SecureSend

Privacidad y Seguridad

Grupo: 82

ID de grupo de prácticas: 9

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.gitortada

Indice

¿Cuál es el propósito de su aplicación? ¿Cuál es su estructura interna? 3
¿Para qué utiliza la firma digital? 3
    ¿Qué algoritmos ha utilizado y por qué? 3
¿Cómo se gestionan y almacenan las claves y las firmas? 4
¿Cómo se generan los certificados de clave pública? 5
¿Qué jerarquía de autoridades de certificación se ha desplegado? 5
    ¿Por qué se ha escogido esta configuración? 6
    ¿Cómo se ha implementado? 6
¿Cuándo se usan los certificados y para qué? 6
Pruebas realizadas para garantizar la calidad del código 7

¿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.

¿Para qué utiliza la firma digital?

En nuestra aplicación, la firma digital se utiliza para garantizar tres propiedades fundamentales de seguridad en los documentos confidenciales que los usuarios suben al sistema: autenticidad, integridad y no repudio.

Específicamente, la implementación realizada en el módulo digital_signature.py y orquestada en main.py cumple los siguientes propósitos:

- Garantía de Autoría (Autenticidad): Permite verificar matemáticamente que un documento fue subido realmente por un usuario específico (el firmante). Al verificar la firma con la clave pública del usuario (accesible mediante KeyManager), el sistema confirma que solo el poseedor de la clave privada correspondiente pudo haber generado esa firma.
- Detección de Manipulaciones (Integridad): La firma digital actúa como un sello de inviolabilidad. Cualquier modificación, incluso de un solo bit, en el archivo original después de haber sido firmado (ya sea por corrupción de datos o por un ataque malintencionado) provocará que la función verify_signature falle. Esto asegura al receptor que el documento es idéntico al original.
- No Repudio en origen: En la función upload_document, hemos implementado un mecanismo explícito donde se pide de nuevo al usuario que introduzca su contraseña justo antes de firmar. Esto desbloquea la clave privada únicamente para esa operación, vinculando el hecho de haber firmado con la intención del usuario, impidiendo así que este usuario pueda negar haber realizado el envío del documento en el futuro.

¿Qué algoritmos ha utilizado y por qué?

Para la implementación de la firma digital en la clase SignatureManager, hemos seleccionado una combinación de algoritmos modernos y robustos:

- Algoritmo de Clave Pública: RSA
    - Configuración: Se utilizan claves de 2048 bits (definido en config.py y utilizado en KeyManager).
    - Por qué: RSA es el estándar industrial para criptografía asimétrica. Una longitud de 2048 bits ofrece un equilibrio adecuado entre seguridad computacional (resistente a la factorización de enteros actual) y rendimiento en la generación y verificación de firmas.
- Esquema de Relleno (Padding): PSS (Probabilistic Signature Scheme)

    - Por qué: Se ha elegido PSS en lugar del antiguo estándar PKCS#1 v1.5. PSS es un esquema probabilístico, lo que significa que incorpora un valor aleatorio (*salt*) en el proceso de firma. Esto hace que firmar el mismo documento dos veces genere firmas binarias diferentes, aunque ambas sean válidas. Esto ofrece una seguridad matemáticamente demostrable (provable security) y es más resistente a ataques teóricos que afectan a esquemas deterministas más antiguos.
- Algoritmo de Hashing: SHA-256 (Secure Hash Algorithm 2)
    - Implementación: hashes.SHA256()
    - Por qué: Antes de cifrar con la clave privada, es necesario obtener un resumen (digest) del documento. Se utiliza SHA-256 porque pertenece a la familia SHA-2, es ampliamente soportado, libre de colisiones conocidas en la práctica y proporciona un nivel de seguridad (128 bits de resistencia a colisiones) acorde con la seguridad de la clave RSA de 2048 bits.

¿Cómo se gestionan y almacenan las claves y las firmas?

