uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

Criptografía y seguridad informática 25/26

Grupo 81

Práctica 1

«OmniMessenger: Envío de mensajes en masa - Parte 2»

Denis Loren Moldovan — 100522240

Jorge Adrian Saghin Dudulea — 100522257

Profesor

Lorena Gonzalez Manzano

Tabla de Contenidos

1. Propósito y estructura 2
   1.1. Como ejecutar la aplicación 2
2. Firma digital 3
   2.1. Algoritmos utilizados 3
   2.2. Estructura del PDF 4
   2.3. Gestión y Almacenamiento de las claves y las firmas 4
   2.4. Representación visual de la firma 5
3. Certificados de Clave pública 6
   3.1. Generación de los Certificados de Clave Pública 6
   3.2. Jerarquía de Autoridades de Certificación 6
   3.3. Justificación de la Configuración 7
   3.4. Implementación 8
4. Pruebas realizadas 10

1. Propósito y estructura

La aplicación consiste en el envío masivo de mensajes, usando servicios como Telegram (Bots), Whatsapp (Business) y Gmail (Conexión al correo correspondiente), los cuales se almacenan para luego distribuirlos de forma ordenada.

Esta consiste en un backend escrito en Python, y un frontend en HTML, CSS, y JavaScript, los cuales se comunican mediante peticiones HTTP sobre TLS. Como no se va a desplegar la aplicación en un entorno de servidor real, se ha omitido el uso de «dotfiles» para la gestión de direcciones y de tokens. Por otro lado, el uso de la dirección «localhost» está codificado fuertemente dentro del código, entonces, si se quisiese usar un dominio personalizado, habría que implementarlo dentro del código.

Para el acceso a todos los servicios, se ha usado Caddy como proveedor de archivos, y proxy reversa. Además de asegurar un cifrado en la conexión, nos habilita el uso de subdominios y subdirectorios, donde se han usado para separar, por ejemplo, el visor de la base de datos. Por otro lado, se ha usado PostgreSQL para la base de datos y Adminer para la visualización de la base de datos. Finalmente, para asegurar el funcionamiento en la mayoría de máquinas, se ha contenerizado toda la aplicación usando Docker.

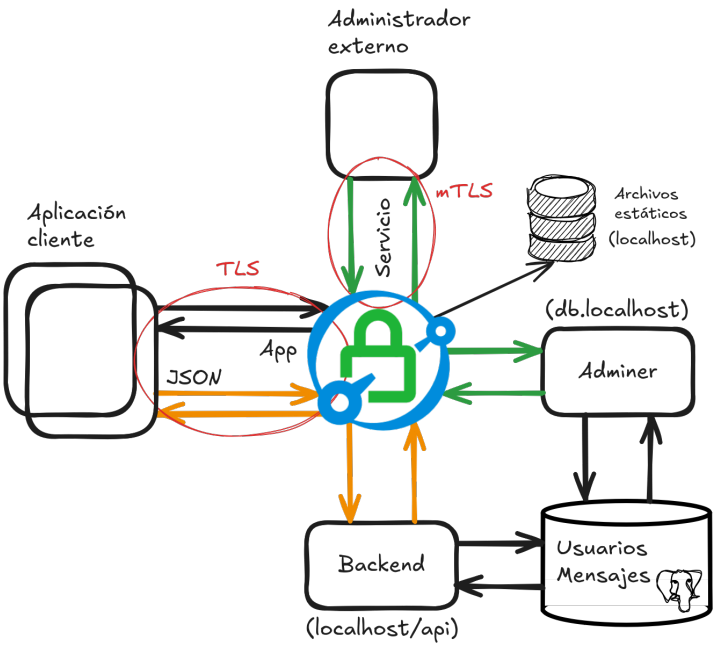

Figura 1: Diseño de la aplicación

1.1. Como ejecutar la aplicación

Para ejecutar la aplicación, primero hay que descargar y configurar Docker. Después, hay que construir la imagen personalizada usando `./setup.sh`, y finalmente ejecutar la aplicación usando `./run.sh`. Para detenerlo, hay que ejecutar `./down.sh`.[1]

La página principal se encontrará en `https://localhost`, y el gestor de la base de datos en `https://db.localhost`.[2] El certificado requerido para poder acceder al gestor se encuentra en `Docker/conf/SSL/Client/client.p12`, el cual hay que instalar en el navegador.[3]

