Entrega 2

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. Habrá posibilidad de poder enviar mensajes entre usuarios de forma segura. 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.

FIRMA DIGITAL

Utilización de la firma digital

En nuestro proyecto (PadelApp), utilizamos la firma digital para verificar que un mensaje ha sido enviado por el remitente legítimo y que no ha sido alterado durante la transmisión. Para ello, usamos 2 métodos a los que recurrimos desde los métodos que gestionan tanto la carga como la descarga de mensajes. El método `sign_message()`, es utilizado cuando se manda un mensaje en `send_message()`, para crear la firma y añadirla en la fila de la tabla correspondiente a ese mensaje. Sin embargo en los métodos de descarga de mensaje, utilizamos otros dos métodos que son `load_messages_receiver()`, en el que internamente se utiliza `verify_signature()`, para como su propio nombre indica verifica mediante el mensaje y la firma si es correcta esta.

Algoritmo utilizado

Para implementar la firma digital , usamos el algoritmo RSA con padding PSS (Probabilistic Signature Scheme), un esquema que mejora la seguridad frente a ataques. En cuanto a este algoritmo, tenemos ciertos componentes claves como:

`uncipher_private_key.sign`: Este método firma el mensaje utilizando la clave privada RSA, generando la firma digital correspondiente.

`message.encode()`: Convierte el mensaje en una secuencia de bytes, ya que la firma trabaja con datos binarios.

`padding.PSS`: Define el esquema de padding probabilístico PSS, que introduce aleatoriedad en el proceso de firma.

`mgf=padding.MGF1(hashes.SHA256())`: Utiliza MGF1 (una función de generación de máscara) con SHA-256 como función hash, garantizando que la máscara generada sea segura.

`salt_length`=padding.PSS.MAX_LENGTH: Configura la longitud de la sal al máximo permitido, añadiendo robustez contra ataques de precomputación.

`hashes.SHA256()`: Especifica SHA-256 como el algoritmo de hash para la generación de la firma.

Hemos recurrido a este algoritmo ya que esta combinación de RSA con PSS y SHA-256 garantiza que la firma digital sea resistente a ataques de falsificación y permite la validación segura del mensaje por parte del receptor.

Gestión de claves y firmas

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). Tras ser cifrada mediante el password del usuario, se almacena también en la fila de la tabla correspondiente al mismo

Como ya se ha dicho antes, la firma se genera a la hora de cargar el mensaje; toma el mensaje a firmar, el nombre del remitente y la contraseña del usuario. La clave privada del usuario se descifra utilizando la contraseña proporcionada y luego se utiliza para firmar el mensaje con el algoritmo RSASSA-PSS y el hash SHA-256. La firma generada se almacena en la base de datos junto con el mensaje.

La firma, para que sirva de algo se ha de verificar. Para ello se utiliza la clave pública del remitente que se serializa y se utiliza para verificar la firma del mensaje. Si la verificación es exitosa, se confirma que el mensaje no ha sido alterado y que proviene del remitente legítimo.

Comprobación de la firma digital

Nos planteamos al principio, como podríamos corroborar que la firma digital estaba bien implantada en el proyecto, para ello decidimos alterar el mensaje a la hora de descargarlo, y meterlo en el método de `verify_signuture()`, alterado.

```
# Verificar la firma digital, se altera el mensaje para
# comprobarlo. Funcionamiento:
# 1. Se altera el mensaje
# 2. Cambiar parametros de signature_bool
'''altered_message = message_decode + "altered"'''
signature_bool = self.crypto.verify_signature(message_decode,message[8],
                                              sender_publik_key)
```

Siendo signature_bool solo False, cuando no se verifica la firma digitel. Comprobamos que al alterar el mensaje, la verificación de firma digital es errónea como bien esperábamos que pasara.