La gestión y almacenamiento se realiza siguiendo el principio de separación de privilegios y persistencia en disco, integrando los módulos key_manager.py y main.py:

Gestión y Almacenamiento de Claves

- Generación:
    - 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.
- Almacenamiento de Clave Privada:
    - Se almacena en el directorio del usuario: data/users/{username}/private_key.pem.
    - Seguridad Crítica: La clave privada nunca se guarda en texto plano. Se utiliza el estándar PKCS8 y se cifra utilizando la contraseña del usuario mediante algoritmos simétricos robustos (AES-256) proporcionados por serialization.BestAvailableEncryption. Esto asegura que ni siquiera un administrador del sistema pueda usar la clave privada sin la contraseña del usuario.
- Almacenamiento de Clave Pública:
    - Se almacena en formato PEM (texto plano) en data/users/{username}/public_key.pem. Esta clave es accesible por el sistema para operaciones de verificación (verify_signature) y cifrado de claves simétricas.
- Ciclo de vida: Las claves se cargan en memoria (load_private_key) solo durante el instante necesario para firmar y se liberan inmediatamente después.

Gestión y Almacenamiento de Firmas

- Almacenamiento (Archivo .sig): Para no alterar el documento original, la firma se almacena como un archivo independiente en el mismo directorio que el documento cifrado (data/documents/{username}/) y con un formato binario.
    - Si el documento es contrato.pdf, la firma se guarda como contrato.pdf.sig.
- Vinculación: La relación entre el documento cifrado (.enc), la firma (.sig) y las claves de cifrado se mantiene mediante un archivo de metadatos (.meta). En main.py, al finalizar la subida, se escribe en el JSON de metadatos una referencia al archivo de firma, garantizando que el sistema sepa qué archivo de firma corresponde a qué documento.

¿Cómo se generan los certificados de clave pública?

Los certificados se crean según el estándar X.509 utilizando la librería cryptography de Python. Primero se genera un par de claves RSA (privada y pública), cifrando la privada si es necesario. Luego se construye el certificado con CertificateBuilder, incluyendo datos como Subject, Issuer, número de serie, fechas de validez y extensiones X.509 (como BasicConstraints, KeyUsage, SubjectKeyIdentifier y AuthorityKeyIdentifier).
El certificado se firma con RSA y SHA-256: la AC Raíz se autofirma, mientras que las subordinadas y usuarios se firman con la clave de su autoridad superior. Los certificados se guardan en formato PEM. Para los usuarios se añaden extensiones como ExtendedKeyUsage (autenticación y correo seguro) y SubjectAlternativeName (email), garantizando así certificados únicos, verificables y seguros.

¿Qué jerarquía de autoridades de certificación se ha desplegado?

Se ha implementado una PKI de tres niveles:

1. **AC Raíz (SecureSend Root CA):** punto de confianza principal, autofirmado, con validez de 10 años y *path_length=1*.

2. **AC Subordinada (SecureSend Subordinate CA):** firmada por la raíz, válida 5 años, con *path_length=0*, responsable de emitir certificados de usuario.

3. **Certificados de usuario:** firmados por la AC Subordinada, válidos 1 año y con *ca=False*.
   Los certificados de usuario se validan comprobando toda la cadena de confianza hasta la AC Raíz. Si cualquier verificación falla, el certificado se considera inválido.

¿Por qué se ha escogido esta configuración?

La arquitectura de dos niveles (Raíz y Subordinada) equilibra seguridad, escalabilidad y simplicidad. Separar ambas funciones protege la clave privada de la Raíz, que puede

mantenerse offline, mientras la Subordinada gestiona las operaciones diarias.
Si se comprometiera la Subordinada, bastaría con revocarla sin reconstruir toda la PKI.
Además, permite crear múltiples AC subordinadas para diferentes propósitos (usuarios, dispositivos, regiones, etc.) y usar periodos de validez adaptados a cada nivel (10 años la Raíz, 5 la Subordinada, 1 los usuarios).
Una sola AC sería insegura por la exposición constante de su clave raíz, y más niveles añadirían complejidad innecesaria. La configuración actual es el punto óptimo entre seguridad y operatividad.

¿Cómo se ha implementado?