[1] La aplicación ha sido desarrollada y probada en un entorno de Linux.
[2] Usuario predeterminado: «postgres» / Contraseña: «Cripto2526».
[3] La contraseña es «pkcs12»

2. Firma digital

Nuestro sistema implementa un mecanismo de firma electrónica basada en el estándar PAdES (PDF Advanced Electronic Signatures). Esto garantiza los siguientes pilares de la seguridad de la información:

- Integridad. Se asegura que el documento no ha sido modificado desde el momento de la firma. Cualquier cambio posterior invalidaría la firma.
- Autenticidad y no repudio. Vincula la identidad del firmante (a través de su certificado digital personal) con el documento e impide que este pueda negar la autoría de la firma a posteriori.

El objetivo principal es implementar una firma en el lado del cliente. Esto permite firmar los documentos en el navegador del usuario sin tener que enviar su clave privada al servidor, protegiendo la soberanía de los datos sensibles del usuario.

2.1. Algoritmos utilizados

Para la implementación criptográfica se han combinado varios estándares de PKCS para cubrir todo el proceso de la firma:

- PKCS12 para almacenamiento de claves: Utilizamos este estándar para la lectua segura del certificado digital y la clave privada del usuario. El sistema utiliza la librería Forge de JavaScript para extraer la clave privada del contenedor en la memoria durante el proceso de firma. Esto aporta ventajas de seguridad muy importantes:
  - Soberanía de la identidad. La clave nunca viaja a través de la red, lo que elimina el riesgo de intercepción (ataques Man-in-the-middle) o el almacenamiento en bases de datos de terceros.
  - El servidor solo recibe el documento final firmado, nunca va a tener acceso a las credenciales que generaron dicha firma.
  - Al residir las claves únicamente en variables locales de la función JavaScript, el ciclo de vida de los datos sensibles se limita solo al instante de la firma. El recolector de basura del navegador y el cierre de sesión garantizan que no queden residuos de la clave privada.
- PKCS7 para el encapsulamiento de la firma. La estructura de firma presenta tres características:
  1. Firma separada. Se ha configurado la firma en modo detached. La estructura PKCS7 contiene únicamente los datos criptográficos y los metadatos, manteniendo el contenido del PDF fuera. El vínculo entre ambos se asegura mediante el ByteRange y el hash del documento, lo que permite que el archivo final siga siendo un PDF válido y legible.
  2. Validación autocontenida. El contenedor generado no solo incluye la firma digital, sino también la copia pública del certificado X.509 del firmante. Esto garantiza que la firma sea autoportable, cualquier visor estándar puede verificar la identidad del firmante y la integridad del documento utilizando solamente la información incrustada, sin necesidad de acceso a la clave pública del usuario.
  3. Atributos autenticados. Para elevar el nivel de seguridad, no se firma únicamente el hash del documento. El sistema construye y firma un bloque de atributos autenticados que incluye, además del hash, el atributo signingTime (fecha y hora de la firma). Al estar protegido el atributo por la firma criptográfica, se garantiza la integridad temporal, por lo que si alguien intentara cambiar la fecha de su ordenador, la validación matemática fallaría porque el hash no coincidiría con los atributos firmados.

Para garantizar la interoperabilidad entre diferentes sistemas operativos y visores de PDF, las estructuras criptográficas generadas siguen las reglas de codificación DER(Distinguished Encoding Rules) del estándar ASN.1, asegurando que la representación binaria de la firma sea única e inequívoca.

- SHA-256 para la integridad del documento.
- Criptografía Asimétrica (RSA). Se utiliza el algoritmo RSA para el cifrado del hash. El sistema utiliza la clave privada, extraida del P12, para cifrar el resumen SHA-256 del documento. Cualquier persona con la clave pública (que viaja dentro del certificado) puede descifrarlo y verificar que coincide, garantizando la autenticidad y el no repudio.

