Universidad Carlos III de Madrid

Escuela Politécnica Superior

Departamento de Tecnología Electrónica

Proyecto de Fin de Carrera

HERRAMIENTAS DE VALIDACIÓN DE CERTIFICADOS EN PKI CON TARJETA INTELIGENTE

INGENIERÍA SUPERIOR DE TELECOMUNICACIÓN

Autor: Luis Salvador del Pozo

Director: Aitor Mendaza Ormaza

Tutor: Raúl Sánchez Reíllo

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Contexto

Este documento recoge la información de desarrollo y el trabajo realizado para el proyecto de fin de carrera titulado "Herramientas de validación de certificados en PKI con tarjeta inteligente", dirigido por el Dr. Raúl Sánchez Reíllo y realizado por Luis Salvador del Pozo, desarrollado en el Grupo Universitario de Tecnologías de Identificación (GUTI) perteneciente al Departamento de Tecnología Electrónica de la Universidad Carlos III de Madrid.

Dicho grupo trabaja en la elaboración y desarrollo de numerosos proyectos relacionados con todos los aspectos pertenecientes al campo de la biometría como son: sistemas de identificación y verificación, análisis y mejora de estos, medidas de seguridad, control de acceso a sistemas y equipos, desarrollo de hardware biométrico y estandarización.

1.2 Objetivos

El objetivo de este proyecto es el de proveer de un sistema de generación de firma digital y de validación de firma para identificar de forma unívoca al remitente. La firma se realizará aprovechando una infraestructura de clave pública existente de una tarjeta inteligente de identificación, TII de aquí en adelante.

Haciendo uso de los certificados presentes en una TII con infraestructura de clave pública ya existente, y de su módulo criptográfico, se puede conseguir un sistema de firma que provea de identificación no repudiable del firmante de un archivo sin necesidad de crear una nueva infraestructura. De esta forma se evita el proceso de expedición de certificados para cada uno de los participantes, anulando costes de implantación y posibles problemas de seguridad a la hora de distribuir las claves y establecer una autoridad de confianza que emita los certificados.

Los únicos requisitos son, por lo tanto, que el firmante posea una TII, conozca su clave personal, en caso de que la TII tenga configurada una clave de acceso a su almacén de certificados, tenga instalado el software necesario para trabajar con la TII en cuestión y posea un lector de tarjetas inteligentes válido (compatible con la TII). Para el proceso de validación de firma digital no es necesario nada de lo anterior ya que en el mismo archivo firmado se incluye toda la información necesaria para comprobar la validez de la firma.

1.3 Aplicación

La aplicación desarrollada ofrece las siguientes funciones al usuario que la ejecute:

- Firma de datos presentes en el sistema de archivos del ordenador desde el que se ejecuta la aplicación. A partir de un archivo de datos contenido en el ordenador en el que se ejecuta la aplicación se genera un nuevo archivo que contiene los datos originales del primer archivo junto con su firma digital.
- Firma de datos obtenidos de un formulario genérico. El contenido del formulario se transforma al formato *Portable Document Format*, comúnmente conocido como *PDF*, y se firma internamente (en el documento *PDF*). Posteriormente se trata el archivo *PDF* firmado como un archivo de datos y se procede a su firma como si se tratara del tipo de firma definido en el punto anterior: se genera un archivo que contiene los datos del *PDF* firmado internamente junto con la firma digital calculada sobre éste.
- Validación de archivos de firma digital creados con esta aplicación.

Los datos almacenados en un archivo informático no difieren en formato de cualquier otro archivo sea cual sea su contenido, salvo en el nombre, tamaño y extensión. Es por ello que la aplicación desarrollada en este proyecto no se limita únicamente a la firma de datos de un tipo concreto, sino que se puede aplicar a cualquier archivo presente en el sistema de archivos presente en el ordenador que ejecuta la aplicación.

El usuario de la aplicación interactúa con el programa a través de una sencilla interfaz gráfica a través de la cual se puede elegir la función a utilizar y los archivos a usar. Esta interfaz gráfica es la encargada también de solicitar al usuario el código PIN de su TII, en caso de que así esté configurado, para acceder al almacén de certificados de su carnet.

La aplicación ha sido desarrollada enteramente en lenguaje Java, versión 1.6.11, con el entorno de desarrollo NetBeans 6.1 y 6.5. Puede ejecutarse tanto de forma local mediante Java Network Launching Protocol (JNLP) o a través de un navegador con Java habilitado.

1.4 Herramientas utilizadas

A la hora de desarrollar este proyecto se ha hecho uso de las siguientes librerías de java, todas ellas de código abierto y publicadas bajo sus correspondientes licencias:

- Commons Codec 1.3 de Apache. Publicada bajo la licencia Apache License 2.0 (http://www.apache.org/licenses/LICENSE-2.0). Provee implementaciones de codificadores y decodificadores, como Base64, Hex64, etc.
- iText 2.1.3 de Bruno Lowagie. Publicada bajo licencia Mozilla Public License Version 1.1 (http://www.mozilla.org/MPL/). Provee las clases necesarias para trabajar con documentos PDF.
- Bouncy Castle Crypto API 1.41. Publicada sin limitaciones. Provee clases necesarias para el correcto funcionamiento de ciertas funciones de iText.

El uso de estas librerías no implica necesidad alguna para el usuario de la aplicación ya que, para facilitar el despliegue, se han incluido las fuentes necesarias en el archivo .jar de la aplicación.

1.5 Estructura de la memoria.

La presente memoria sigue la siguiente estructura:

- Capítulo 1: Breve introducción y planteamiento de los objetivos y el marco en el que se realiza el proyecto

- Capítulo 2: Estado del arte de los sistemas de cifrado y clave pública. Breve descripción de los principales estándares de datos criptográficos, con especial interés en XAdES-BES.

- Capítulo 3: Descripción de las tarjetas inteligentes de identificación y sus principales rasgos comunes. Definición de la infraestructura de clave pública típica que respalda a las tarjetas inteligentes de identificación

- Capítulo 4: Aplicación desarrollada. Descripción de los requisitos de uso y la funcionalidad de la herramienta desarrollada en el ámbito de este proyecto. En este capítulo se incluyen además una serie de ejemplos que ilustran el funcionamiento de la aplicación. También se describe el formato de salida de la aplicación.

- Capítulo 5: Configuración de la herramienta. En este capítulo se explican los requisitos de despliegue y las distintas posibilidades de configuración a la hora de poner la herramienta a disposición de los usuarios. También se explica cómo modificar el formato del documento *PDF* de salida.

- Capítulo 6: Estructura del código fuente. Este capítulo ofrece una visión general sobre la estructura del código fuente que compone la herramienta desarrollada. De esta forma se puede alcanzar una mayor comprensión acerca de cómo se obtiene la funcionalidad alcanzada.

- Capítulo 7: Conclusiones y líneas futuras. Este capítulo incluye las conclusiones alcanzadas tras la realización del proyecto, los criterios de diseño utilizados en el desarrollo de la herramienta y posibles líneas futuras de trabajo que amplíen o mejoren la funcionalidad de la misma.

- Capítulo 8: Este capítulo recoge un desglose de la estimación de los distintos costes en que se habría incurrido con la realización del presente proyecto.

- Finalmente se incluyen también las referencias y bibliografía utilizadas en la elaboración del presente proyecto.

2 Sistemas de cifrado y clave pública

2.1 Introducción

2.1.1 Necesidad de cifrado

Dentro de la nueva sociedad de la información el papel que juega Internet como plataforma fundamental sobre la que desarrollar el futuro de los negocios y los trámites con organizaciones tanto públicas cómo privadas es vital. Sin embargo, paralelo al desarrollo de la funcionalidad y actividades de la red, también se ha generado un nuevo tipo de ataques y explotación de vulnerabilidades que deben ser tratados para facilitar la correcta evolución de los servicios disponibles online por parte de organizaciones públicas y privadas. Destacamos a continuación los ataques más comunes:

- Suplantación de identidad: un intruso malintencionado puede hacerse pasar por una identidad diferente. Este tipo de ataques son especialmente peligrosos en el caso de operaciones bancarias online, donde dicho intruso puede realizar transferencias en nombre de la persona a la que suplanta.

- Reactuación: uno o varios mensajes legítimos son repetidos para producir un efecto no deseado e ilegítimo. Un ejemplo sencillo de este tipo de ataque es, siguiendo con el ejemplo bancario, hacer un ingreso en una cuenta bancaria repetidas veces aunque sólo se haya realizado una transferencia.

- Modificación de mensajes: una parte de un mensaje legítimo es alterada por una entidad malintencionada, o bien los mensajes son retrasados o reordenados con

la intención de producir un efecto no autorizado. De nuevo, utilizando el ejemplo bancario, el usuario malintencionado podría modificar un mensaje que suponga un ingreso de 100€ en una cuenta ajena de forma que quede en un ingreso de 1000€ en otra cuenta de su elección.

Ante estos ataques y amenazas se deben de proveer ciertos servicios o medidas de seguridad:

- Autenticación: se debe proveer al sistema en cuestión la garantía de que con quién se mantiene la comunicación es realmente quien afirma ser.

- Integridad: se debe proveer de mecanismos que permitan saber si la información recibida en una comunicación ha sido manipulada o no.

- No repudio: se debe proveer de la capacidad de asegurar que las partes que participen en una transacción no puedan negar haberlo hecho.

- Confidencialidad: ser capaz de garantizar que sólo aquellos a los que va dirigida la información tienen acceso a ella.

2.1.2 Concepto de Criptografía

Según el diccionario de la Real Academia la palabra *criptografía* proviene del griego *kriptos*, que significa escondido, y *graphos*, que significa escritura, y la define como "el arte de escribir con clave secreta o de un modo enigmático". La finalidad de la criptografía es, en primer lugar, garantizar el secreto en la comunicación entre dos entidades (personas, organizaciones, etc.) y, en segundo lugar, asegurar que la información que se envía es auténtica en un doble sentido: que el remitente sea realmente quien dice ser y que el contenido del mensaje enviado, habitualmente denominado criptograma, no haya sido modificado en su tránsito.

En la actualidad, la criptografía no sólo se utiliza para comunicar información de forma segura ocultando su contenido a posibles intrusos. Una de las ramas de la criptografía que más ha revolucionado el panorama actual de las tecnologías de la información es la de la firma digital: tecnología que busca asociar al emisor de un mensaje con su contenido de forma que aquel no pueda posteriormente repudiarlo. Más adelante se estudia este caso con detalle.

2.1.3 Tipos de Cifrado

Dentro de la criptografía existe una clara división en dos tipos de cifrado en función del tipo de claves usadas para el procesado de los mensajes. El proceso básico de cifrado es el siguiente:

La entidad A cifra el mensaje T utilizando una clave de cifrado K1, resultando un mensaje codificado C. La entidad A envía entonces el mensaje C a la entidad destinatario B. Ésta decodificará el mensaje con la clave de descifrado K2. De esta forma la entidad A debe conocer la clave K1 y la entidad B debe conocer la clave K2. En la forma de estas dos claves radican las diferencias entre los dos tipos básicos de cifrado que se contemplan a continuación:

2.1.3.1 Cifrado simétrico

En el caso de cifrado simétrico la clave de cifrado y descifrado es la misma (K1 es igual que K2). Tiene como ventaja que el algoritmo es extremadamente sencillo y rápido. Sin embargo para la situación que se contempla en este estudio resulta absolutamente ineficiente e inadecuada para una red pública como Internet, por los problemas que supone la distribución de claves inicial (ambos participantes en una transacción deben conocer la clave).

Aunque el problema de la distribución de claves se pudiese solventar, seguiría existiendo otro problema, el de la inalteración de la clave. Al permanecer inalterada durante toda su vigencia es muy susceptible a ataques de fuerza bruta. Sin embargo esto no es óbice para que en aplicaciones como inicios de sesión o acuerdos de clave, se utilice cifrado simétrico. La clave generada para una sesión expira al finalizar ésta, con lo que la inalteración de la clave no es problemática.

2.1.3.2 Cifrado asimétrico

Los sistemas de cifrado asimétrico, también conocidos como cifrados de clave pública, otorgan a cada entidad usuario un par de claves, una clave pública y otra clave privada. El hecho de utilizar dos claves, una para cada paso de la comunicación (codificación y descodificación) hace que este tipo de cifrado se conozca como criptografía asimétrica, en contraposición a los sistemas simétricos de clave secreta que usan una única clave para ambos procesos.

Los sistemas de cifrado asimétricos se inventaron con el fin de evitar por completo el problema del intercambio de claves de los sistemas de cifrado simétricos. Con las claves públicas no es necesario que el remitente y el destinatario se pongan de acuerdo en la clave a emplear. Todo lo que se requiere es que, antes de iniciar la comunicación secreta, el remitente consiga una copia de la clave pública del destinatario. Es más, esa misma clave pública puede ser usada por cualquiera que desee comunicarse con su propietario. Por tanto, se necesitarán sólo *n* pares de claves por cada *n* personas que deseen comunicarse entre sí.

La clave pública se puede distribuir libremente a cualquier entidad con la que se

quiere establecer comunicación, pero la clave privada debe permanecer conocida tan sólo para la entidad propietaria de dicha clave. Ambas claves están relacionadas matemáticamente entre sí pero conocer una clave pública no aporta conocimientos sobre la clave privada con la que dicha clave pública está vinculada. Lo que se cifra con una de las dos claves de un par se descifra únicamente con la otra clave no usada en el proceso de cifrado.

El hecho de que se utilice una clave distinta para cifrado y descifrado conlleva una gran ventaja respecto al cifrado simétrico: la distribución de claves es más sencilla y segura, ya que cada par de claves se distribuye únicamente a su entidad propietaria. Para la comunicación entre dos entidades poseedoras de pares de claves basta con que intercambien sus claves públicas, acción que no supone ningún riesgo ya que no hace visible la clave privada. Sin embargo el sistema de cifrado asimétrico tiene varias desventajas respecto al cifrado simétrico:

- Para una misma longitud de clave y mensaje requiere una mayor potencia de cálculo, lo que lleva a un mayor tiempo de procesamiento.
- Las claves deben ser de mayor tamaño que las simétricas para ofrecer la misma resistencia a ataques de fuerza bruta.
- El mensaje cifrado ocupa más espacio que el original.

A pesar de estas desventajas el cifrado asimétrico o de clave pública cubre varios de los aspectos mencionados con anterioridad: es capaz de aportar confidencialidad mediante el cifrado con la clave pública y de conseguir con algoritmos de firma digital el no repudio y la autenticación del remitente.

Un problema importante que se plantea en la criptografía de clave pública es la distribución de claves públicas, de forma que quede demostrado que una clave pública es auténtica y que no ha sido manipulada o sustituida por terceros maliciosos. La manera típica de resolver este problema es mediante el uso de una infraestructura de clave pública, PKI (*Public Key Infrastructure*).

2.2 Aplicaciones Directas: Autenticación y Firma

En los sistemas de cifrado con clave pública, debido a la asimetría inherente al proceso de comunicación, ambas claves son necesarias, pero la clave privada debe permanecer secreta en manos del propietario. De esta forma este tipo de criptografía tiene varios usos, como los que se explican en este apartado. En cada uno de estos posibles usos se alternan las funciones que cumplen la clave privada y la clave pública, como se muestra a continuación.