Todo se gestiona mediante el módulo pki_manager.py y la clase PKIManager.

- **create_root_ca()** crea la AC Raíz (clave RSA, certificado autofirmado, extensiones y firma con SHA-256).

- **create_subordinate_ca()** genera la AC Subordinada, firmada por la Raíz y con las extensiones adecuadas.

- **issue_user_certificate()** emite certificados de usuario (con *BasicConstraints=ca:False*, *KeyUsage* y *ExtendedKeyUsage* adecuados).

- **verify_certificate_chain()** valida toda la cadena de confianza y las fechas de validez

La inicialización automática de la PKI ocurre al arrancar la aplicación desde main.py, y los certificados de usuario se crean tras su registro y generación de claves RSA.

¿Cuándo se usan los certificados y para qué?

Se utilizan para autenticar y verificar usuarios en operaciones sensibles:

- **Subida de documentos:** solo usuarios con certificado válido pueden hacerlo.

- **Verificación de identidad:** permite comprobar la legitimidad de otro usuario mediante la cadena de confianza.

- **Cifrado de documentos:** (planificado) se usará la clave pública del destinatario verificada mediante su certificado.

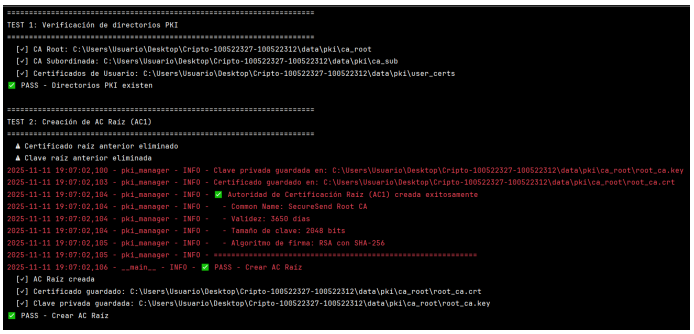
- **Firmas digitales:** garantizarán autenticidad, integridad y no repudio de documentos.

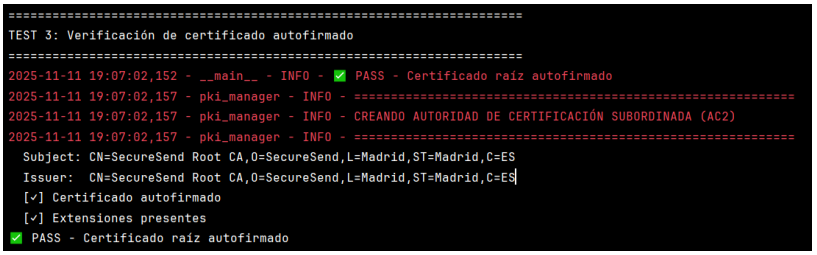
- **Visualización y control de acceso:** los usuarios pueden consultar su certificado, y en el futuro se usarán atributos del mismo para definir permisos.

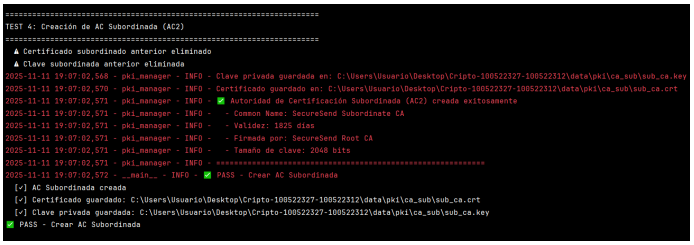
En conjunto, los certificados son la base de la seguridad de SecureSend, proporcionando autenticación, confidencialidad e integridad, siguiendo los mismos principios que las infraestructuras PKI reales en Internet y entornos corporativos.