2.2. Estructura del PDF

El proceso de inyección de firma se realiza mediante la manipulación directa de bytes:

1. Reserva de espacio. Antes de firmar, se modifica la estructura lógica del PDF para añadir un campo de firma vacío relleno de ceros. Este proceso pre-calcula el tamaño final del archivo.
2. Cálculo del ByteRange. Se implementa el mecanismo de rangos de bytes definido por Adobe. Este mecanismo divide el archivo en dos partes: todo lo que hay antes de la firma y todo lo que hay después.
3. Inyección hexadecimal. La firma se convierte en una cadena hexadecimal y se inyecta en el hueco reservado. Gracias al cálculo del ByteRange, el PDF sabe que debe verificar el hash de todo el documento excepto los bytes donde reside la firma.

2.3. Gestión y Almacenamiento de las claves y las firmas

- Gestión de claves. Las claves privadas nunca se almacenan en la aplicación ni se envían a través de la red.
  - El usuario carga su archivo .p12 localmente.
  - El código lee el archivo en memoria, extrae la clave, realiza la operación matemática de la firma y una vez completado el proceso, los datos sensibles se eliminan.
  - Esto elimina el riesgo de que el servidor comprometa y filtre las claves privadas.
- Almacenamiento de la firma. La firma no se guarda en una base de datos externa, sino que se inyecta dentro del propio PDF.
  - Se modifica la estructura interna del PDF para reservar un espacio.
  - Se calcula la firma y se escribe en formato hexadecimal en ese espacio reservado.
  - El resultado final es un archivo PDF autónomo que tiene tanto el contenido como la prueba criptográfica de su validez. El servidor solamente recibe y guarda el archivo firmado.