Un primer uso puede ser el siguiente: la entidad poseedora de un par de claves puede cifrar datos con su clave privada y enviarlos a otra entidad conocedora de la clave

pública de la primera. En este caso, la entidad que recibe el mensaje cifrado puede descifrarlo y confirmar que dicho mensaje proviene de quien dice enviarlo, ya que si ha conseguido descifrar el mensaje con éxito, significa que ha sido cifrado con la correspondiente clave privada, lo que identifica al emisor. Para esto se necesita por supuesto que el formato del mensaje sea reconocible o que la entidad receptora sepa identificar cuándo un mensaje descifrado tiene sentido. Éste es el proceso que se usa en firmas digitales.

Otro uso de este tipo de cifrado es el contrario, asegurar que el único destinatario que pueda recibir un mensaje sea la entidad a la que corresponde la clave pública usada en el cifrado. Si la entidad A cifra un mensaje con la clave pública del destinatario B, tan sólo B podrá descifrarlo con su clave privada que se asume secreta. De esta forma sólo el auténtico poseedor de la clave privada de B podrá responder correctamente a un reto lanzado por la entidad A ante la que se quiere autenticar.

2.2.1 Firma Digital

La firma digital de un documento es el resultado de aplicar cierto algoritmo matemático, denominado función hash, a su contenido, y seguidamente aplicar un cifrado de tipo asimétrico (en el que se emplea la clave privada del firmante) al resultado de la operación anterior, generando la firma electrónica o digital. El software de firma digital puede además efectuar varias validaciones, entre las cuales se encuentran:

- Vigencia del certificado digital del firmante. El certificado digital es un documento digital mediante el cual un tercero de confianza garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública.

- Revocación del certificado digital del firmante (puede ser por OCSP o CRL)

- Inclusión de sellado de tiempo (mecanismo on-line que permite demostrar que una serie de datos han existido y no han sido alterados desde un instante específico en el tiempo).

La función hash es un algoritmo matemático que permite calcular un valor resumen de los datos a ser firmados digitalmente. Funciona en una sola dirección, es decir, no es posible calcular los datos originales a partir del valor del resumen. Cuando la entrada es un documento, el resultado de la función es un valor que identifica inequívocamente al texto. Si se adjunta este valor al texto, el destinatario puede aplicar de nuevo la función y comprobar su resultado con el que ha recibido.

Para que sea de utilidad, la función hash debe satisfacer dos importantes requisitos. Primero, encontrar dos documentos cuyo valor para la función hash sea idéntico debe ser muy costoso computacionalmente. Segundo, dado uno de estos valores, debe ser imposible producir un documento con sentido que de lugar a ese hash.

Existen funciones hash específicamente diseñadas para satisfacer estas dos importantes propiedades. SHA y MD5 son dos ejemplos de este tipo de algoritmos.

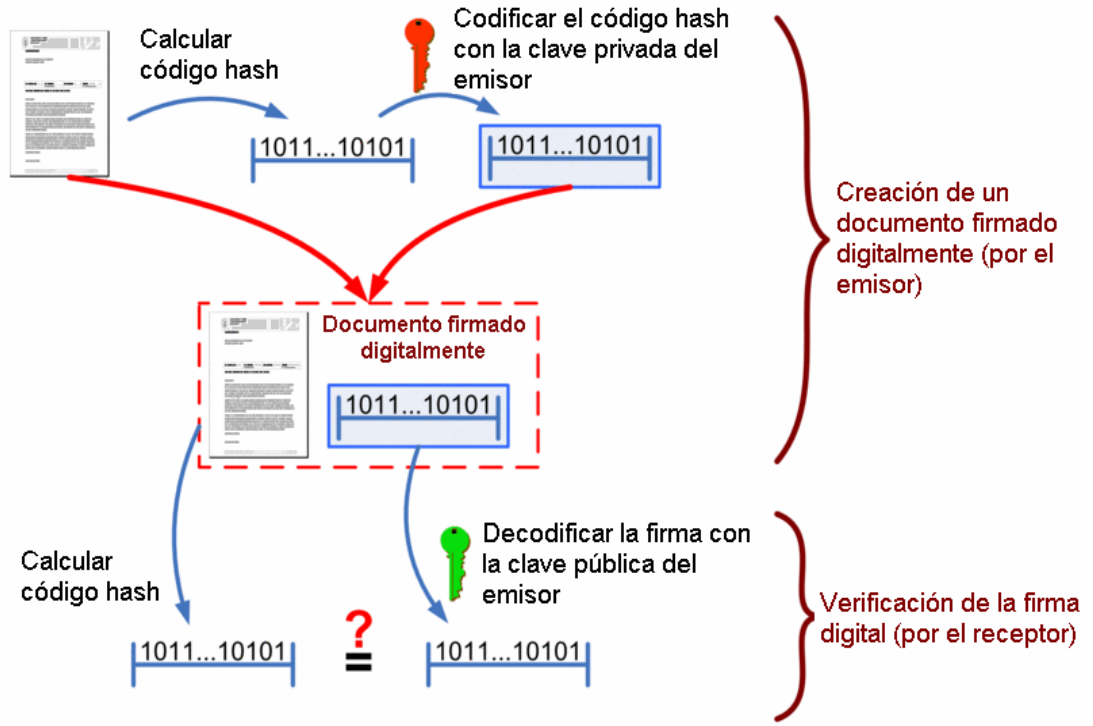

Figura 2-1: Esquema básico de firma digital. Si el hash calculado en recepción no coincide con el de la firma decodificada entonces o bien el documento fue modificado tras la firma o la firma no fue generada con la clave privada del supuesto emisor

Las normas TS 101 733 y TS 101 903 definen los formatos técnicos de la firma electrónica. La primera se basa en el formato clásico PKCS#7 (ver apartado 2.5.2 de este documento) y la segunda en XML-DSig, que consiste en la firma XML especificada por el consorcio W3C (ver apartados 2.5.3 y 2.5.4).

Bajo estas normas se definen tres modalidades de firma:

- Firma básica. Incluye el resultado de operación de hash firmado con la clave privada del firmante, el certificado asociado a la clave privada del firmante e identifica los algoritmos utilizados en el proceso de firma.

- Firma fechada. A la firma básica se añade un sello de tiempo calculado a partir del hash del documento y firmado por una TSA (*Time Stamping Authority*, autoridad de sellado de tiempo en español).

- Firma validada o firma completa. A la firma fechada se añade información sobre la validez del certificado procedente de una consulta de CRL (*Certificate Revocation List*, lista de revocación de certificados en español) o de OCSP (*Online Certificate Status Protocol*, en español protocolo en línea de estado de

certificados) realizada a la CA (*Certification Authority*, autoridad de certificación) pertinente.

La firma completa libera al receptor de la firma del problema de ubicar al prestador de servicios de certificación responsable de gestionar las consultas de validez del certificado del firmante, y de determinar los procedimientos de validación disponibles.

2.3 Infraestructura de Clave Pública, PKI

En criptografía, una infraestructura de clave pública PKI (*Public Key Infrastructure*) es una combinación tanto de hardware y software, como de políticas y procedimientos de seguridad que permiten la ejecución con garantías de operaciones criptográficas. Dichas operaciones pueden ser el cifrado, la firma digital o el no repudio de transacciones electrónicas.

Una infraestructura de clave pública (PKI) establece una unión entre claves públicas y sus respectivas identidades de usuario por medio de una Autoridad de Certificación (CA por sus siglas en inglés, *Certification Authority*). La identidad de usuario debe ser única para cada Autoridad de Certificación. Este vínculo usuario-claves se establece a través de un proceso de registro y expedición, que, dependiendo del nivel de garantías que posea el vínculo, será realizado mediante software en una Autoridad de Certificación o bajo supervisión humana.

La infraestructura PKI permite a los usuarios autenticarse frente a otros usuarios y usar la información de los certificados de identidad (por ejemplo, las claves públicas de otros usuarios) para cifrar y descifrar mensajes, firmar digitalmente información, garantizar el no repudio de un envío, y otros usos.

En una operación criptográfica que use infraestructura PKI, intervienen conceptualmente como mínimo las siguientes partes:

- Un usuario que inicia la operación
- Una serie de autoridades, que den fe de la ocurrencia de la operación y garanticen la validez de los certificados implicados en la misma (autoridad de certificación, Autoridad de registro y sistema de Sellado de tiempo).
- Un destinatario de los datos cifrados/firmados/enviados, garantizados por parte del usuario que inició la operación (puede ser él mismo).

Las operaciones criptográficas de clave pública son procesos en los que se utilizan algoritmos de cifrado conocidos y accesibles para todos. Por este motivo la seguridad que puede aportar la tecnología PKI, está fuertemente ligada a la privacidad de la llamada clave privada y los procedimientos operacionales o políticas de seguridad que se aplican (ver apartado 2.3.1.7, *Autoridad de Aprobación de Políticas, PAA*).

Es de destacar la importancia de las políticas de seguridad en esta tecnología, puesto que ni los dispositivos más seguros ni los algoritmos de cifrado más fuertes sirven de nada si, por ejemplo, una copia de la clave privada protegida por una tarjeta inteligente criptográfica se guarda en un disco duro convencional de un PC conectado a Internet.

2.3.1 Componentes de la Infraestructura de Clave Pública PKI

En este apartado se definen los componentes más habituales de los que consta una infraestructura de clave pública PKI. Puede darse el caso de que una misma entidad cumpla los papeles de más de uno de los componentes descritos en este apartado.

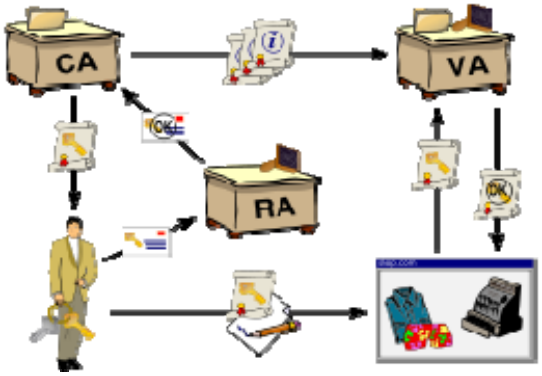

Figura 2-2: Esquema básico de infraestructura de clave pública PKI [10]. CA: Autoridad de Certificación, RA: Autoridad de Registro, VA: Autoridad de Validación

2.3.1.1 Usuario Suscriptor

Se entiende por usuario suscriptor de la PKI al usuario que voluntariamente confía y hace uso de los certificados de los que es titular. Posee (al menos) un par de claves (pública y privada) junto con un certificado asociado a su clave pública y utiliza un conjunto de aplicaciones que hacen uso de la tecnología PKI para validar firmas digitales, cifrar documentos para otros usuarios, etc.

2.3.1.2 Autoridad de Certificación (CA)

La autoridad de certificación CA es la entidad de confianza que da legitimidad a la relación de una clave pública con la identidad de un usuario o servicio. Se encarga de emitir y revocar certificados: expide un certificado de clave pública para cada usuario, con el que la identidad del usuario asociada a su clave pública, así como las condiciones de validez y otros atributos, son infalsificables.

La CA, por sí misma o mediante la intervención de una autoridad de registro RA,

verifica la identidad del solicitante de un certificado antes de su expedición o, en caso de certificados expedidos con la condición de revocados, elimina la revocación de los certificados al comprobar dicha identidad. Los certificados son documentos que recogen ciertos datos de su titular y su clave pública. Se encuentran firmados electrónicamente por la CA, utilizando la clave privada de ésta. La CA es un tipo particular de prestador de servicios de certificación, que legitima ante terceros, que confían en sus certificados, la relación entre la identidad de un usuario y su clave pública. La confianza de los usuarios en la CA es importante para el correcto funcionamiento del servicio y justifica la filosofía de su empleo, pero no existe un procedimiento normalizado para demostrar que una CA merece dicha confianza.

2.3.1.3 Autoridad de Registro (RA)

La autoridad de registro RA (del inglés *Registration Authority*) es la responsable de verificar el vínculo entre los certificados (concretamente, entre la clave pública del certificado) y la identidad de sus titulares. Es una parte opcional de la PKI que mantiene las identidades de aquellos usuarios de los que las autoridades de certificación pueden expedir certificados digitales. La autoridad de registro es la entidad autorizada por la autoridad de certificación para proveer administración de tiempos de validez de los certificados.

2.3.1.4 Repositorios

Los repositorios: son las estructuras encargadas de almacenar la información relativa a la PKI. Los dos repositorios más importantes son el repositorio de certificados y el repositorio de listas de revocación de certificados. En una lista de revocación de certificados (o, en inglés, CRL, *Certificate Revocation List*) se incluyen todos aquellos certificados que por algún motivo han dejado de ser válidos antes de la fecha establecida dentro del mismo certificado.

2.3.1.5 Autoridad de Validación (VA)

La autoridad de validación (o, en inglés, VA, *Validation Authority*) es la encargada de comprobar la validez de los certificados digitales.

2.3.1.6 Autoridad de Sellado de Tiempo (TSA)

La autoridad de sellado de tiempo (o, en inglés, TSA, *Time Stamping Authority*) es la encargada de firmar documentos con la finalidad de probar que existían y que no han sido alterados, antes de un determinado instante de tiempo.

De acuerdo con el estándar RFC 3161, un sellado de tiempo de confianza debe ser expedido por un tercero de confianza que actúa como TSA. Se usa para probar la

existencia de ciertos datos antes de un determinado momento (por ejemplo contratos, datos de investigación, expedientes médicos,…) sin la posibilidad de que el propietario pueda adelantar los sellos de tiempo. Múltiples autoridades de sellado de tiempo pueden ser usadas para aumentar la confianza y reducir la vulnerabilidad.

El nuevo estándar ANSI_ASC_X9.95_Standard para sellados de tiempo de confianza extiende el estándar RFC 3161 con requisitos de seguridad a nivel de datos para garantizar integridad de los mismos a través de una fuente de tiempo de confianza y que sea probable para cualquier tercero. Este estándar se ha aplicado para autenticar datos firmados digitalmente para conformidad regulatoria, transacciones financieras y evidencias legales.

La técnica de sellado de tiempo se basa en firmas digitales y funciones de hash. En primer lugar se calcula el hash de los datos a sellar. Este hash es enviado a la autoridad de sellado de tiempo, TSA, que concatena un sellado de tiempo al hash y calcula el hash de la concatenación resultante. Este segundo hash es firmado digitalmente con la clave privada de la TSA. El conjunto sellado de tiempo - hash firmado se envía al solicitante del sellado de tiempo, que lo almacena junto con los datos originales. Esta técnica es confidencial para el solicitante del sellado de tiempo, dado que la TSA no llega a ver nunca los datos originales, tan sólo su hash, a partir del cual no se pueden obtener los datos que lo generan.

2.3.1.7 Autoridad de Aprobación de Políticas (PAA)

La autoridad de aprobación de políticas PAA, por las siglas de *Policy Approval Authority* es una entidad a la que, sujeto a criterios de cada PKI, determinados usuarios de la infraestructura pueden ser invitados a formar parte. Esta autoridad debe ejercer su rol de entidad neutral y representativa a cargo de definir las políticas que se aplicarán a la misma infraestructura a la que pertenece. Por tanto la composición y las reglas de la PAA son esenciales para garantizar la neutralidad y fiabilidad de toda la infraestructura. Se podrá buscar también participación de terceras partes que no sean necesariamente usuarios directos de la PKI como medio para asegurar más si cabe la neutralidad y confianza de la estructura. Estas terceras partes pueden ser organizaciones internacionales, organizaciones no gubernamentales, gobiernos, compañías tecnológicas entre otros.