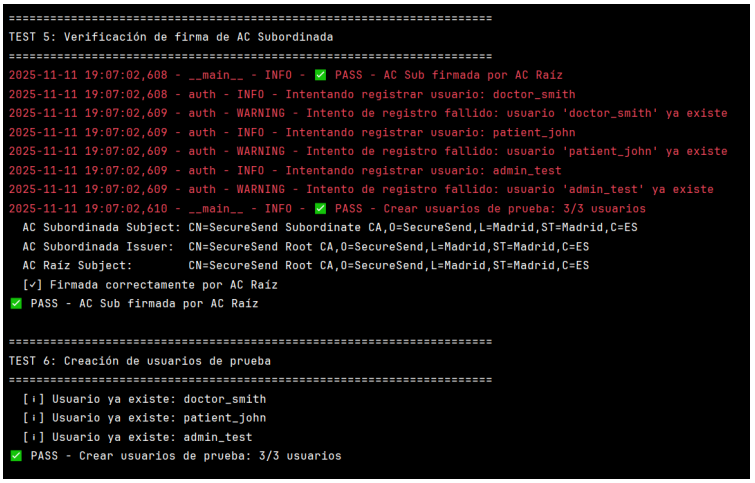
Pruebas realizadas para garantizar la calidad del código

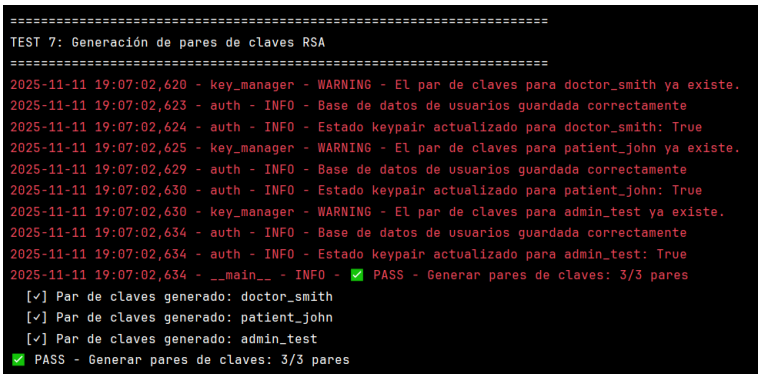
Para la Autenticación de las claves públicas mediante certificados se han realizado las siguientes pruebas:

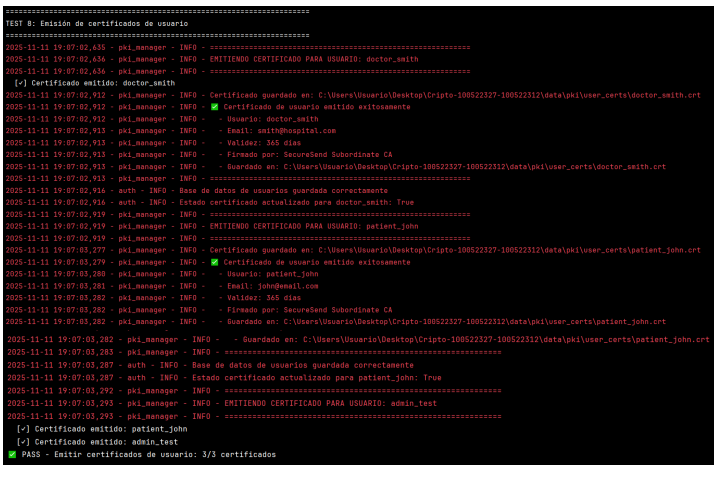


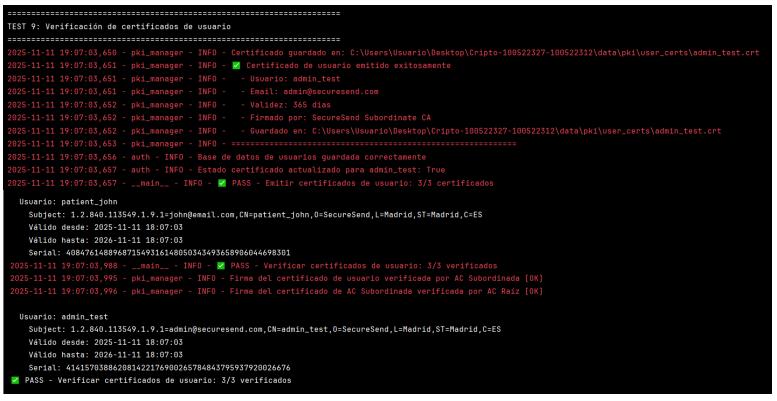


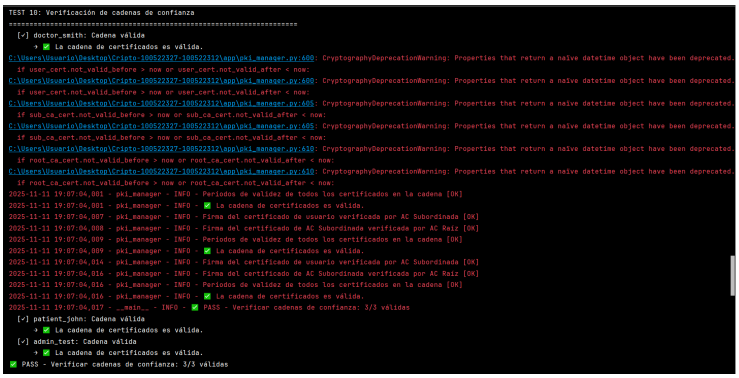


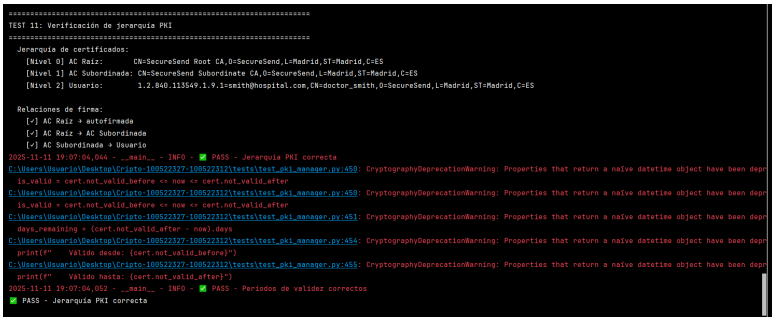


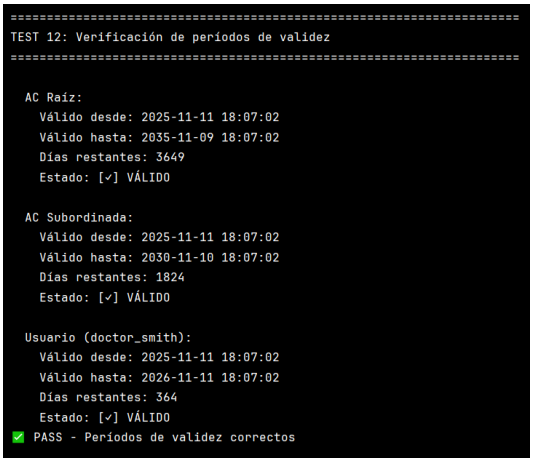












Los tests para la comprobación del correcto funcionamiento de la firma digital son los siguientes:

```
PS C:\Uni\Cripto\Cripto-100522327-100522312> python -m unittest tests/test_digital_signature.py
.Error al descifrar la clave privada de test_user_paco. Contraseña incorrecta.
No se pudo cargar la clave privada de test_user_paco. Abortando firma.
.Verificación de firma DIGITAL: FALLO. La firma no coincide o el documento fue alterado.
.Verificación de firma DIGITAL: FALLO. La firma no coincide o el documento fue alterado.
.Verificación de firma DIGITAL: FALLO. La firma no coincide o el documento fue alterado.
.
----------------------------------------------------------------------
Ran 5 tests in 0.495s

OK
PS C:\Uni\Cripto\Cripto-100522327-100522312>
```

En estos tests se comprueba si:

1. La verificación de la firma es correcta;
2. La integridad del documento, si un solo byte varía falla;
3. La autenticidad, si se intenta verificar la firma con la clave pública de otro usuario falla;
4. La contraseña correcta, si se intenta verificar partiendo de una contraseña incorrecta falla; y
5. Si se corrompe la firma, si esto ocurre también falla