CERTIFICADO DE CLAVE PÚBLICA

Certificado de clave pública

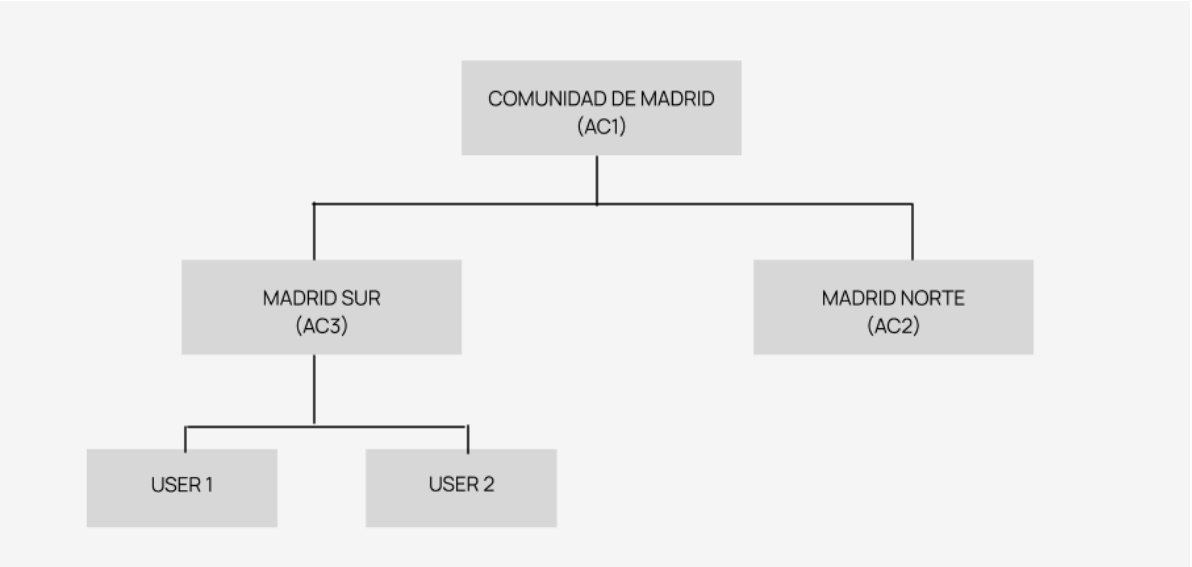
En cuanto a la generación de los certificados de clave pública, debemos de distinguir entre las diferentes autoridades:

- Generación de un Certificado Autofirmado (AC1):
  El proceso comienza con la creación de una clave privada para la autoridad certificadora raíz, conocida como AC1. Esta clave privada será el núcleo de la seguridad del sistema de certificados, será cifrada antes de añadirla a la base de datos por una contraseña que se pedirá por pantalla (La gestión de certificados se realiza desde la misma aplicación, por lo que se da por hecho que la persona que inicia por primera vez la aplicación es la autoridad raíz). Luego, se define un nombre para el certificado, que identifica de manera única a la autoridad. Con estos elementos, se construye el certificado, que posteriormente es firmado utilizando la clave privada de AC1. Finalmente, tanto la clave privada como el certificado se serializan, lo que permite su almacenamiento seguro o transferencia.
- Generación de Certificados Subordinados (AC2 y AC3):
  A continuación, se generan certificados para autoridades certificadoras subordinadas. Para cada una, primero se crea su propia clave privada. Posteriormente, se elabora una Solicitud de Firma de Certificado (CSR, por sus siglas en inglés), que incluye la información necesaria para la creación del certificado. Esta solicitud es firmada por AC1, estableciendo así la cadena de confianza. Al igual que con AC1, la clave privada y el certificado resultante se serializan para su manejo seguro.
- Generación de Certificados para Usuarios:
  Por último, se emiten certificados para usuarios finales. En este caso, el usuario genera su CSR, que es enviada a una de las autoridades certificadoras subordinadas (AC2 o AC3). La autoridad elegida firma el CSR (solo si la elección

es la correcta) con su clave privada, que para descifrarla, la autoridad deberá introducir la clave maestra. Hasta que esto no suceda, no se podrá entregar el certificado. Una vez generado el certificado se le adjudica al usuario. Este certificado es luego serializado, completando el proceso de certificación para el usuario.

Jerarquía de autoridades y por qué:

La jerarquía de autoridades en cuanto a los certificados, se resume en una foto.



Podemos ver cómo la Comunidad de Madrid, es la autoridad raíz en nuestro proyecto, ya que vimos bien la subdivisión en la que las otras dos autoridades de certificación subordinadas fueran una división parcial de Madrid, para no complicar demasiado el trabajo, pero mantener la coherencia.

La jerarquía es importante porque establece una cadena de confianza. Los certificados de usuario son confiables porque han sido emitidos por una autoridad subordinada, que a su vez ha sido certificada por la autoridad raíz. Esta estructura permite una gestión segura y escalable de certificados digitales.

Implementación

En esta aplicación, se implementan mediante la generación de certificados autofirmados y certificados subordinados, así como la verificación de los mismos.