La PAA debe emprender una labor constante de establecimiento y mantenimiento de políticas y prácticas que se apliquen a la PKI, lo que conlleva que se tengan siempre en consideración desarrollos tecnológicos, legales, políticos, culturales o económicos que afecten o puedan afectar a la PKI, de manera que se sigan persiguiendo de la mejor manera posible los objetivos iniciales de la PKI.

2.4 Certificados Digitales

Un certificado digital es un documento digital mediante el cual un tercero de confianza (una CA) garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública.

Si bien existen variados formatos para certificados digitales, los más empleados siguen el estándar ITU-T X.509. El certificado contiene el nombre de la entidad certificada, número de serie, fecha de expiración, una copia de la clave pública del titular del certificado (utilizada para la verificación de su firma digital) y la firma digital de la autoridad emisora del certificado, de forma que el receptor pueda verificar que esta última ha establecido realmente la asociación.

Un certificado emitido por una entidad de certificación autorizada, además de estar firmado digitalmente por ésta, debe contener, por lo menos, los siguientes campos:

- Nombre, dirección y domicilio del suscriptor.
- Identificación del suscriptor nombrado en el certificado.
- El nombre, la dirección y el lugar donde realiza actividades la entidad de certificación.
- La clave pública del usuario.
- La metodología para verificar la firma digital del suscriptor impuesta en el mensaje de datos.
- El número de serie del certificado.
- Fecha de emisión y expiración del certificado.

Cualquier individuo o institución puede generar un certificado digital, pero si éste emisor no es reconocido por quienes interactúen con el propietario del certificado, el valor del mismo es prácticamente nulo. Por ello, los emisores deben acreditarse. Así se denomina al proceso por el cuál entidades reconocidas, generalmente públicas, otorgan validez a la institución certificadora, de forma que su firma pueda ser reconocida como fiable, transmitiendo esa fiabilidad a los certificados emitidos por la citada institución.

2.4.1 Estándar X.509

En criptografía, el estándar X.509 es un estándar de la ITU-T para infraestructuras de clave pública PKI para *Single Sign-On* (SSO) y para infraestructuras de administración de privilegios (PMI, por sus siglas en inglés, *Privilege Management Infrastructure*). El estándar X.509 especifica, entre otras cosas, los formatos estándar para certificados de clave pública, listas de revocación de certificados, certificados de atributos y un algoritmo de validación de árboles de certificación.

El estándar X.509 fue publicado en julio de 1988, asociado inicialmente con el estándar X.500. Asume un sistema estrictamente jerárquico de autoridades de certificación para expedir certificados. Este enfoque contrasta con otros modelos de

webs de confianza, como PGP (*Pretty Good Privacy*), en los que cualquiera, no sólo autoridades de certificación especiales, pueden firmar y de esa forma atestiguar la validez de certificados de claves de otros.

La tercera versión del estándar X.509 incluye la flexibilidad de admitir otras topologías como puentes y mallas en los árboles de certificación., de forma que se adapta mejor a la organización más flexible de Internet.

En un sistema basado en el estándar X.509, una autoridad de certificación emite un certificado que vincula una clave pública a un nombre distintivo (*Distinguished Name*), según un enfoque X.500 tradicional en el que se basa el estándar X.509, o a un nombre alternativo (*Alternative Name*).

El *Distinguished Name* DN es un nombre globalmente único dentro de una PKI que cualquiera podría usar para referirse a la entidad a la que pertenece. El estándar X.509 mantiene la definición del estándar X.500 para este campo. El *Alternative Name* es novedad respecto al estándar X.500 que añade flexibilidad al vincular una clave pública a un nombre, ya que permite asociarlo a una dirección de correo electrónico o una entrada de servidor de nombres de dominio DNS.

Navegadores como Internet Explorer, Mozilla, Opera, Safari, etc., tienen certificados raíz preinstalados. De esta forma determinados certificados para comunicaciones sobre SSL, de grandes proveedores que han pagado por este privilegio, funcionarán de forma instantánea.

El estándar X.509 incluye también métodos para implementar listas de revocación de certificados aunque el método aprobado por la IETF de comprobar la validez de certificados es mediante el protocolo de estado de certificado en línea, OCSP por sus siglas en inglés.

En lo que a seguridad respecta, en 2005 A. Lenstra y B. de Wegger demostraron cómo usar colisiones de hash para construir dos certificados X.509 basados en función *hash* MD5 que contienen firmas idénticas pero difieren en la clave pública. Dicho ataque se realizó basándose en un ataque de colisión en la función de hash MD5. Esta vulnerabilidad es inherente al uso de la función *hash* MD5. Los certificados X.509 basados en la función *hash* SHA-1 se presuponen seguros.

2.4.1.1 Estructura de un Certificado X.509

La estructura de un certificado digital X.509 v3 es la siguiente:

- Certificado
  - Versión
  - Número de Serie
  - Identificador de Algoritmo
  - Emisor

  - Validez
    - No Antes De
    - No Después De
  - Sujeto
  - Información de Clave Pública del Sujeto
    - Algoritmo de Clave Pública
    - Clave Pública del Sujeto
  - Identificador Único del Emisor (opcional)
  - Identificador Único del Sujeto (opcional)
  - Extensiones (opcional)

    + ...
- Algoritmo de Firma del Certificado
- Firma del Certificado

2.4.2 Lista de Revocación de Certificados CRL (*Certificate Revocation List*)

Cuando una autoridad de certificación emite un certificado digital, lo hace con un periodo máximo de validez que oscila entre tres y cinco años. El objetivo de este periodo de caducidad es obligar a la renovación del certificado para adaptarlo a los cambios tecnológicos. Así se disminuye el riesgo de que el certificado quede comprometido por un avance tecnológico. La fecha de caducidad viene indicada en el propio certificado digital.

Sin embargo, existen otras situaciones que pueden invalidar el certificado digital aún cuando no ha caducado, de manera inesperada:

- El usuario del certificado cree que su clave privada ha sido robada.
- Desaparece la condición por la que el certificado fue expedido. Por ejemplo, el cambio de apoderado de una entidad jurídica.
- El certificado contiene información errónea o información que ha cambiado.
- Una orden judicial.
- Etc.

Por tanto, debe existir algún mecanismo para comprobar la validez de un certificado antes de su caducidad. Las CRL son uno de estos mecanismos. Una CRL es una lista de números de serie de certificados digitales, revocados por una autoridad de certificación concreta. Dicha lista está firmada digitalmente por la propia autoridad de certificación.

Cuando un tercero desea comprobar la validez de un certificado debe descargar una CRL actualizada desde los servidores de la misma autoridad de certificación que emitió el certificado en cuestión. A continuación, comprueba la autenticidad de la lista gracias a la firma digital de la autoridad de certificación. Después debe comprobar que el número de serie del certificado que se está verificando, se encuentra en la lista. En caso afirmativo, no se debe aceptar el certificado como válido.

Estrictamente hablando, no es necesario descargar una CRL cada vez que se verifica un certificado. Solamente es necesario cuando no se dispone de la CRL de una entidad de certificación concreta, y cuando dicha lista tiene una cierta antigüedad que aconseja su renovación.

La única ventaja de las CRL es que se pueden consultar sin necesidad de una conexión de datos permanente con cada autoridad de certificación. Basta establecer dicha conexión con cierta periodicidad para descargar las CRL actualizadas. Sin embargo, las desventajas de las CRL son varias:

- Existe el peligro de que un certificado haya sido revocado, pero no aparezca en la CRL del tercero que comprueba su validez. Esto se debe a que la CRL utilizada podría no estar actualizada.

- Si existe responsabilidad legal por el uso de un certificado revocado, no hay forma de demostrar quién es el culpable: el tercero por no comprobar la validez, o la autoridad de certificación por no incluirlo en la CRL a tiempo.

- Las CRL solamente crecen en tamaño, resultando ineficientes para su tratamiento directo.

2.4.3 Online Certificate Status Protocol OCSP

Mediante el protocolo *Online Certificate Status Protocol,* OCSP, se define un método para determinar el estado de revocación de un certificado digital X.509 usando otros medios que no sean el uso de CRL (Listas de Revocación de Certificados). Este protocolo se describe en el RFC 2560 y se encuentra en el registro de estándares de Internet.

Los mensajes OCSP se codifican en ASN.1 y habitualmente se transmiten sobre el protocolo HTTP. La naturaleza de las peticiones y respuestas de OCSP hace que a los servidores OCSP se les conozca como "OCSP responders".

El protocolo OCSP fue creado para solventar ciertas deficiencias de las CRL. Cuando se despliega una PKI (Infraestructura de Clave Pública), es preferible la validación de los certificados mediante OCSP sobre el uso de CRL por varias razones:

- OCSP puede proporcionar una información más adecuada y reciente del estado de revocación de un certificado.

- OCSP elimina la necesidad de que los clientes tengan que obtener y procesar las CRL, ahorrando de este modo tráfico de red y procesado por parte del cliente.

- El contenido de las CRL puede considerarse información sensible, análogamente

a la lista de morosos de un banco.

- Un "OCSP responder" puede implementar mecanismos de tarificación para pasarle el coste de la validación de las transacciones al vendedor, en vez de al cliente.
- OCSP soporta el encadenamiento de confianza de las peticiones OCSP entre los "responders". Esto permite que los clientes se comuniquen con un "responder" de confianza para lanzar una petición a una autoridad de certificación alternativa dentro de la misma PKI.

- Una consulta sobre el estado de un certificado sobre una CRL, debe recorrerla completa secuencialmente para decir si es válido o no. Un "OCSP responder" en el fondo, usa un motor de base de datos para consultar el estado del certificado solicitado, con todas las ventajas y estructura para facilitar las consultas. Esto se manifiesta aún más cuando el tamaño de la CRL es muy grande.

Una diferencia importante entre CRL y OCSP, es que una CRL puede ser almacenada temporalmente para hacer consultas locales. En cambio, para usar OCSP se requiere de conexión con el OCSP *Responder*. Si bien la CRL está disponible sin conexión, mientras más tiempo esté sin actualizarse, menos confiable se hace la información que nos brinda, porque pueden haberse revocado algunos certificados desde la última actualización. Esto da lugar a una diferencia en la eficiencia del uso del ancho de banda para efectos exclusivos de consultas por estado de certificados.

Una petición de OCSP básicamente está compuesta por la versión del protocolo y los identificadores de los certificados que se desean validar. Este identificador está formado por el número de serie, el hash del DN del emisor del certificado y el hash de la clave pública del mismo.

En una misma petición se pueden consultar el estado de varios certificados, incluso pertenecientes a diferentes CA. La firma de la petición es opcional y depende de lo que decida la autoridad de validación OCSP.

Un *Responder* de OCSP puede devolver una respuesta firmada, lo cual significaría que el certificado indicado en la petición es "bueno" (*good*), "revocado" (*revoked*) o "desconocido" (*unknown*). Estas respuestas serán para cada uno de los certificados de los que se ha solicitado la consulta. También puede devolver un código de error, en cuyo caso la respuesta no tendría que estar firmada. Desafortunadamente, el borrador de la primera versión de OCSP (OCSP v.1) es ambigua en el sentido de "desconocido". Podría significar que el sujeto que figura en el certificado es desconocido, o que lo que es desconocido el estado de revocación del certificado.

El formato de petición OCSP soporta extensiones adicionales. Esto permite una adaptación y configuración más extensas a un esquema de PKI en concreto.

2.5 Formatos de Datos Criptográficos

Existen varios formatos de datos criptográficos con los que se puede trabajar. A continuación se definen varios de los formatos más comúnmente utilizados.

2.5.1 Cryptographic Message Syntax CMS

La última versión del estándar de sintaxis de mensaje criptográfico CMS (*Cryptographic Message Syntax*) está definida por la norma RFC3852 de la IETF. Es un estándar utilizado para firmar digitalmente, obtener el *digest*, autenticar, o cifrar arbitrariamente el contenido de un mensaje. Está parcialmente basado en la sintaxis definida anteriormente por PKCS#7.

Este estándar es suficientemente general como para soportar muchos tipos de contenido diferentes. Define un tipo de contenido protegido, *ContentInfo*, que encapsula un único tipo identificado de contenido cada vez. Ese tipo encapsulado puede a su vez proveer de encapsulación a otros tipos de datos.

En este formato se definen seis tipos de contenidos: datos, datos firmados, datos encapsulados o envueltos, datos procesados, datos codificados y datos autenticados, aunque admite más definiciones externas. Los únicos tipos obligatorios para cumplir con las especificaciones de CMS son el contenido de protección *ContentInfo* y los tipos de contenido de datos, datos firmados y datos encapsulados o envueltos; el resto son opcionales.

Como regla general de diseño, cada tipo de contenido permite obtener el resultado del algoritmo de cifrado utilizado haciendo una única pasada sobre los datos que se están procesando (modo *single-pass*) utilizando codificación de longitud indefinida según las reglas básicas de codificación BER (*Basic Encryption Rules*).

Operar con un sólo barrido sobre los datos es especialmente útil si el contenido es grande, se encuentra almacenado en cintas o está siendo obtenido como un flujo de datos desde otro proceso. La gran desventaja de operar en modo *single-pass* es que es muy difícil operar utilizando reglas avanzadas de codificación DER (*Distinguished Encoding Rules*) ya que las longitudes de los distintos componentes no tienen por qué ser conocidas de antemano.

En cualquier caso, los atributos firmados dentro del tipo de contenido de datos firmados y los atributos autenticados dentro del tipo de contenido de datos autenticados necesitan ser transmitidos en forma DER De esta forma se garantiza que los receptores puedan verificar contenidos que contengan uno o mas atributos no reconocidos. Los atributos firmados y los atributos autenticados son los únicos tipos de datos que requieren codificación DER en el formato CMS.

2.5.2 PKCS#7

Estándar de la familia *Public Key Cryptography Standards* PKCS usado para firmar y/o cifrar mensajes en una infraestructura de clave pública, PKI. Está definido por la RFC 2315 de la IETF. Este estándar también es usado para la distribución de certificados.

Al igual que en el caso de CMS, PKCS#7 tiene una sintaxis suficientemente general como para soportar distintos tipos de contenido. También mantiene la definición del tipo de contenido de protección *ContentInfo* y reconoce seis tipos de contenidos definidos, aunque difiere en varios de ellos: datos, datos firmados, datos envueltos, datos firmados y envueltos, datos procesados y datos codificados. Admite el uso de otros tipos de contenido, pero sometido a acuerdo bilateral entre las partes que intercambien contenido.

PKCS#7 define dos clases de tipos de contenido: básica y ampliada. Estos tipos se definen a continuación.

Los tipos de contenido de la clase básica contienen sólo datos, sin mejoras criptográficas. Actualmente, esta clase sólo tiene un tipo de contenido, el tipo de contenido de datos. Los tipos de contenido en la clase ampliada tienen contenido de algún tipo (posiblemente codificado) y otras mejoras criptográficas. Por ejemplo, los contenidos de tipo 'datos envueltos' pueden tener contenido de datos firmados (codificados), que puede tener a su vez contenido de tipo datos.

Los cinco tipos de contenido que no son datos conforman la clase ampliada de tipo de contenido. De esta forma los tipos de contenido de la clase ampliada emplean encapsulación, dando lugar a las expresiones de contenido "exterior" (el que contiene las ampliaciones) y contenido "interior" (el que se ve mejorado o ampliado).