2.4. Representación visual de la firma

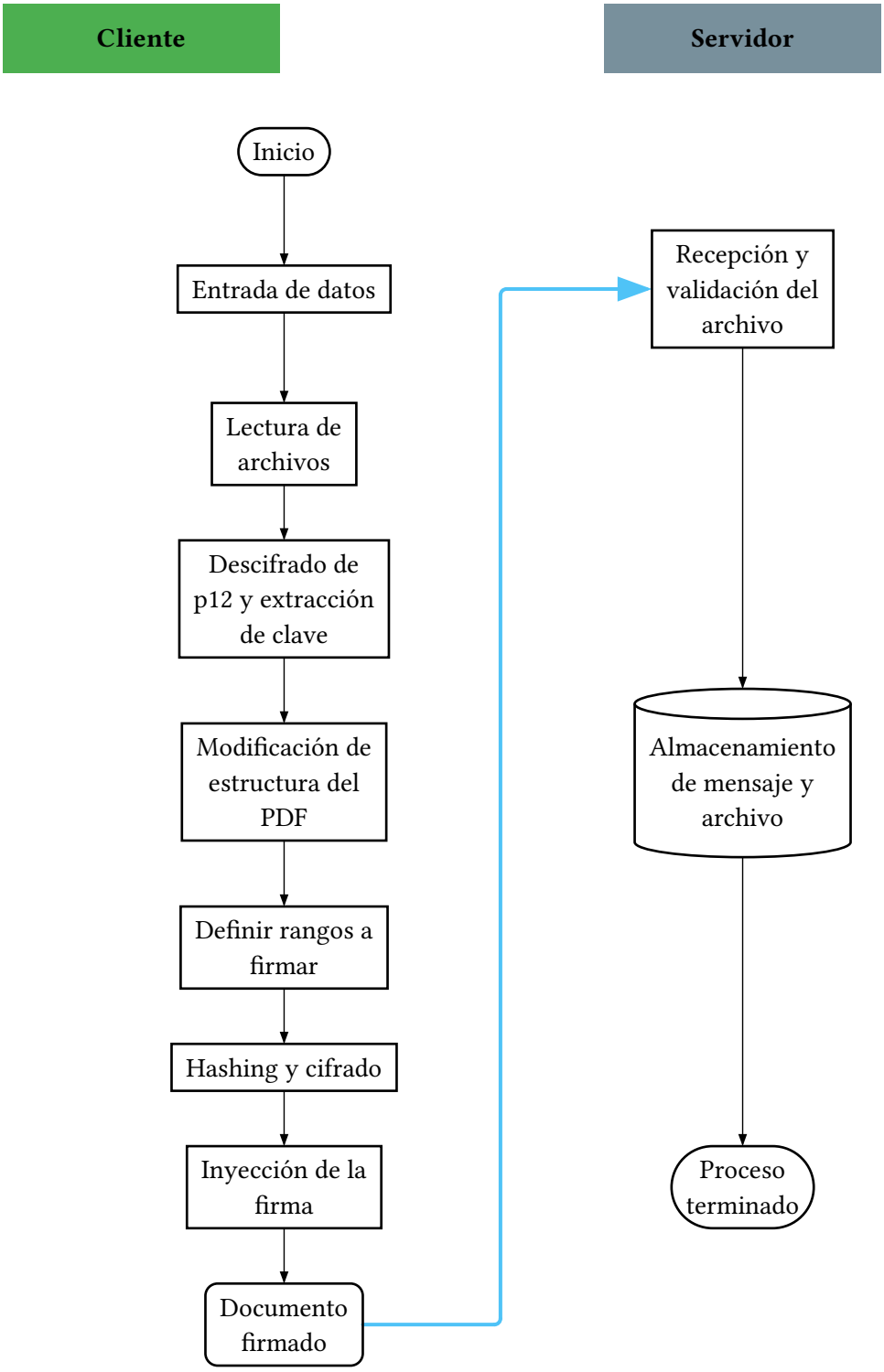

Figura 2: Diagrama de flujo del proceso de firma

3. Certificados de Clave pública

3.1. Generación de los Certificados de Clave Pública

El proceso de generación sigue el estándar X.509 y se divide en tres etapas:

- Generación del par de claves. Se utiliza el algoritmo RSA:
  - Para las Autoridades de Certificación, tanto la raiz como la intermedia, se han generado claves de 4096 bits.
  - Para las entidades de Cliente y Servidor se han utilizado claves de 2048 bits, obteniendo un equilibrio entre seguridad y rendimiento.

3.2. Jerarquía de Autoridades de Certificación

Hemos implementado una jerarquía de dos niveles, compuesta por las siguientes capas:

1. Root CA (Autoridad Raiz), «LocalRootCA»:
   - Es el ancla de confianza.
   - Está autofirmada.
   - Tiene una validez de 20 años.
2. Intermediate CA (Autoridad Intermedia), «LocalDevCA»:
   - Emitida y firmada por la autoridad raiz.
   - Actúa como la entidad encargada de emitir los certificados finales.
   - Tiene una validez de 10 años.
3. Entidades finales:
   - Server Cert. Certificado para el servidor, firmado por la CA Intermedia.
   - Client Cert. Certificado para el usuario, también firmado por la CA Intermedia.