Cuando se carga la aplicación por primera vez, se generan tanto los certificados del raíz, como de los subordinados. Esto se realiza mediante los métodos `generate_self_signed_cert()`,que genera un certificado de autofirmado y `generate_subordinate_cert()`, que genera un certificado subordinado utilizando el certificado raíz. Una vez generados, los almacenamos en una tabla

específicamente para los certificados, tanto estos como sus claves privadas (cifradas para que no sean accesibles).

Para los usuarios, se utilizan otros dos métodos diferentes, que son `apply_certificate_norte()`, que aplica un certificado subordinado para un usuario en la zona "Madrid Norte". `apply_certificate_sur()`, que aplica un certificado subordinado para un usuario en la zona "Madrid Sur". No es posible obtener un certificado de Madrid Norte, si en el registro has puesto que eres de Madrid Sur y viceversa.

Y por último tenemos el método `verify_certificate()`, que verifica si un certificado es válido comparando la clave pública del certificado almacenado con la clave pública proporcionada. La no certificación, supone el impedimento de realización de actividades dentro de la App.

Utilización de los certificados

Dentro de la App, como he indicado anteriormente, le hemos dado sentido a los certificados, a la hora de trabajar con los mensajes entre jugadores. Primero, un jugador no podrá mandar un mensaje si no está certificado. Y en caso de que tú (que quieres mandar un mensaje) sí estás certificado pero el receptor del mensaje no lo está, el programa te advertirá de que el usuario al que vas a mandar un mensaje no está certificado. En ese punto tú decides si enviarlo de todas formas o cancelar el mensaje.

Con esta mejora en nuestra App conseguimos que la identidad de los usuarios y los mensajes sean seguros y tener firmeza de que las acciones provienen de fuentes confiables.

COMPLEJIDAD Y DISEÑO DE CÓDIGO

Estructura y modularidad

En el diseño de nuestra aplicación, hemos optado por una arquitectura modular que separa claramente las responsabilidades. Por un lado, contamos con una clase `Crypto` encargada de las operaciones criptográficas fundamentales, como el cifrado, descifrado, generación de claves y firmas digitales. Por otro lado, la clase `Certificate` se encarga de la gestión de certificados digitales, lo que incluye la emisión, firma, y verificación de certificados, así como su almacenamiento en base de datos.

Complejidad funcional

El nivel de complejidad de nuestro proyecto se incrementa debido a la incorporación de diversas funcionalidades avanzadas:

- **Operaciones criptográficas:** La generación y gestión de claves RSA de 2048 bits, junto con el uso de algoritmos como SHA-256 y RSASSA-PSS, garantizan un alto nivel de seguridad, aunque requieren una comprensión profunda de los principios criptográficos.

- **Gestión de certificados:** Implementamos un sistema jerárquico de certificados, con una autoridad certificadora raíz (AC1) y autoridades subordinadas (AC2 y AC3). Esta estructura añade una capa de complejidad tanto en términos de diseño como de implementación, ya que cada nivel debe interactuar de manera segura y coherente.

- **Integración con la base de datos:** El uso de una base de datos para almacenar claves privadas cifradas y certificados introduce desafíos adicionales en términos de rendimiento y consistencia, especialmente en operaciones como la verificación de usuarios y certificados.

Seguridad

La seguridad es un pilar fundamental de nuestro diseño. Hemos implementado múltiples medidas para garantizarla:

- **Protección de claves privadas:** Estas se almacenan cifradas utilizando las contraseñas de los usuarios, lo que asegura que incluso si la base de datos es comprometida, la información no será útil sin las credenciales correspondientes.

- **Firmas digitales y verificación:** Hemos implementado funciones para la firma y verificación de mensajes, utilizando estándares modernos que aseguran la autenticidad e integridad de los datos.

- **Certificados digitales:** Cada usuario puede contar con un certificado que autentica su identidad, emitido por una autoridad subordinada confiable.

Complejidad técnica

El uso de bibliotecas avanzadas como `cryptography` nos ha permitido implementar funcionalidades robustas, aunque también ha requerido un alto grado de especialización por parte del equipo. Por ejemplo, el manejo de objetos como `x509.Certificate` o `RSAPrivateKey` implica una comprensión detallada de sus métodos y características.

Adicionalmente, el diseño jerárquico de certificados y la gestión de claves privadas cifradas requieren una coordinación precisa entre las clases `Crypto` y `Certificate`. Esta dependencia cruzada, aunque necesaria, introduce un nivel adicional de complejidad.