Tal y como ocurre con CMS, PKCS#7 soporta codificación BER de longitud indefinida y mediante single-pass para los tipos de contenido mejorado, con las mismas ventajas y desventajas que en el caso de CMS.

Sin embargo, los tipos de contenido de datos firmados, de datos firmados y envueltos, y de datos procesados, necesitan codificación DER. Por ello un segundo barrido de los datos puede ser necesario cuando el contenido "interior" de uno de esos tipos de contenido no es de tipo de contenido de datos.

2.5.3 XML Signature, XML-DSig

La firma XML, XML Signature (también llamada XML-DSig) es una recomendación del World Wide Web Consortium, W3C, que define una sintaxis SML para firmas digitales [1]. Funcionalmente tiene mucho en común con PKCS#7 pero es más extensible y está orientada hacia la firma de documentos XML. Este formato de

firma es ampliamente usado por varias tecnologías web como SOAP y SAML entre otros.

Las firmas XML pueden usarse para firmar datos (recursos de cualquier tipo), aunque típicamente se trata de documentos XML. Sin embargo cualquier recurso que sea accesible a través de una dirección URL puede ser firmado.

Si la firma se realiza sobre un recurso localizado fuera del documento que la contiene se denomina firma separada o *detached*; si por el contrario la firma se refiere a algún elemento dentro de su mismo documento se denomina firma envuelta o *enveloped*; finalmente si la misma firma contiene los datos que se han firmado se denomina firma envolvente o *enveloping.*

Las firmas de tipo XML-DSig consisten de un elemento *Signature* expresado según el espacio de nombres definido por el W3C [1], y que consta de los siguientes elementos:

```
<Signature>
  <SignedInfo>
    <SignatureMethod />
    <CanonicalizationMethod />
    <Reference>
       <Transforms>
       <DigestMethod>
       <DigestValue>
    </Reference>
    <Reference /> etc.
  </SignedInfo>
  <SignatureValue />
  <KeyInfo />
  <Object />
</Signature>
```

Figura 2-3: Elemento *Signature* de *XML-DSig*

En el capítulo 4.5.1, figura 4-43 se muestra un ejemplo de archivo firmado mediante la aplicación desarrollada en el ámbito de este proyecto. Dicho ejemplo ilustra bien el uso de los elementos mostrados en la figura 2-3, ya que el formato de firma mostrado en él es XAdES-BES, una extensión de XML-DSig. A continuación se muestra una breve explicación de la información contenida en cada uno de los distintos elementos de la estructura XML de la figura 2-3:

- *SignedInfo*: el elemento *SignedInfo* contiene o referencia los datos firmados y especifica qué algoritmos se han utilizado para realizar la firma digital
- *SignatureMethod*: el elemento *SignatureMethod* es utilizado por el elemento *SignatureValue*. Se incluye en *SignedInfo* para prevenir manipulaciones, ya que al estar incluido en ese elemento, será firmado y cualquier alteración sería detectable.
- *CanonicalizationMethod*: al igual que el elemento *SignatureMethod*, este elemento es utilizado por el elemento *SignatureValue.* Por el mismo motivo que aquél se incluye en *SignedInfo*.

- *Reference*: uno o más elementos de este tipo especifican el/los recurso/s que se va/n a firmar mediante referencia URI.
- *Transforms*: elemento que indica que transformaciones deben ser aplicadas sobre el recurso indicado en el elemento *Reference* que contiene a este elemento antes de proceder a la firma.
- *DigestMethod*: este elemento especifica el algoritmo de hash que se va a utilizar.
- *DigestValue*: este elemento contiene el resultado de aplicar el algoritmo de hash especificado en el elemento *DigestMethod* sobre el recurso, una vez transformado según el elemento *Transforms*.
- *SignatureValue*: este elemento contiene el resultado de aplicar los parámetros de firma especificados en el elemento *SignatureElement* sobre el elemento *SignedInfo* tras aplicar el algoritmo especificado por el elemento *CanonicalizationMethod*. Este resultado de la firma está codificado en Base64.
- *KeyInfo*: este elemento es opcional y permite que el firmante incluya información para validar la firma. Generalmente esto se consigue incluyendo uno o más certificados digitales X.509.
- *Object*: este elemento también es opcional y contiene los datos firmados en caso de tratarse de una firma envolvente (*enveloping*).

A la hora de validar una firma de tipo XML-DSig se sigue un proceso denominado *Core Validation*. En primer lugar se realiza la validación de las referencias, o *Reference Validation*. En este paso se verifica el resumen o *digest* de cada referencia recuperando el correspondiente recurso, aplicándole las transformaciones y el metodo de *digest* necesarios. El resultado se compara con el valor almacenado en el elemento *DigestValue*. Si estos valores no coinciden la validación no se supera.

El segundo paso del proceso de *Core Validation* es el de validación de firma, o *Signature Validation*. El elemento *SignedInfo* se serializa teniendo en cuenta el método de canonización indicado en *CanonicalizationMethod*, se obtienen los datos de la clave utilizada a partir de *KeyInfo* o por otros medios (recordemos que *KeyInfo* es opcional) y la firma se verifica usando el método especificado *SignatureMethod*.

De esta forma, este proceso de validación concluye si los recursos fueron realmente firmados por quien dice haberlos firmado. No obstante, debido a la compleja extensión de los métodos de canonización y de transformación, la parte que verifica debe asegurarse de que lo que se firmó y se procesó mediante un método de *digest,* es realmente lo que estaba presente en los datos originales. Dicho de otra forma, debe comprobar que los algoritmos usados son de confianza y que no cambian el contenido de los datos firmados.

El mencionado paso de canonización es necesario por la siguiente razón. La creación de firmas XML-DSig es más compleja que la creación de firmas digitales ordinarias debido a que un documento XML dado puede tener más de una representación serializada legal. La serialización consiste en un proceso de codificación de un objeto en una serie de bits que permiten su fácil almacenamiento o transmisión. La serie resultante de bits se debe poder reordenar de acuerdo con el formato de

serialización para crear un clon semánticamente idéntico del objeto serializado original.

Por ejemplo, un espacio en blanco dentro de un elemento XML no tiene ninguna significancia sintáctica, de forma que <*Elemento*> es sintácticamente idéntico a <*Elemento* >, que incluye un espacio en blanco antes del carácter "**>**". Como la firma digital se crea usando un algoritmo de clave asimétrica para codificar el resultado de hacer pasar el documento XML serializado por una función de hash, un único byte de diferencia haría que la firma digital variara.

Para evitar este problema y garantizar que documentos XML idénticos en sentido lógico produzcan firmas digitales idénticas, se suele emplear una transformación de canonización XML. El término canonización, una traducción aproximada de *Canonicalization*, a menudo se representa como *C14n*, donde *C* y *n* son la primera y última letras de *Canonicalization* y 14 el número de letras entre esos dos caracteres. Estas transformaciones garantizan que documentos lógicamente idénticos produzcan representaciones serializadas exactamente idénticas.

Surge otra complicación no contemplada hasta ahora debido a la manera en que el algoritmo de *C14n* trata por defecto declaraciones de espacios de nombres: es frecuente que un documento XML firmado deba ser incluido en otro documento. En este caso el algoritmo original de canonización no dará el mismo resultado que daría de ser aplicado al documento por sí solo. Por esta razón se creó lo que se conoce como *Exclusive Canonicalization*, que serializa las declaraciones de espacios de nombres, *namespaces*, de forma independiente del código XML que las rodea.

2.5.4 XAdES

XAdES son las siglas en inglés de *XML Advanced Electronic Signatures* (firma electrónica avanzada XML). Se trata de un conjunto de extensiones a las recomendaciones XML-DSig, haciéndolas adecuadas para la firma electrónica avanzada.

XAdES extiende el modelo de XML-DSig de forma que especifica perfiles precisos de XML-DSig para ser usados con firma electrónica, reconocida con el sentido de la directiva 1999/93/EC de la Unión Europea [2]. Estos perfiles de uso definidos en la especificación XAdES se explicarán más adelante, entrando en detalle en el perfil utilizado en la aplicación desarrollada en el ámbito de este proyecto.

Un gran ventaja de XAdES es que los documentos firmados electrónicamente pueden seguir siendo válidos durante largos periodos de tiempo, incluso aunque los algoritmos criptográficos de firma hayan sido vulnerados.

Las propiedades de firma digital del nuevo conjunto que define la especificación de XAdES pueden ser utilizadas en diferentes combinaciones para satisfacer distintos requisitos en función del tipo de firma XAdES que se desee obtener. A continuación se muestra un breve listado de estas nuevas propiedades:

- *SignaturePolicyIdentifier*: esta propiedad contiene información para la identificación inequívoca de la política de firma bajo la cual se ha realizado la firma digital. Esto garantiza que la parte verificadora será capaz de usar la misma política de firma durante el proceso de verificación. Las políticas de firma son útiles para clarificar de forma precisa qué rol y qué compromiso asume el firmante respecto al objeto de datos firmado y para evitar alegaciones por parte del verificador de que el firmante asumió una política de firmado diferente

- Propiedades de validación de datos. Son un determinado número de tipos XML capaces de contener tanto datos de validación (cadenas de certificados, CRLs, respuestas OCSP, etc.) como referencias a ellos (identificadores de certificados, CRLs, respuestas OCSP, etc.). Las propiedades de estos tipos permiten incorporar todo el material utilizado para la validación dentro de la firma. Además se pueden usar de forma conjunta con propiedades de sellado de tiempo para proporcionar validez de largo plazo. Estas propiedades son las siguientes:
  - *CompleteCertificateRefs*: contiene referencias a los certificados de la Autoridad de Certificación, CA, usados para validar la firma.
  - *CompleteRevocationRefs*: contiene referencias a todo el conjunto de información de revocación usado para la verificación de la firma digital.
  - *AttributeCertficateRefs*: contiene referencias a todo el conjunto de certificados de *Attribute Authorities* usados para validar el certificado de atributos.
  - *AttributeRevocationRefs*: contiene referencias a todo el conjunto de referencias a los datos de revocación que han sido usados en la validación de los atributos certificado presentes en la firma.
  - *CertificateValues*: contiene el valor de los certificados usados para validar la firma.
  - *RevocationValues*: contiene todo el conjunto de información de revocación usada para la verificación de la firma digital.
  - *AttrAuthoritiesCertValues*: contiene los valores de los certificados de las *Attribute Authorities* que han sido usados para validar el certificado atributo cuando éste está presente en la firma.
  - *AttributeRevocationValues*: contiene el conjunto de datos de revocación que ha sido usado para validar el atributo certificado si está presente en la firma.

- Propiedades de *token* de sellado de tiempo. La especificación XAdES define dos tipos XML concretos y uno abstracto para permitir la inclusión de *tokens* de sellado de tiempo en las firmas XML-DSig. Los tipos concretos son *XAdESTimeStampType* y *OtherTimeStampType* y el tipo abstracto es el *GenericTimeStampType*.

- Otras propiedades: propiedades que no se enmarcan en ninguno de los otros dos tipos anteriores pero que pueden ser útiles en un amplio rango de entornos. Concretamente:

- *SigningCertificate*: esta propiedad contiene una referencia inequívoca al certificado del firmante, formada por su identificador y el valor del *digest* del certificado. Su uso es particularmente importante cuando un firmante posee varios certificados distintos con la misma clave pública para evitar alegaciones por parte de un verificador de que la firma implica otro certificado con distinto contenido. También es importante cuando el firmante tiene distintos certificados con distintas claves públicas, para proveer al verificador de información correcta para la verificación. Finalmente, también es importante en el caso de que la clave de expedición de la CA que provee el certificado estuviera comprometida.
- *SigningTime*: esta propiedad contiene el instante de tiempo en que el firmante afirma haber realizado el proceso de firma.
- *DataObjectFormat*: esta propiedad identifica el formato de un objeto de datos firmado (cuando las firmas electrónicas no son intercambiadas en un contexto restringido) para posibilitar su presentación o uso al verificador (texto, sonido, video) en el modo en que pretendía el firmante.
- *CommitmentTypeIndication*: esta propiedad identifica el compromiso asumido por el firmante al firmar objeto/s de datos en el contexto de la política de firma seleccionada (cuando se está usando de forma explícita un determinado compromiso). Es necesario cuando una política de firma especifica más de un único tipo de compromiso, ya que cada uno de los cuales podría tener diferentes interpretaciones legales acerca del propósito de la firma (por ejemplo prueba de origen, prueba de recibo, prueba de creación, etc.).
- *SignatureProductionPlace*: esta propiedad contiene la indicación del lugar donde el firmante afirma que se ha producido la firma.
- *SignerRole*: esta propiedad contiene los roles (certificados o simplemente reivindicados) asumidos por el firmante al crear la firma.
- *CounterSignature*: esta propiedad contiene la/s firma/s producidas sobre la misma firma.

Estas propiedades pueden ser combinadas para generar distintos perfiles de firma. XAdES define seis perfiles (formas) según el nivel de protección ofrecido, si bien sólo a las cuatro primeras se les requiere conformidad. Cada perfil incluye y extiende al previo. Únicamente se explicará de forma extensa el perfil de firma XAdES-BES, ya que es el empleado en la aplicación desarrollada en este proyecto:

- XAdES-BES (*Basic Electronic Signature*), forma básica que simplemente cumple los requisitos legales de la directiva de la Unión Europea para firma electrónica avanzada,
- XAdES-T (*Timestamp*), añade un campo de sellado de tiempo para proteger contra el repudio,
- XAdES-C (*Complete*), añade referencias a datos de verificación (certificados y listas de revocación) a los documentos firmados para permitir verificación y validación off-line en el futuro (pero no almacena los datos en sí mismos),

- XAdES-X (*eXtended*), añade sellos de tiempo a las referencias introducidas por XAdES-C para evitar que pueda verse comprometida en el futuro una cadena de certificados,
- XAdES-X-L (*eXtended Long-term*), añade los propios certificados y listas de revocación a los documentos firmados para permitir la verificación en el futuro incluso si las fuentes originales (de consulta de certificados o de las listas de revocación) no estuvieran ya disponibles,
- XAdES-A (*Archival*), añade la posibilidad de sellado de tiempo periódico (por ejemplo cada año) de documentos archivados para prevenir que puedan ser comprometidos debido a la debilidad de la firma durante un periodo largo de almacenamiento.

2.5.4.1 Basic Electronic Signature, XAdES-BES

XAdES-BES o *XAdES Basic Electronic Signature* es la forma más básica de firma digital aceptada legalmente según la directiva europea de firma electrónica. Es el mínimo formato de firma generable por un firmante que quiera crear una firma reconocida legalmente. Está construida sobre la estructura de XML-DSig incorporando propiedades calificadoras (*qualifying properties*) recogidas en un objeto definido como *ds:Object*. Provee un servicio básico de autenticación y protección de integridad.

Este tipo de firma por sí sólo no provee de suficiente información para poder ser verificada a largo plazo. Por ejemplo, la información de revocación relevante expedida por el emisor de información de estado de certificados, debe estar disponible a largo plazo si se quiere validar una vez transcurrido un largo periodo de tiempo desde la firma, dado que esta información no está incluida en la propia firma XAdES-BES y se debe poder recuperar de alguna manera.