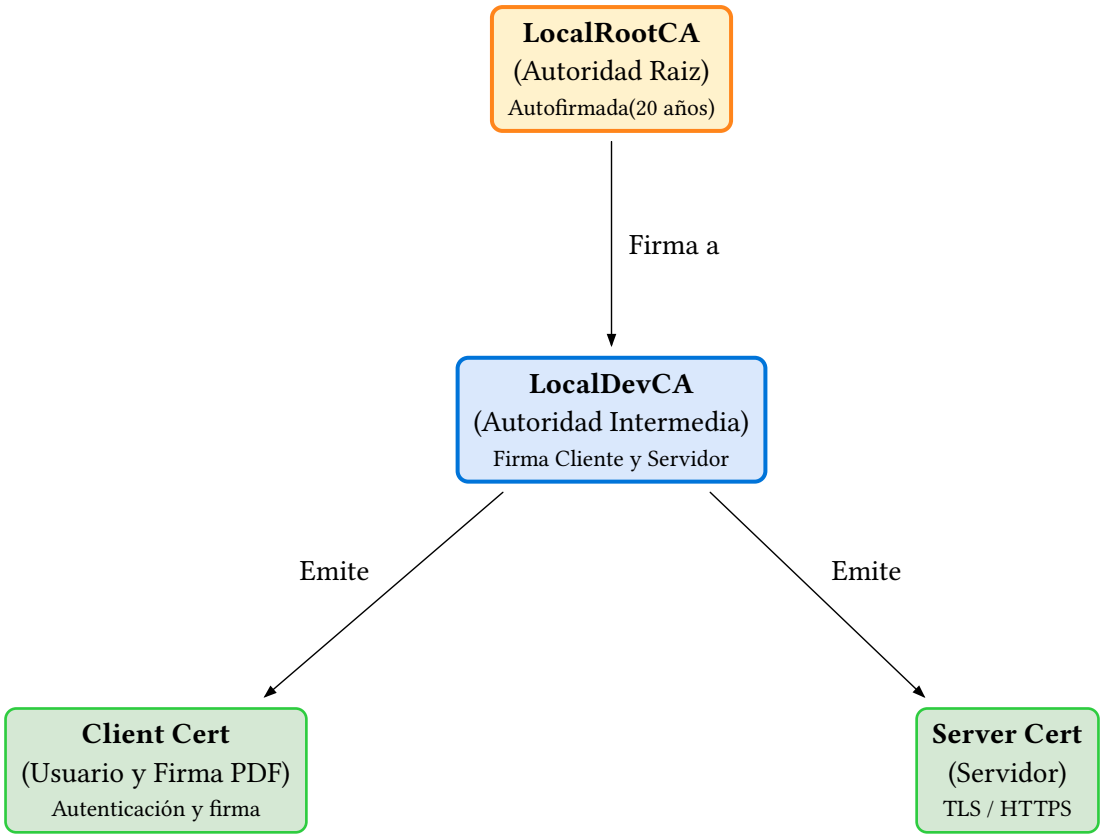

Figura 3: Jerarquía de Autoridades de Certificación(PKI)

A raiz de esta arquitectura jerárquica, el despliegue de los certificados en el servidor no se realiza mediante archivos individuales, sino mediante cadenas de certificados, para que los clientes puedan construir la ruta de confianza completa hasta la raiz.

- Cadena del servidor (server-chain.crt): Contiene la concatenación del certificado del servidor + certificado de la CA intermedia + certificado de la CA Raíz. Al enviar la ruta completa durante el handshake TLS, el servidor garantiza que el cliente reciba toda la información necesaria para validar la conexión.
- Cadena del cliente (client-chain.crt): Contiene la concatenación del certificado del cliente + el certificado de la CA intermedia + el certificado de la CA Raíz. Es necesario para la firma PAdES, ya que permite incrustar toda la ruta de confianza dentro del PDF, asegurando que cualquier sistema confñie en la raiz.
- Cadena de Autoridad (ca-chain.crt): Contiene la concatenación de la CA intermedia + la CA raiz. Se utiliza en el servidor para validar los certificados de cliente asegurando que solo se acepten usuarios autenticados por la jerarquía completa.

3.3. Justificación de la Configuración

Hemos optado por una jerarquía con una CA intermedia en lugar de firmar con la CA raiz por distintos motivos:

- Mayor seguridad para la CA raiz. Se mantiene offline, lo que reduce el riesgo de compromiso. Si la raiz permanece segura, toda la jerarquía mantiene su integridad.
- Mejor organización. Facilita separar roles administrativos. La CA raiz se encarga de la firma de las CAs intermedias y estas se encargan de la emisión de los certificados a usuarios y servidores.

- Posibilidad de reemplazo sencilla. Si una CA intermedia se ve comprometida o necesita ser reemplazada, no afecta a la raiz. Se crea una nueva CA sin tener que reinstalar toda la infraestructura.
- Mayor escalabilidad. Esta estructura permite crear diferentes CAs intermediaas para distintos propósitos sin multiplicar las raíces de confianza.

3.4. Implementación

Se ha realizado la implementación mediante OpenSSL en línea de comandos:

1. Creación de la Autoridad Raiz (Root CA):

Se crea la identidad principal que firmará a las intermedias. Se usa una clave de 4096 bits y se autofirma.

- Generar la clave privada:

```
openssl genrsa -out root-ca.key 4096
```

- Generar el certificado raiz autofirmado:

```
openssl req -x509 -new -nodes -key root-ca.key -sha256 -days 7300 -out root-ca.crt -subj "/C=US/ST=Local/L=Local/O=Root Authority/CN=LocalRootCA"
```

2. Creación de la Autoridad Intermedia:

La entidad operativa. Se crea una solicitud y es la CA raiz quien la firma para darle validez.

- Generar la clave privada de 4096 bits:

```
openssl genrsa -out ca.key 4096
```

- Generar solicitud de firma:

```
openssl req -new -key ca.key -out ca.csr -subj "/C=US/ST=Local/L=Local/O=Local CA/CN=LocalDevCA"
```

- Firmar el certificado, emitido por la CA raiz:

```
openssl x509 -req -in ca.csr -CA root-ca.crt -CAkey root-ca.key -CAcreateserial -out ca.crt -days 3650 -sha256 -extfile ca-ext.cnf
```

3. Emisión de Certificado de Servidor:

Se genera el certificado para localhost. Aqui la clave será de 2048 bits para mejorar el rendimiento del handshake TLS.

