario como entrada y trata de encontrar la contraseña y la salt asociado al mismo, que está almacenada en la base de datos.

Contraseña hash del usuario: Esta contraseña de usuario autenticado se hashea luego con el mismo algoritmo (`hash_password`) así como con la salt

recuperado.Comparación hash: El nuevo hash creado se compara luego con el hash de la base de datos para la contraseña. Si ocurre tal situación, valida al usuario; de lo contrario, la aplicación informa sobre una contraseña incorrecta o que no existe tal cuenta de usuario.

Algoritmos utilizados

PBKDF2 con **HMAC-SHA256:** el hash de la contraseña se deriva utilizando **PBKDF2HMAC** con **SHA256** como función hash. **PBKDF2** es un algoritmo común utilizado para almacenar contraseñas de forma segura, ya que agrega seguridad a través de un proceso de derivación que dificulta los ataques de fuerza bruta. En este caso se realizan 100.000 iteraciones, lo que aumenta el tiempo necesario para verificar cada contraseña y la hace más resistente a ataques.

Generación de salt: se genera un valor aleatorio o salt usando os.urandom para garantizar que dos contraseñas iguales generen hashes diferentes. Esto protege contra ataques de diccionario y ataques de colisión en los que un atacante intenta adivinar el hash de una contraseña común.

Gestión de contraseñas de los usuarios registrados

La contraseña la introduce el usuario a la hora de registrarse en la aplicación. Es introducida de una manera clara y concisa, ya que ese 'str' pasa una serie de filtros para verificar su seguridad (mínimo de 8 caracteres, inclusión de mayúsculas, minúsculas, dígitos y símbolos). Una vez pasa los filtros, se le añade a la contraseña un salt. Esta combinación procede a derivarse con **PBKDF2HMAC** con **SHA256** (hash) .

Para poder llevar a cabo la autenticación en futuros inicios de sesión, el salt se almacena en la base de datos junto al hash de la contraseña.

CIFRADO SIMÉTRICO/ ASIMÉTRICO

Uso del cifrado asimétrico

El cifrado asimétrico se utiliza en el proyecto para asegurar la confidencialidad de los mensajes enviados entre usuarios. En el archivo message.py, se utiliza el cifrado **RSA** para cifrar los mensajes tanto para el receptor como para el remitente. Esto se debe a que tanto el receptor va a poder ver (descifrar) los mensajes recibidos por otros usuarios, como el remitente va a poder ver (descifrar) sus propios mensajes enviados.

Algoritmos utilizados:

El cifrado **RSA** es un algoritmo de cifrado asimétrico que utiliza un par de claves: una clave pública y una clave privada. La clave pública se utiliza para cifrar los datos, mientras que la clave privada se utiliza para descifrarlos. Este método asegura que sólo el destinatario previsto, que posee la clave privada correspondiente, pueda descifrar y leer los datos cifrados.

Gestión de claves durante el cifrado

En cuanto a la gestión de claves, hay que diferenciar entre:

Clave Pública: Se genera mediante el algoritmo `generate_rsa_keys`, esta depende directamente de la clave privada. Una vez generada la clave se almacena en formato pem en la base de datos, siendo así accesible sin necesidad de ninguna otra clave.

Clave Privada: Se genera mediante el algoritmo `generate_rsa_keys`, que directamente usa un método de la biblioteca **RSA**, con `public_exponent=65537, key_size=2048` (la recomendada para el algoritmo en cuestión)**.**

CIFRADO SIMÉTRICO

Una vez generada la clave privada, antes de guardarla en la base de datos, el método `cipher_private_key` utiliza el algoritmo proporcionado por **serialization.BestAvailableEncryption** para cifrar la clave privada con la contraseña del usuario.Así únicamente el propio usuario pueda saber su clave privada .Ya que a la hora de descifrar el mensaje, antes se tendrá que descifrar la clave privada.

Después, las claves se almacenan en la base de datos del usuario.

Gestión de los mensajes

Una vez gestionadas las claves, se cifran los mensajes de dos maneras diferentes. Primero se cifran los mensajes con la clave pública almacenada del receptor, para que él con su clave privada descifrada (ya que al iniciar sesión se descifra su propia clave privada) pueda leer el mensaje correctamente. Y por otro lado también se cifra el mensaje con la clave pública del emisor, para que él con su clave privada pueda ver sus mensajes enviados. Una vez cifrados los dos, se almacenan en la base de datos de mensajes.

GENERACIÓN/ VERIFICACIÓN DE ETIQUETAS DE AUTENTICACIÓN DE MENSAJES

Propósito de las Funciones MAC

Integridad: Aseguran que el mensaje no ha sido alterado durante la transmisión. Si se modifica el contenido, la verificación del MAC fallará.

Autenticidad: Garantizan que el mensaje proviene de una fuente confiable. Solo el emisor y el receptor, que conocen la clave secreta, pueden generar y verificar el MAC.

Algoritmos Utilizados

- **HMAC (Hashed Message Authentication Code):**

  La clase `Crypto` utiliza **HMAC** con el algoritmo **SHA-256**. Este algoritmo combina un hash criptográfico con una clave secreta, generando un código que se guarda junto al mensaje.

- **Razones para su uso:**

  Seguridad: **SHA-256** es un algoritmo robusto que proporciona una alta resistencia a colisiones.

  Eficiencia: **HMAC** es relativamente rápido y adecuado para ser utilizado en una amplia variedad de aplicaciones.

Gestión de Claves

- **Generación de Claves:** Las claves **HMAC** se generan a partir de la contraseña del usuario utilizando el algoritmo **PBKDF2 (Password-Based Key Derivation Function 2)**, que aplica una función hash con un número alto de iteraciones para fortalecer la seguridad de la clave generada.
- **Almacenamiento:** Las claves generadas se almacenan encriptadas en una base de datos.

Algoritmos de Cifrado Autenticado

La clase `Crypto` implementa un cifrado autenticado utilizando **RSA** para propiedades de cifrado asimétrico junto con **HMAC** para la autenticación.

Ventajas del Cifrado Autenticado

1. Confidencialidad y Autenticidad
2. Protección contra Modificaciones
3. Integración Sencilla