Algunas de las propiedades que añade este perfil de firma están cubiertas por la firma (*signed properties*) y otras no (*unsigned properties*). Es por ello que se deben cumplir determinados requisitos mínimos para garantizar la validez del perfil como firma digital válida.

En una firma XAdES-BES el valor de la firma puede ser calculado de la manera típica de XML-DSig a partir de los objetos de datos que se quieren firmar y a partir del conjunto completo de propiedades (elemento *SignedProperties*) en caso de estar presente.

En este tipo de firma es obligatorio proteger el certificado de firmado con la firma de alguna de las dos siguientes maneras:

- Incorporando la propiedad *SigningCertificate*
- No incorporando la propiedad *SigningCertificate* pero incluyendo el certificado de firma dentro del elemento *ds:KeyInfo* y firmándolo.

En consecuencia, una firma XAdES-BES debe contener al menos uno de los siguientes elementos con el contenido especificado:

- La propiedad firmada *SigningCertificate*: esta propiedad debe contener la referencia y el valor del *digest* del certificado de firma. Puede contener referencias y valores del *digest* de otros certificados (que pueden formar una cadena hasta el punto de confianza), pero no es necesario. En el caso de que existan ambigüedades para identificar el certificado del firmante, la propiedad *SigningCertificate* debe estar presente.

- El elemento *ds:KeyInfo*: las siguientes restricciones sólo se aplicarán a este elemento en caso de que *SigningCertificate* no esté presente en la firma:
  - El elemento *ds:KeyInfo* debe incluir un elemento *ds:X509Data* que contenga el certificado de firma.
  - El elemento *ds:KeyInfo* puede contener otros certificados que formen una cadena que puede alcanzar el punto de confianza
  - El elemento *ds:SignedInfo* debe contener un elemento *ds:Reference* que haga referencia a *ds:KeyInfo* para que este último esté incluido en el cálculo del valor de la firma. De esta forma, el certificado de firma está protegido por la firma.

Al incorporar uno de estos elementos XAdES-BES previene la simple sustitución del certificado del firmante.

Una firma XAdES-BES puede también contener las siguientes propiedades:

- La propiedad firmada *SigningTime*
- La propiedad firmada *DataObjectFormat*
- La propiedad firmada *CommitmentTypeIndication*
- La propiedad firmada *SignerRole*
- La propiedad firmada *SignatureProductionPlace*
- Una o más propiedades firmadas *IndividualDataObjectsTimeStamp* o *AllDataObjectTimeStamp*
- Una o más propiedades no firmadas de tipo *CounterSignature*.

A continuación se muestra un ejemplo de la estructura que tiene una firma de tipo XAdES-BES obtenida incorporando directamente la información calificativa en los nuevos elementos XML al esquema de XML-DSig.

En el esquema, "*?*" indica cero o una ocurrencia; "*+*" denota una o más ocurrencias y "*" denota cero o más ocurrencias. La definición del esquema XML para este tipo de firma define el prefijo "*ds*" para todos los elementos XML ya definidos en la especificación XML-DSig y establece que el espacio de nombres por defecto es el definido para el presente documento. En consecuencia, en este ejemplo los elementos ya definidos en XML-DSig aparecen con el prefijo "*ds*", mientras que los nuevos elementos XML definidos en el presente documento aparecen sin prefijo:

```
<ds:Signature ID?>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod/>
    <ds:SignatureMethod/>
    (<ds:Reference URI? >
      (<ds:Transforms>)?
      <ds:DigestMethod/>
      <ds:DigestValue/>
    </ds:Reference>)+
  </ds:SignedInfo>
  <ds:SignatureValue/>
  (<ds:KeyInfo>)?

  <ds:Object>

    <QualifyingProperties>

      <SignedProperties>

        <SignedSignatureProperties>
          (SigningTime)?
          (SigningCertificate)?
          (SignatureProductionPlace)?
          (SignerRole)?
        </SignedSignatureProperties>

        <SignedDataObjectProperties>
          (DataObjectFormat)*
          (CommitmentTypeIndication)*
          (AllDataObjectsTimeStamp)*
          (IndividualDataObjectsTimeStamp)*
        </SignedDataObjectProperties>

      </SignedProperties>

      <UnsignedProperties>

        <UnsignedSignatureProperties>
          (CounterSignature)*
        </UnsignedSignatureProperties>

      </UnsignedProperties>

    </QualifyingProperties>

  </ds:Object>

</ds:Signature>
```

Figura 2-4: Ejemplo de firma tipo XAdES-BES

3 Tarjeta Inteligente de Identificación

3.1 Tarjetas inteligentes

En la actualidad, el uso de este tipo de cifrados con clave pública se está extendiendo más allá de las limitaciones propias del almacenamiento en memoria de los certificados de usuario gracias a su combinación con las tarjetas inteligentes o *Smart Cards*. Una tarjeta inteligente es cualquier tarjeta con circuitos integrados incluidos que permiten la ejecución de cierta lógica programada.

La percepción estándar de una tarjeta inteligente es una tarjeta con un pequeño microprocesador de las dimensiones de una tarjeta de crédito (o más pequeña, como por ejemplo, tarjetas SIM de telefonía) con varias propiedades especiales (por ejemplo: procesador criptográfico seguro, sistema de archivos seguro, características legibles a simple vista). Es capaz de proveer servicios de seguridad (ej. confidencialidad de la información en la memoria).

Los tamaños de estas tarjetas vienen definidos por los estándares de la ISO/IEC 7810: los tamaños ID-1 para el caso de las tarjetas mayores (tamaño tarjeta de crédito, 85.60×53.98mm) e ID-000 para el de las menores (tipo SIM, 25x15mm) son los más comunes. Ambos tamaños presentan un grosor de 0.76mm.

Según las capacidades de su chip, las tarjetas más habituales son:

- Tarjetas de memoria: tarjetas que únicamente son un contenedor de ficheros pero que no albergan aplicaciones ejecutables. Se usan generalmente en

aplicaciones de identificación y control de acceso sin altos requisitos de seguridad.

- Tarjetas microprocesadas: tarjetas con una estructura análoga a la de un ordenador (procesador, memoria volátil, memoria persistente). Albergan ficheros y aplicaciones y suelen usarse para identificación y pago con monederos electrónicos.
- Tarjetas criptográficas: tarjetas microprocesadas avanzadas que poseen módulos hardware para la ejecución de algoritmos usados en cifrados y firmas digitales. En estas tarjetas se pueden almacenar de forma segura un certificado digital (y su clave privada) y firmar documentos o autenticarse con la tarjeta sin que el certificado salga de la tarjeta (sin que se instale en el almacén de certificados de un navegador web, por ejemplo) ya que es el procesador de la propia tarjeta el que realiza la firma. Ejemplos claros de este tipo de tarjeta son las emitidas por la Fábrica Nacional de Moneda y Timbre (FNMT) y el recientemente introducido DNI electrónico, o DNIe. Este tipo de tarjetas son las que permiten trabajar con sistemas de validación de cifrados de clave pública.

También se pueden clasificar por la manera en que se establece comunicación con el lector de tarjetas:

- Tarjetas de contacto: estas tarjetas tienen un área de contacto compuesta por varios contactos menores bañados en oro. El área de contacto tiene un tamaño de aproximadamente 1cm cuadrado. Cuando una tarjeta de este tipo se inserta en el lector, el chip hace contacto con conectores eléctricos que pueden leer información de éste y escribir información en él. Los estándares ISO/IEC 7816 e ISO/7810 definen la forma física de estas tarjetas inteligentes, las posiciones y formas de los conectores eléctricos, las características eléctricas, los protocolos de comunicación (esto incluye el formato de los comandos enviados a la tarjeta y las respuestas devueltas por ésta), la robustez necesaria de la tarjeta y su funcionalidad. Las tarjetas no incluyen ningún dispositivo de almacenamiento de energía para su funcionamiento, como baterías. La energía necesaria debe ser suministrada por el lector de tarjetas.
- Tarjetas sin contacto: en este tipo de tarjetas el chip se comunica con el lector mediante tecnología de inducción RF con tasas de datos entre 106 y 848 Kb/s. Estas tarjetas tan sólo necesitan estar próximas a una antena lectora para completar una transacción. Su uso es habitual en situaciones en las que las transacciones deben ser procesadas rápidamente o incluso sin manos, como por ejemplo en medios de transporte públicos, donde este tipo de tarjetas pueden ser usadas sin ser sacadas de la cartera o monedero. El estándar para tarjetas inteligentes sin contacto, ISO/IEC 14443, data de 2001, y define dos tipos de tarjetas inteligentes sin contacto (tipos “A” y “B”). La distancia máxima de comunicación llega hasta los 10cm entre el lector y la tarjeta. Varias propuestas para añadir al estándar ISO/IEC 14443 cuatro nuevos tipos (“C”, “D”, “E” y “F”) han sido rechazadas por la ISO. De manera alternativa se puede seguir el estándar ISO 15693 para tarjetas inteligentes sin contacto, que permite comunicaciones hasta distancias de

50cm. Al igual que las tarjetas de contacto, este tipo de tarjeta no incluye batería. Utilizan un inductor para capturar parte de la señal de radiofrecuencia de interrogación del lector, rectificarla y usarla para alimentar los componentes electrónicos de la tarjeta.

La tecnología RFID, *Radio Frequency Identification*, está altamente ligada a este último tipo de tarjetas. Sin embargo, no son equivalentes. Aunque RFID se pueda usar para aplicaciones similares a las de tarjetas inteligentes sin contacto, los dispositivos RFID generalmente no incluyen memoria manipulable o capacidad de proceso.

Los protocolos de comunicación con las tarjetas inteligentes son los siguientes:

- T=0: protocolo de transmisión asíncrona half-duplex a nivel de byte, definido en el ISO/IEC 7816-3. Para tarjetas inteligentes de contacto.
- T=1: protocolo de transmisión asíncrona hlaf-duplex a nivel de bloque, definido en el ISO/IEC 7816-3. Para tarjetas inteligentes de contacto.
- T=CL: transmisión de unidad de datos de protocolo de aplicación (*Application Protocol Data Unit*, APDU) vía interfaz sin contacto. Definida en el ISO/IEC 14443-4. Para tarjetas sin contacto.
- T=2 y T=4: reservados para futuras operaciones full-duplex de tarjetas de contacto.
- T=14: indica que la tarjeta inteligente utiliza un protocolo no estándar/propietario.

3.1.1 Trabajo con Tarjetas Inteligentes: API del Estándar PKCS#11

Para acceder a los datos de estas tarjetas inteligentes criptográficas el estándar PKCS#11 y el API que define están ampliamente adoptados. PKCS#11 es un estándar de la familia Public-Key Cryptography Standards PKCS. Define un API independiente de plataforma para interactuar con módulos criptográficos, tales como módulos de seguridad por hardware y tarjetas inteligentes.

Al no existir un estándar real para módulos criptográficos, este API se ha desarrollado para servir de capa de abstracción para módulos criptográficos genéricos. El API de PKCS#11 define los tipos de objeto criptográfico más comúnmente usados, tales como claves RSA, certificados X.509, claves DES, etc., y todas las funciones necesarias para usar, generar, modificar y borrar esos objetos.

La mayoría del software existente para interacción con autoridades de certificación utiliza PKCS#11 para acceder a la clave de firmado de la autoridad certificadora o para inscribir certificados de usuario. Software multiplataforma, como por ejemplo el popular navegador Mozilla Firefox o las librerías OpenSSL, utilizan PKCS#11 para trabajar con tarjetas inteligentes.

3.1.2 Trabajo con Tarjetas Inteligentes: CryptoAPI y CSP de Microsoft

CryptoAPI es un interfaz de programación de aplicaciones incluido con los sistemas operativos Microsoft Windows que provee de servicios para desarrolladores que deseen hacer aplicaciones seguras para Windows basándose en métodos criptográficos. Se compone de un conjunto de librerías enlazadas dinámicamente que proveen una capa de abstracción que aísla al programador de la parte del código que realiza las tareas criptográficas. Este API soporta tanto criptografía simétrica como asimétrica, e incluye funcionalidad tanto para codificar y decodificar datos como para autenticación mediante certificados digitales.

CryptoAPI funciona con varios proveedores de servicios criptográficos CSPs (*Cryptographic Service Provider*) instalados en la máquina Windows en cuestión. Los CSPs son los módulos que realizan el trabajo real de codificación/decodificación de datos operando con funciones criptográficas.

La última generación de este API se integra fácilmente con el susbsistema de tarjetas inteligentes, ya que incorpora un nuevo módulo, el *Base CSP*, que encapsula el API para tarjetas inteligentes. De esta forma los proveedores de tarjetas inteligentes tan sólo deben hacer sus productos y equipos compatibles con éste módulo, en vez de proveer una solución desde cero.

3.2 Descripción de la Tarjeta Inteligente de Identificación

La Tarjeta Inteligente de Identificación TII usada es una tarjeta de policarbonato, que incorpora un chip con información digital. Su tamaño, por tanto, coincide con las dimensiones de las tarjetas de crédito comúnmente utilizadas (85,60mm de ancho x 53,98mm de alto, tamaño ID-1 según ISO/IEC 7810).

Figura 3-1: TII de contacto genérica acorde con la norma ISO/IEC 7810, aún sin las características exteriores impresas/grabadas

Las claves privadas del titular de la TII se encuentran almacenadas en el procesador de la tarjeta criptográfica. Con esto se consigue que las claves privadas no abandonen nunca el soporte físico de la TII, minimizando las posibilidades de poner en riesgo dichas claves.

Para el acceso a las claves y al certificado de firma el titular deberá emplear una clave personal de acceso (PIN) generada en el momento de expedir la TII y que sólo él debe conocer. En todo momento el titular podrá modificar la clave personal de acceso en un terminal de desbloqueo y actualización de la autoridad emisora de las TII mediante el siguiente procedimiento:

- Si conoce la clave personal de acceso - PIN - podrá emplearlo durante el proceso de cambio
- En caso de no recordar la clave personal de acceso – PIN - (o encontrarse bloqueada la tarjeta al superar el número de tres intentos con un PIN incorrecto) podrá realizar el cambio mediante los métodos que la entidad emisora haya incluido en su infraestructura.

En ningún caso el olvido de la clave personal de acceso supondrá la revocación de los certificados almacenados en la TII, ya que siempre podrá ser modificada por el procedimiento anterior.

Los certificados presentes en la tarjeta usada son tres. En primer lugar tenemos el Certificado de Componente. Su propósito es la autenticación de la tarjeta mediante el protocolo de autenticación mutua definido en CWA 14890. Permite el establecimiento de un canal cifrado y autenticado entre la tarjeta y los drivers instalados en el ordenador. Este certificado no estará accesible directamente por los interfaces estándar (PKCS11 o CSP).

Además de este certificado, la TII posee los siguientes certificados de usuario: certificados X.509v3 de ciudadano (autenticación y firma) y claves privadas asociadas, que se generarán e insertarán durante el proceso de expedición de la TII.