Mantenibilidad y extensibilidad

Hemos tomado decisiones de diseño para garantizar que el sistema sea fácil de mantener y extender:

- **Código legible y comentado:** Hemos incluido comentarios claros para explicar los pasos más importantes en cada funcionalidad.
- **Uso de estándares:** Nos hemos adherido a estándares bien establecidos en criptografía y gestión de certificados, lo que facilita la futura integración con otros sistemas.
- **Separación de responsabilidades:** La división en clases independientes hace que sea sencillo modificar o añadir características en áreas específicas sin riesgo de romper otras partes del sistema.

Limitaciones y retos

A pesar de estas ventajas, hemos identificado algunas limitaciones y retos en nuestro diseño:

- **Dependencias cruzadas:** La interacción entre `Crypto` y `Certificate` puede complicar el flujo de datos y requerir más pruebas para asegurar la consistencia.
- **Complejidad en pruebas:** Dado el nivel de abstracción y las múltiples capas de seguridad, las pruebas del sistema son complejas y requieren herramientas específicas para validar el correcto funcionamiento de los certificados y las firmas digitales.

En resumen, nuestro proyecto combina funcionalidades avanzadas de criptografía y gestión de certificados con un diseño modular que prioriza la seguridad y la escalabilidad. Aunque el nivel de complejidad es elevado, consideramos que las decisiones tomadas sientan una base sólida para el desarrollo futuro.

MEJORAS IMPLEMENTADAS

Jerarquía extendida de la PKI

El equipo ha ampliado la jerarquía básica de la infraestructura de clave pública (PKI), creando no solo una **Autoridad certificadora raíz (AC1)**, sino también dos autoridades subordinadas adicionales: **Madrid Norte (AC2)** y **Madrid Sur (AC3)**. Esta mejora, aunque no requerida explícitamente, añade robustez al sistema y simula un entorno más realista, donde cada autoridad subordinada puede gestionar certificados en su ámbito regional.

Gestión avanzada de certificados

Se han desarrollado funciones específicas para emitir certificados personalizados a los usuarios desde las autoridades subordinadas (`apply_certificate_norte` y `apply_certificate_sur`). Al iniciar la aplicación se solicitará a la entidad una contraseña maestra para cifrar las claves privadas de la autoridad raíz y subordinadas. Cuando un usuario solicita un certificado, salta un pop-up en el cual se pedirá la contraseña maestra para conceder este certificado si se ha insertado la correcta, es decir la que se proporcionó al abrir la aplicación por primera vez . Estas funcionalidades aseguran que los usuarios puedan integrarse fácilmente en la PKI y permiten su autenticación de manera descentralizada, algo especialmente útil en escenarios reales.

Base de datos para almacenamiento seguro

Hemos implementado una base de datos para almacenar de forma estructurada claves privadas cifradas, certificados y datos relacionados con los usuarios. Este enfoque garantiza un almacenamiento seguro y facilita la gestión de datos sensibles, al mismo tiempo que permite consultas eficientes.

Interfaz gráfica de usuario (GUI)

Además de las funcionalidades criptográficas, se ha desarrollado una interfaz gráfica para la aplicación, utilizando Tkinter. Esto convierte el proyecto en una herramienta funcional y accesible para usuarios finales, trascendiendo la línea de comandos y acercándose a un caso de uso real.

Recuperar contraseña

Hemos implementado una clase para gestionar el cambio de contraseña del usuario, se solicita un número de teléfono para enviar un SMS con un pin y poder verificar la identidad de este. Sin embargo no funciona ya que esta contraseña cifra la clave privada, por lo que sí es cambiada no se podrá acceder a la información del usuario.

Validación de datos y manejo de excepciones

Aunque no era estrictamente necesario, se ha implementado la validación de las entradas de los usuarios, reduciendo el riesgo de ataques por inyección o errores. Además, el código incorpora captura de excepciones y genera logs detallados sobre las operaciones criptográficas realizadas, mejorando tanto la seguridad como la trazabilidad del sistema.

Ampliación del diseño funcional

El equipo ha diseñado una aplicación completamente funcional que incluye autenticación de usuarios, gestión de claves y certificados, y firma/verificación de mensajes. El enfoque del diseño está orientado a casos de uso reales, donde la seguridad y la interoperabilidad son prioridades.