- Generar clave privada de 2048 bits:

```
openssl genrsa -out server.key 2048
```

- Generar la solicitud:

```
openssl req -new -key server.key -out server.csr -subj "/C=US/ST=Local/L=Local/O=Local Server/CN=localhost"
```

- Firmar el certificado, emitido por la CA intermedia:

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 365 -sha256 -extfile server-ext.cnf
```

4. Emisión del Certificado de Cliente:
   - Generar la clave privada, de 2048 bits:

     ```
     openssl genrsa -out client.key 2048
     ```

   - Generar la solicitud:

```
openssl req -new -key client.key -out client.csr -subj "/C=US/ST=Local/
L=Local/O=Local Client/CN=client"
```

- Firmar el certificado, emitido por la CA intermedia:

```
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -
out client.crt -days 365 -sha256 -extfile client-ext.cnf
```

- Empaquetado PKCS12. Se une la clave privada y el certificado público en un solo archivo protegido por una contraseña[4].

```
openssl pkcs12 -export -in client.crt -inkey client.key -name 'client' -out
client.p12
```

5. Construcción de Cadenas de Certificados:

Para el despliegue en el servidor, es necesario concatenar los certificados en un orden específico para crear la cadena de confianza completa.

- Crear la cadena del servidor (Certificado Servidor + Certificado CA intermedia + CA Raíz):

```
cat server.crt ca.crt root-ca.crt > server-chain.crt
```

- Crear la cadena del cliente (Cliente + CA Intermedia + CA Raíz):

```
cat client.crt ca.crt root-ca.crt > client-chain.crt
```

- Crear la cadena de confianza completa (CA Raiz + CA Intermedia):

```
cat ca.crt root-ca.crt > ca-chain.crt
```

[4]La contraseña es pkcs12

4. Pruebas realizadas

ID	Descripción	Entrada	Resultado esperado
1	Flujo completo	PDF válido, P12 válido, contraseña correcta y mensaje de texto < 2000 car.	El navegador descarga `DEBUG_nombre.pdf`, la UI muestra «Mensaje almacenado correctamente» y dicho mensaje aparece reflejado en la base de datos.
2	Contraseña P12 incorrecta	PDF válido, P12 válido, contraseña errónea.	Se muestra un mensaje al usuario como que no se ha podido firmar el archivo.
3	Archivo de entrada no es PDF	Seleccionar una imagen `.png` o `.docx` en el `fileInput` de firma.	Excepción capturada por `PDFLib.load()`. Mensaje en consola/UI: «Failed to parse PDF header» o similar. No se descarga nada.
4	Validación Estricta de Firma	Abrir el archivo `DEBUG_*.pdf` generado en el Test 1 en un lector compatible	Aparece el archivo firmado, pero con un emisor irreconocido debido a la emisión en local.
5	P12 sin clave privada	Un archivo `.p12` exportado solo con certificados públicos (sin la private key).	Se muestra un mensaje de error al usuario al intentar enviar el mensaje como que el certificado no contiene una clave privada.
6	Exceso de longitud de mensaje	Texto del mensaje mayor que 2500 caracteres.	Petición devuelve un mensaje de error indicándoselo al usuario.
7	Token de sesión inválido/ caducado	Modificar `localStorage` manualmente con un token falso antes de enviar.	Se comprueba en el backend que el token del cliente sea válido, y no se almacena nada en la base de datos, al no poder verificarlo.
8	Overflow del hueco de firma	Reducir temporalmente en JS `SIGNATURE_LENGTH = 100` e intentar firmar.	El JS lanza error antes de enviar: «La firma es más grande que el hueco reservado».
9	Inyección de caracteres en el PDF	Usar un PDF que contenga la cadena «1234567890» en su texto visible.	El algoritmo de búsqueda `findStringInUint8` debe encontrar el marcador del ByteRange (el diccionario) y no confundirse con el texto del contenido. El PDF resultante debe abrirse sin errores.
10	Integridad del Multipart	Envío correcto del formulario.	Al backend le llega la petición correctamente sin errores, y el archivo PDF sigue íntegro.