El certificado de autenticación del titular tiene como finalidad garantizar electrónicamente la identidad del titular al realizar una transacción telemática. El certificado de autenticación (*Digital Signature*) asegura que la comunicación electrónica se realiza con la persona que dice que es. El titular podrá, a través de su certificado, acreditar su identidad frente a cualquiera, ya que se encuentra en posesión del certificado de identidad y de la clave privada asociada al mismo. El uso de este certificado no está habilitado en operaciones que requieran no repudio de origen, por tanto los terceros aceptantes y los prestadores de servicios no tendrán garantía del compromiso del titular de la TII con el contenido firmado. Su uso principal será para generar mensajes de autenticación (confirmación de la identidad) y de acceso seguro a sistemas informáticos (mediante establecimiento de canales privados y confidenciales con los prestadores de servicio). Este certificado puede ser utilizado también como medio de identificación para la realización de un registro que permita la expedición de certificados reconocidos por parte de entidades privadas, sin verse estas obligadas a realizar una fuerte inversión en el despliegue y mantenimiento de una infraestructura de registro.

Por último, el certificado de firma del titular de la TII. Este certificado es el que se debe utilizar para la firma de documentos, garantizando la integridad del documento

firmado y el no repudio de origen. Es un certificado X509v3 estándar, que tiene activo en el *Key Usage* el bit de *ContentCommitment* (No Repudio) y que está asociado a un par de claves pública y privada, generadas en el interior del chip de la TII.

La validez de la TII la fija la entidad emisora, así como el periodo de validez de los certificados incluidos en la TII.

Por tratarse de una tarjeta inteligente de contacto, el lector usado para trabajar con la TII debe cumplir el estándar ISO/IEC 7816 (1, 2 y 3), soportar tarjetas asíncronas basadas en protocolos T=0 (y T=1) a velocidades de comunicación mínimas de de 9.600 bps y también debe soportar los estándares API PC/SC (*Personal Computer/Smart Card*), CSP de Microsoft y el API PKCS#11.

3.3 PKI de la TII

3.3.1 Autoridad de Aprobación de Políticas PAA

La Autoridad de Aprobación de Políticas (PAA según sus siglas en inglés, Policy Approval Authority) tiene atribuida la función de elaboración y propuesta de aprobación de la Declaración de Prácticas y Políticas de Certificación, DPC, así como de sus modificaciones. Dicha DPC será aprobada por el organismo correspondiente.

Asimismo, la PAA es la responsable, en caso de que se tuviese que evaluar la posibilidad de que una CA externa interactúe con la PKI de la TII, de determinar la adecuación de la DPC de dicha CA a la DPC existente y de regular la prestación del servicio de validación por parte de terceros.

La PAA es también la encargada de analizar los informes de las auditorías, totales o parciales, que se hagan sobre la TII, así como de determinar, en caso necesario, las acciones correctoras a ejecutar.

3.3.2 Autoridades de Certificación CA

La CA relaciona dos pares de claves con el titular de una TII concreta a través de la emisión de sendos certificados de conformidad con los términos de la DPC establecida.

Las CA que componen la PKI de la TII son:

- CA Raíz: Autoridad de certificación de primer nivel. Esta CA sólo emite certificados para sí misma y sus CA Subordinadas. Únicamente estará en funcionamiento durante la realización de las operaciones para las que se establece.
- CA Subordinadas: Autoridades de certificación subordinadas de CA Raíz. Su función es la emisión de certificados para los titulares de la TII.

En el momento de publicación de la DPC vigente al escribir este documento, el dominio de certificación de la TII consta de tres CA subordinadas.

Todas estas autoridades de certificación presentan dos certificados, uno con algoritmo de firma *pkcs1-sha1WithRSAEncryption*, y otro con algoritmo de firma *pkcs1-sha256WithRSAEncryption*. El primero de ellos se publica por razones de interoperabilidad, para facilitar a aquellos sistemas y aplicaciones que no soporten *pkcs1-sha256WithRSAEncryption*, y construir la cadena de confianza en los procesos de validación de certificados y firma. En un futuro la DPC se revisará para indicar de forma expresa que dicho certificado deja de tener efecto.

La incorporación de una nueva CA al dominio o el cese de operación de la misma serán causa de modificación de la DPC vigente y de notificación a través de los mecanismos habilitados a tal efecto.

3.3.3 Autoridades de Registro RA

La RA está constituida por todas las oficinas de expedición de la TII. Tienen por misión realizar las funciones de asistencia a la CA en los procedimientos y trámites relacionados con los titulares para su identificación, registro y autenticación. De esta forma se garantiza la asignación de las claves al solicitante. La situación física serán las que estipule la CA y las instalaciones habilitadas para los equipos móviles, así como otros lugares que a tal efecto determine el órgano encargado de la expedición y gestión de las TII.

3.3.4 Autoridad de Validación VA

La(s) Autoridad(es) de Validación AV tiene(n) como función la comprobación del estado de los certificados almacenados en la TII, mediante el protocolo OCSP. Se determina el estado actual de un certificado electrónico mediante la solicitud de un Tercero Aceptante sin requerir el acceso a listas de certificados revocados por éstas.

Este servicio de consulta debe prestarse garantizando la disponibilidad de un servicio de consulta sobre la vigencia de los certificados rápido y seguro.

3.3.5 Usuario Suscriptor

Se entiende por usuario suscriptor de la PKI de la TII al titular, mayor de edad y con plena capacidad de obrar, que voluntariamente confía y hace uso de los certificados contenidos en su TII y emitidos por la autoridad emisora competente, de los cuales es titular.

Cuando un usuario decida voluntariamente confiar y hacer uso de alguno de sus

certificados le será de aplicación la presente DPC.

3.3.6 Terceros Aceptantes

Los Terceros Aceptantes son las personas o entidades diferentes del titular de la TII que deciden aceptar y confiar en un certificado obtenido del almacén de certificados de una TII. Se entiende como prestador de servicios telemáticos a toda persona física o jurídica que ofrece al titular la posibilidad de realizar transacciones telemáticas utilizando la TII.

4 Aplicación de Firma y Validación: Uso

Este capítulo explica la funcionalidad de la aplicación desarrollada en el ámbito de este proyecto y presenta varios ejemplos de uso paso a paso con amplia y documentación.

4.1 Requisitos

Los requisitos software para poder utilizar la herramienta de firma y validación de certificados en PKI con TII son los siguientes:

- Sistema operativo Microsoft Windows. La aplicación ha sido desarrollada enteramente sobre Microsoft Windows XP y se ha comprobado su correcto funcionamiento en Microsoft Windows Vista. El funcionamiento sobre otros sistemas operativos de Microsoft no está comprobado, aunque no deberían plantear problemas siempre que cumplan los demás requisitos software.
- Los módulos criptográficos de PKCS#11 y CSP provistos por la PKI existente específicos para la TII utilizada.
- Java Runtime Environment 1.6.0_11 o superior. Se trata de una aplicación desarrollada enteramente en lenguaje Java. Con las versiones anteriores del JRE pueden aparecer problemas en tiempo de ejecución, por lo que se recomienda encarecidamente que sólo se use la versión indicada o posteriores.
- En caso de que la aplicación se despliegue como un applet insertado en una página web se precisa de un navegador. Cualquier navegador con la máquina

virtual de java habilitada es válido (siempre que sea de la versión indicada en el punto anterior). Si la aplicación se está ejecutando de forma local no es necesario este punto.

En cuanto a requisitos hardware, al igual que con cualquier tarjeta inteligente, se necesita un lector compatible. En este caso el lector debe ser de tarjetas de contacto, debe cumplir el estándar ISO/IEC 7816 (1, 2 y 3), soportar tarjetas asíncronas basadas en protocolos T=0 (y T=1) a velocidades de comunicación mínimas de de 9.600 bps y también debe soportar los estándares API PC/SC (Personal Computer/Smart Card), CSP (Cryptographic Service Provider, Microsoft) y el API PKCS#11.

4.2 Funcionalidad

La aplicación desarrollada en el ámbito de este proyecto está claramente dividida en dos funciones: el proceso de firma digital y el proceso de validación de archivos firmados.

4.2.1 Firma Digital Mediante la TII

La aplicación está diseñada para obtener un archivo de salida con extensión *.xml* que respeta el formato de firma XAdES-BES, el mínimo formato reconocido por las directivas europeas con validez de firma digital. En dicho archivo de salida se almacena una copia del archivo o documento sobre el que se ha realizado la firma, almacenado en formato *Base64*, además de toda la información necesaria para poder comprobar posteriormente la validez de la firma y para poder identificar al firmante. El formato concreto de salida de la aplicación se explica en detalle más adelante.

El archivo sobre el que se calcula la firma digital puede ser tanto un archivo presente en el sistema de archivos en el que se ejecuta la aplicación como un archivo de tipo *PDF*, obtenido a partir de un formulario incluido en la aplicación. En caso de tratarse de firma de formulario se incluirá también una firma digital en el documento *PDF* antes de calcular la firma sobre él e incluirlo en el archivo de salida.

4.2.2 Validación de Archivos

La aplicación desarrollada no sólo genera archivos de salida que contienen el archivo firmado y la información necesaria para cumplir el formato XAdES-BES; también debe proveer de un método para realizar la comprobación necesaria que indique si la firma es válida y quién ha sido el firmante.

Tras indicar a la aplicación qué archivo se debe validar y dónde se debe extraer el archivo o documento sobre el que se calculó la firma, se comprobará que el archivo tenga el formato necesario y se analizará la validez de la firma. En caso de que el archivo pase el test de validación se mostrará quién es el firmante y se extraerá el

archivo o documento contenido en el archivo de firma a la ruta especificada. En caso de que la validación falle no se extraerá ningún archivo al sistema de archivos de la máquina en la que se está ejecutando la aplicación.

En este proceso no es necesario tener ninguna TII insertado en el lector de tarjetas, ya que toda la información necesaria para llevar a cabo la validación, tal como la clave pública necesaria y el nombre del firmante, se encuentra contenida en el mismo archivo de firma que se está analizando.

4.3 Ejemplos de Uso

A continuación se muestran todos los pasos necesarios para utilizar la aplicación desarrollada en cada uno de los casos posibles: Firma Digital de Archivo, Firma Digital de Formulario y Validación de Archivo Firmado.

4.3.1 Ejemplo: Firma Digital de Archivo

En este ejemplo se muestran los pasos a seguir para realizar la firma sobre un archivo presente en el sistema de archivos del ordenador en el que se ejecuta la aplicación.

Este ejemplo parte de la pantalla principal de la interfaz gráfica de la aplicación. En ella se debe seleccionar el panel etiquetado como *"Firma de Archivo"*. De esta forma la aplicación tendrá un aspecto similar al mostrado por la figura 4-1, si bien puede diferir ligeramente, ya que la aplicación está programada para obtener el *Look & Feel* del sistema en el que se ejecuta.

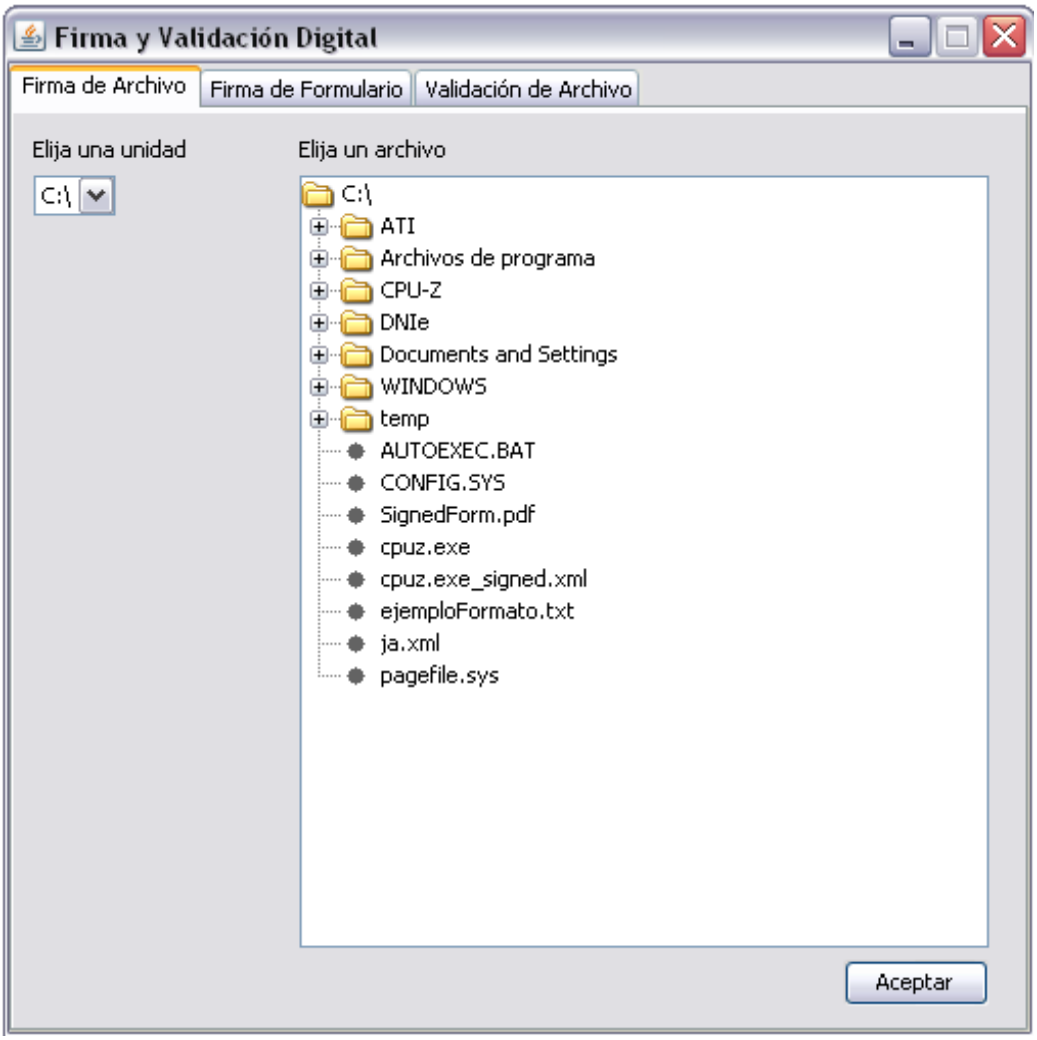

Figura 4-1: Panel de Firma de Archivo

En este panel se muestra a la izquierda un desplegable con todas las unidades válidas en las que se tiene permiso de lectura, y a la derecha se muestra el árbol de archivos de la unidad seleccionada. No se muestran las carpetas ocultas para evitar problemas con la estructura de directorios del sistema en el que se ejecuta la aplicación.

El usuario debe ahora seleccionar un archivo sobre el que desee calcular la firma digital con su TII. Si se pulsa el botón *"Aceptar"* antes de haber seleccionado un archivo válido (debe ser un archivo, no un directorio, y debe tener permiso de lectura) se mostrará el error mostrado en la figura 4-2.

Figura 4-2: Error en panel de Firma de Archivo

Una vez que el usuario ha seleccionado un archivo válido para su firma la interfaz principal de la aplicación queda deshabilitada hasta que se complete el proceso de firma, se aborte por parte del usuario o se produzca un error en el proceso. La interfaz principal queda por tanto tal y como se muestra en la figura 4-3.

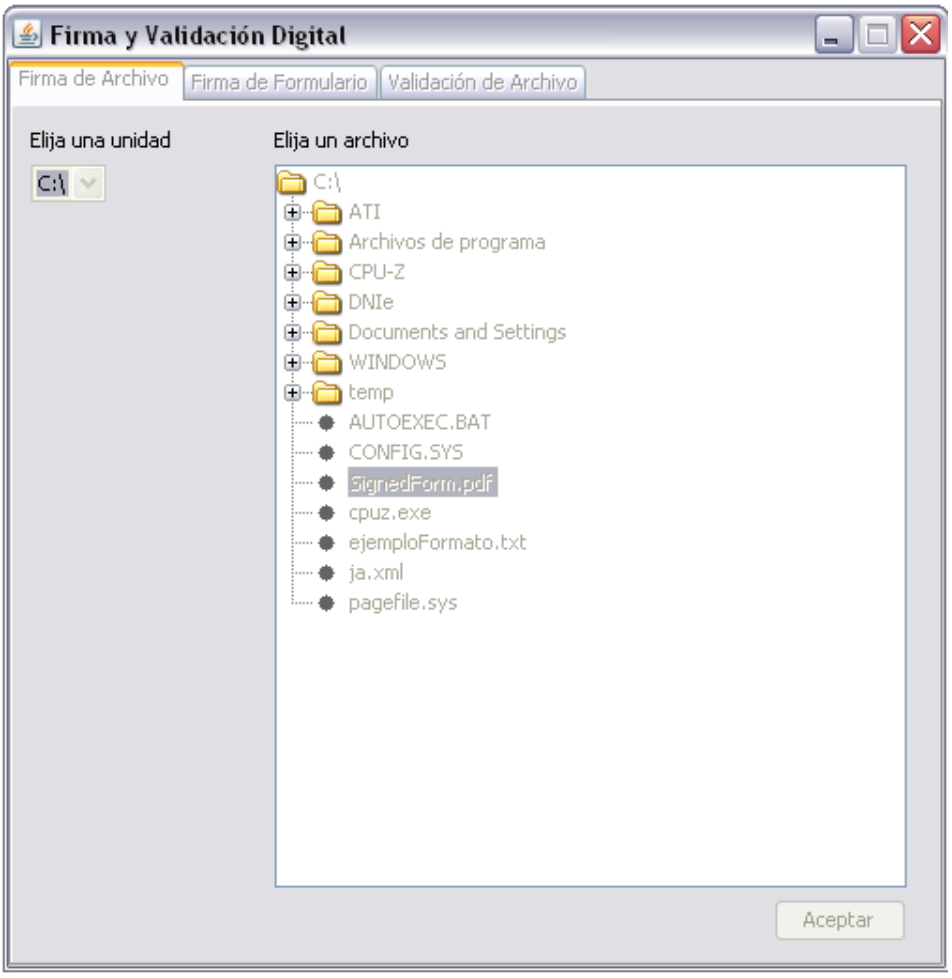

Figura 4-3: Panel de Firma de Archivo deshabilitado

De esta forma el usuario no puede generar comportamientos no deseados mientras se ejecuta el proceso de firma pulsando por error o intencionadamente algún botón de la interfaz gráfica principal, ya sea por pulsar repetidamente el botón *"Aceptar"* o cualquier otro botón de los otros dos paneles.

Tras pulsar el botón de *"Aceptar"* en el panel de *Firma de Archivo* se muestra la siguiente ventana. En ella el usuario puede seleccionar el destino dentro del sistema de archivos donde quiere que se almacene el archivo de salida que contendrá el archivo original y la firma que se ha calculado sobre éste.

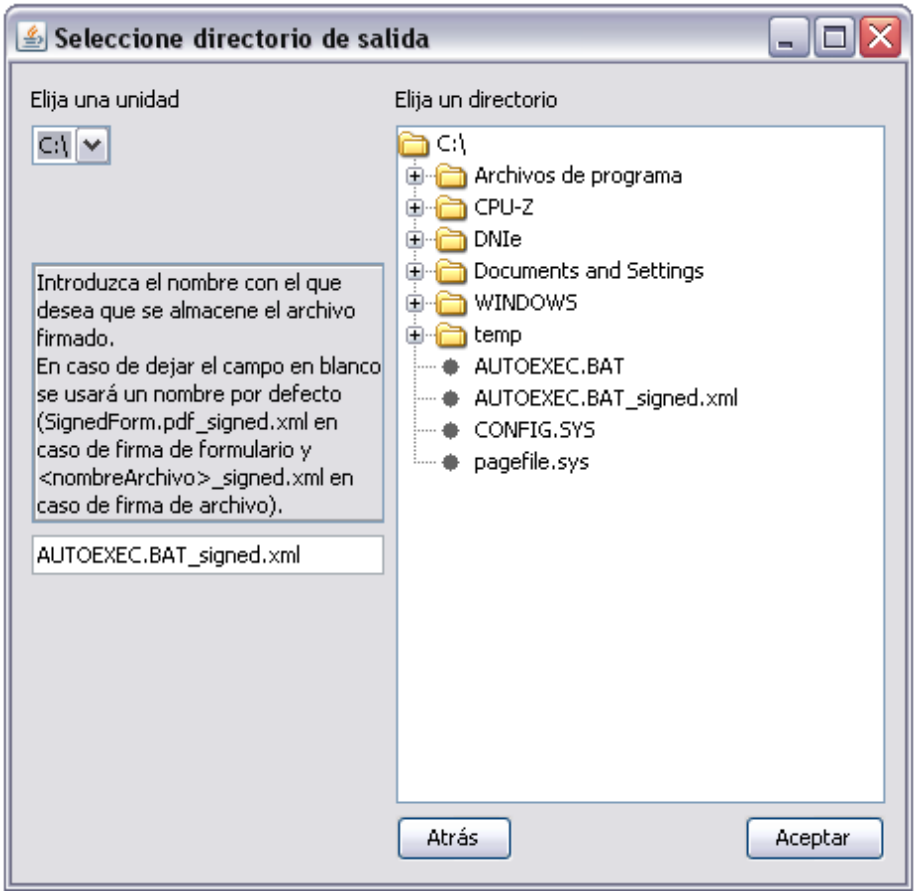

Figura 4-4: Ventana de selección de destino (Firma de Archivo)

Esta ventana se parece mucho al panel del que se parte en el proceso de *Firma de Archivo*, sin embargo esta vez en el desplegable de las unidades del sistema (a la izquierda) se muestran tan sólo las unidades en las que se tiene permiso de escritura y además no están llenas. Posteriormente se comprobará que exista espacio suficiente en la unidad de destino para albergar el archivo de salida generado por la aplicación.

En la figura 4-4 se aprecia también, en la parte izquierda, bajo el campo de texto informativo, el campo donde el usuario puede introducir, si así lo desea, un nombre específico para el archivo de salida. Por defecto la aplicación toma el nombre del archivo sobre el que se calcula la firma y le añade *"_signed.xml"* al final para no tener que introducir un nombre cada vez que se pretenda firmar un archivo. También en caso de dejar este campo en blanco se utilizará el nombre por defecto.

Esta ventana cuenta también con dos botones, el de *"Atrás"* y el de *"Aceptar"*. Pulsando el botón de *"Atrás"* se retrocede a la interfaz principal de la aplicación donde

se puede seleccionar cualquiera de los tres paneles principales (*Firma de Archivo, Firma de Formulario, Validación de Archivo*). Pulsando el botón *"Aceptar"* la aplicación pasa a intentar acceder al almacén de certificados de la TII utilizada, siempre que se haya escogido un directorio de destino válido. En caso de no haber seleccionado ningún directorio válido (un archivo, un directorio sin permiso de escritura o no haber seleccionado nada) se muestra el siguiente error.

Figura 4-5: Error de selección de directorio en ventana de selección de destino (Firma)

También puede darse el caso de que el directorio de destino seleccionado no tenga suficiente espacio para almacenar el archivo de salida que se generaría al procesar el archivo de entrada seleccionado. En este caso el resultado es otro mensaje de error indicando que el fallo se ha dado por problemas de espacio en el sistema de archivos. Pulsando el botón *"Aceptar"* se retrocede nuevamente a la ventana de selección de destino, donde se podrá elegir otro directorio de destino.

Figura 4-6: Error de espacio insuficiente en directorio en ventana de selección de destino (Firma)

Si el directorio de destino seleccionado es válido (es directorio, tiene permiso de escritura y espacio suficiente) se procede a intentar acceder al almacén de certificados de la TII.

En caso de que no se encuentre ninguna TII insertada en el lector aparecerá el mensaje de error mostrado en la figura 4-7, indicándole dicha circunstancia al usuario.

Figura 4-7: Error de acceso a la TII – TII no encontrada

Existe la posibilidad de que el módulo criptográfico suministrado junto con la TII utilizada para trabajar con su PKI, el lector de tarjetas inteligentes o incluso la propia TII den problemas al intentar establecer la comunicación. La corrección de estos fallos está más allá de las posibilidades del programador de aplicaciones que interactúen con la TII, ya que son inherentes al diseño del lector, de la TII y del módulo criptográfico necesario para comunicarse con ella. En estos casos la aplicación mostrará alguno de los mensajes de error de las siguientes figuras, indicando a su vez cual fue el error que provocó el fallo de acceso a la TII.

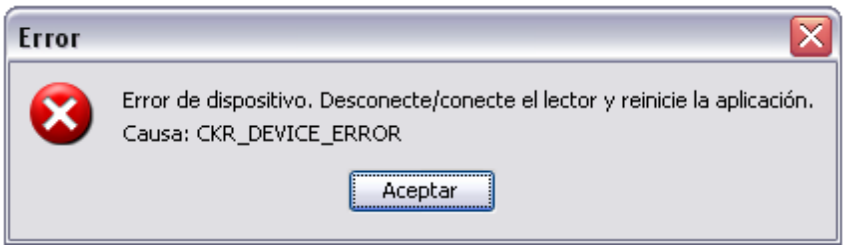

Figura 4-8: Error de acceso a la TII – Error de dispositivo

El error causante del fallo en la aplicación es en este caso un fallo en el módulo criptográfico PKCS#11. El módulo se encuentra en estado de error, generalmente por fallos en un autotest distinto al de consistencia de pares, independiente de la aplicación desarrollada en el ámbito de este proyecto, y hace que todas las funciones de PKCS#11 devuelvan código de error en vez de los resultados esperados.

Figura 4-9: Error de acceso a la TII – Error general

En este caso el error causante del fallo de la aplicación al intentar acceder a la TII es un error general del módulo criptográfico PKCS#11. El módulo ha realizado un autotest de consistencia de pares y no lo ha superado, con lo que ha entrado en estado de error. A partir de ese momento todas las funciones de PKCS#11 devuelven código de error.

En ambos casos, tanto en el caso de *CKR_DEVICE_ERROR* como en el de *CKR_GENERAL_ERROR* es conveniente cerrar la aplicación, extraer la TII y desconectar el lector si es posible antes de intentar de nuevo realizar el proceso de firma.

La normal ejecución de la aplicación debería sin embargo llegar al siguiente punto. En caso de que no se haya accedido aún a la TII habiendo introducido ya el PIN correctamente en esta sesión, se solicita al usuario el PIN de la TII tal y como se muestra en la figura 4-10.

Figura 4-10: Solicitud del PIN de la TII

En la ventana mostrada en la figura 4-10 se ofrecen al usuario un campo de texto donde escribir el PIN de su TII y dos botones, *"Aceptar"* y *"Cancelar"*. El campo de texto donde se debe introducir el PIN oculta mediante caracteres "●" el texto introducido por el usuario de forma que un tercero cercano al monitor del ordenador donde se ejecuta la aplicación no pueda obtener el PIN del usuario. Pulsando el botón *"Cancelar"* se aborta el proceso de firma y la aplicación vuelve a habilitar la interfaz principal. Pulsando el botón *"Aceptar"* se intenta acceder a la TII con el PIN proporcionado, siempre que se haya introducido algún texto. En caso de que el campo del PIN esté vacío la aplicación no hace nada. De esta forma no se intenta acceder a la TII con un PIN que forzosamente sería erróneo, por tener cero caracteres.

En este punto se pueden dar varios casos de error y un único caso válido en que la aplicación continúe el proceso de firma. Los distintos casos de error son aquellos en los que o bien el PIN proporcionado no es correcto o bien el PIN de la TII está bloqueado debido a que se ha intentado acceder tres veces a la TII con un PIN erróneo. En este último caso es imprescindible que el usuario acuda a un terminal de desbloqueo de la autoridad emisora de la TII, ya que este desbloqueo únicamente se puede realizar en las máquinas instaladas a tal efecto.

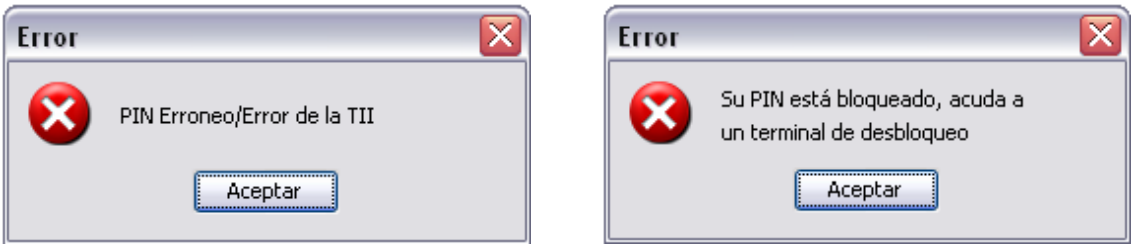

Figura 4-11: Error a) PIN erróneo, b) PIN bloqueado

Si el PIN introducido es el correcto entonces la aplicación accederá al almacén de certificados de la TII sin problemas. En este caso, si se sigue trabajando con la misma TII para firmar otros archivos, no se volverá a solicitar el PIN para acceder a la TII. De esta forma resulta más sencillo firmar distintos archivos en una sola sesión con la aplicación desarrollada.

Al intentar acceder al almacén de certificados puede parecer que la aplicación ha quedado bloqueada o que no responde. Esto se debe a que el proceso de establecer un canal de comunicación con la TII es relativamente lento. En unos instantes se mostrarán por pantalla los certificados almacenados en la TII para que el usuario elija con cual

desea realizar la firma digital. Tal y como se muestra en la figura 4-12, los distintos certificados aparecen en forma de árbol. Cuando se selecciona uno de los certificados de usuario almacenados en la TII, aparece en el recuadro inferior la información de dicho certificado.

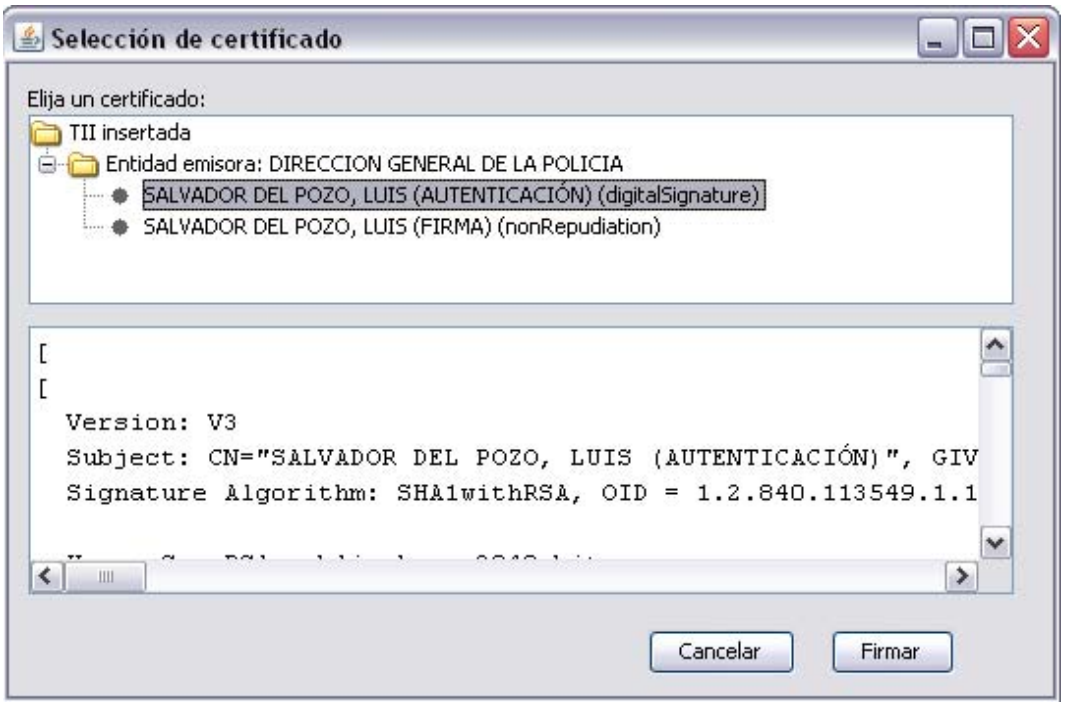

Figura 4-12: Ventana de selección de certificados

En la ventana de *Selección de Certificado* se muestran también dos botones, *"Cancelar"* y *"Firmar"*. Pulsando el botón cancelar se aborta el proceso de firma y se vuelve a activar la interfaz principal de la aplicación. Si se ha seleccionado un certificado, pulsando el botón *"Firmar"* la aplicación creará un nuevo hilo de forma transparente al usuario para realizar la firma con el certificado elegido. En este punto pueden surgir dos problemas, aunque no son comunes.

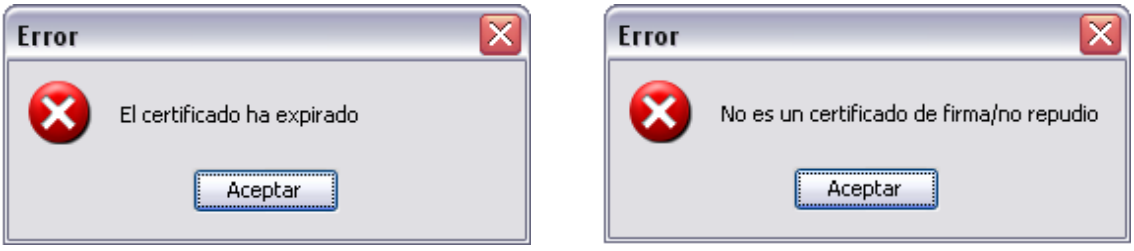

Figura 4-13: Error a) Certificado expirado, b) Certificado no válido

El error representado en la figura 4-13-a aparecerá cuando el certificado almacenado en la TII haya superado su periodo de validez y deba ser renovado en algún terminal de la autoridad emisora. El error representado en la figura 4-13-b es altamente improbable que ocurra y no debería suceder, pero cubre el caso en que el certificado seleccionado almacenado en la TII no esté definido como de firma digital o de no repudio. En ambos casos el proceso de firma se abortará y se retornará al panel de la interfaz principal de la aplicación.

En caso de que el certificado seleccionado por el usuario sea el que provee no repudio del documento firmado se mostrará la advertencia mostrada en la figura 4-14. Dicha advertencia es ajena a la aplicación desarrollada en el ámbito de este proyecto; es lanzada por el módulo criptográfico suministrado por la autoridad emisora de la TII.

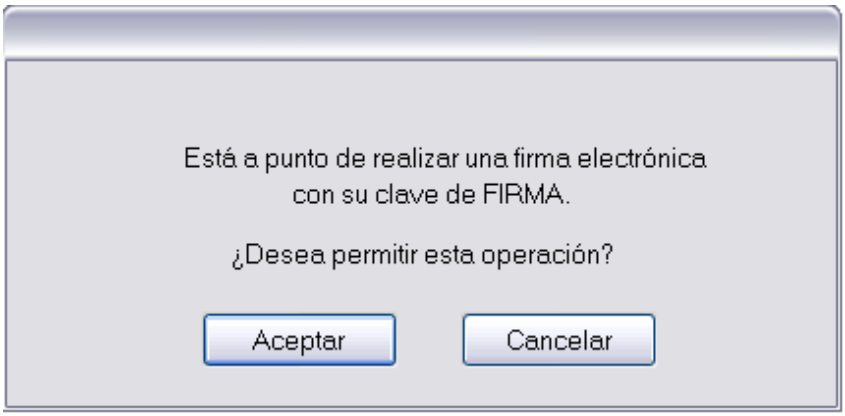

Figura 4-14: Advertencia – Firma con certificado de no repudio

Dicha advertencia al usuario se realiza por el hecho de que, al ser un certificado que provee de no repudio del contenido firmado, se identificará de forma inequívoca al firmante con el documento firmado. En caso de que el usuario no quiera que se le asocie con dicho documento puede abortar el proceso de firma pulsando el botón *"Cancelar"*.

Si la firma se realiza sin problemas aparecerá el siguiente mensaje indicando que el proceso de firma ha finalizado y el archivo con la firma se ha guardado correctamente en el directorio de destino indicado por el usuario.

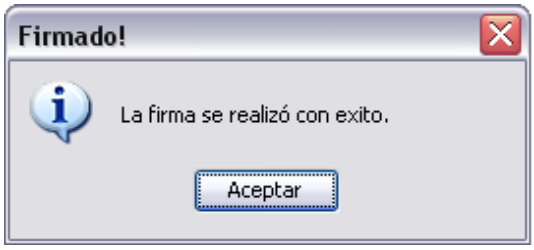

Figura 4-15: Mensaje de firma realizada

En este punto la aplicación regresa al panel de la interfaz principal, habilitado de nuevo, donde el usuario podrá elegir continuar trabajando con el programa para realizar más firmas o bien validar archivos firmados.

4.3.2 Ejemplo: Firma Digital de Formulario

En este ejemplo se muestran los pasos a seguir para realizar la firma sobre la información introducida en un formulario incluido en la aplicación. En este proceso se siguen los siguientes pasos:

1) Se crea un documento *PDF* a partir de los datos introducidos por el usuario de la aplicación en el panel de formulario de la interfaz gráfica principal.
2) Se solicitan los datos necesarios al usuario, tales como carpeta de destino donde guardar la salida, PIN para la TII, certificado a utilizar para la firma.
3) Se genera una firma digital del documento generado en el paso 1) que se incluye de forma visible en éste.
4) Se realiza el proceso de firma sobre el documento *PDF* obtenido en el paso anterior de la misma manera que en el proceso de *Firma Digital de Archivo*. De este paso se obtiene el archivo final de salida, en el que se incluye el documento *PDF* generado en el paso 3).

Este ejemplo parte de la pantalla principal de la interfaz gráfica de la aplicación. En ella se debe seleccionar el panel etiquetado como *"Firma de Formulario"*. De esta forma la aplicación tendrá un aspecto similar al mostrado por la figura 4-16, si bien puede diferir ligeramente, ya que, como se ha mencionado anteriormente, la aplicación está programada para obtener el *Look & Feel* del sistema en el que se ejecuta.

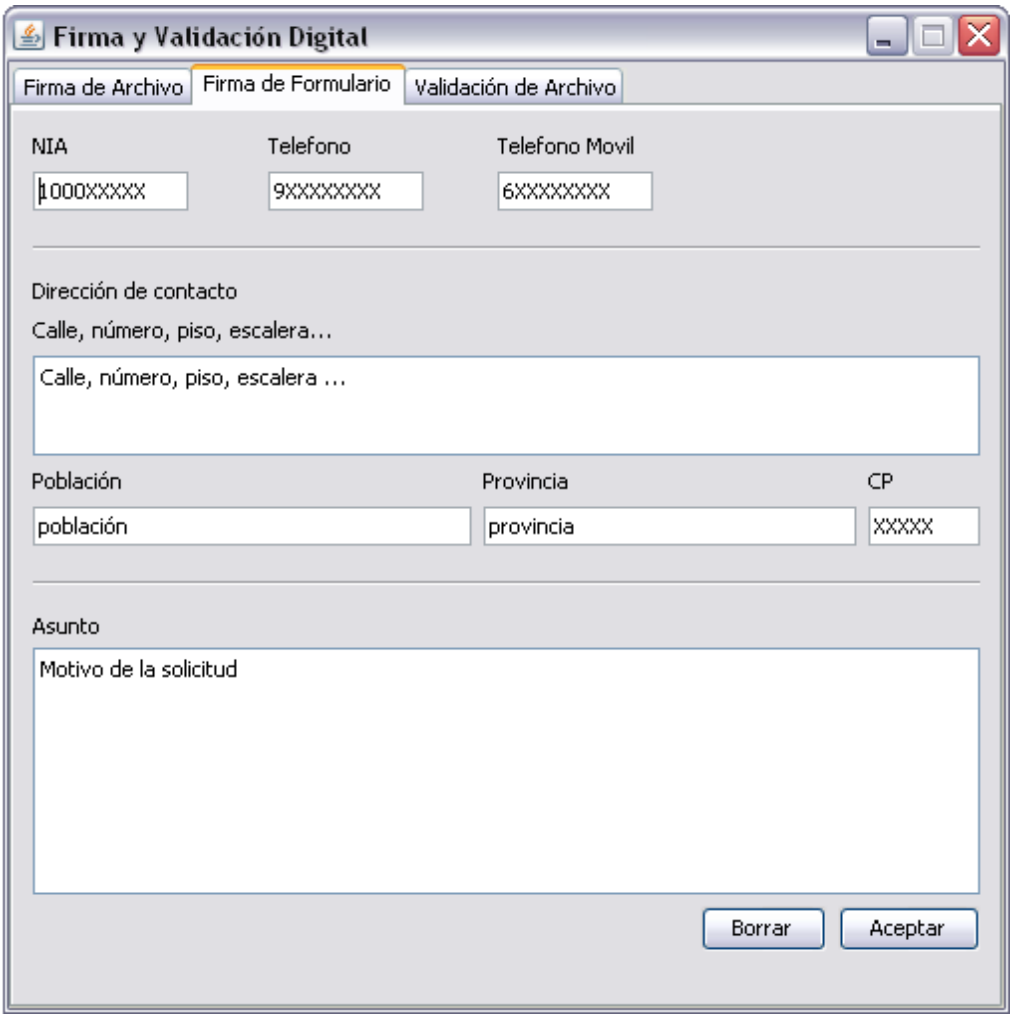

Figura 4-16: Panel de Firma de Formulario

Este panel muestra distintos campos que pueden ser rellenados por el usuario y dos

botones, *"Borrar"* y *"Aceptar"*. El botón *"Borrar"* borra el texto de todos los campos. El botón *"Aceptar"* convierte la información reflejada en el formulario en un documento *PDF* que guarda en memoria para su posterior firma y hace que la aplicación solicite al usuario un directorio de destino donde almacenar el archivo resultante de aplicar la firma al *PDF* recién creado, al que se insertará antes de firmarlo una firma *PDF* interna.

El formulario incluido en la aplicación sirve tan sólo de ejemplo ya que se puede reemplazar por cualquier combinación de campos de texto con una sencilla edición de la clase java que lo define. Siguiendo determinadas pautas en la personalización del formulario, que se indican en el apartado *5.4 Generación de PDF a partir de formulario*, el programa será capaz de interpretar el nuevo formulario sin necesidad de profundizar en el código que transforma el formulario a *PDF*.

La figura 4-17 muestra cómo queda deshabilitada la interfaz principal al pulsar el botón *"Aceptar"*. De esta forma queda garantizado que el usuario no pulsa nada que pueda interrumpir o causar un mal funcionamiento durante el proceso de firma.

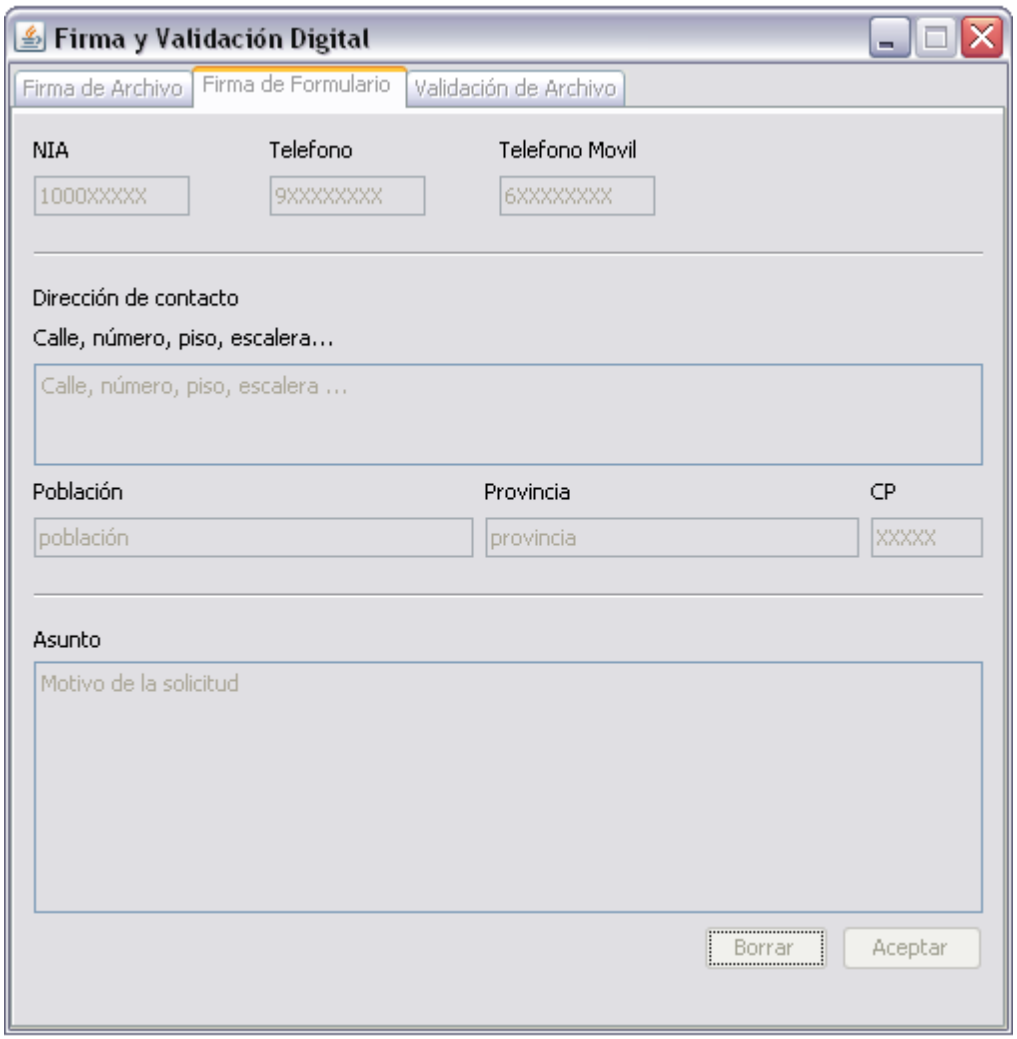

Figura 4-17: Panel de Firma de Formulario deshabilitado

A la vez que se deshabilita la interfaz principal de la aplicación, se muestra al usuario la ventana de selección de directorio de salida. En ella el usuario puede seleccionar el destino, dentro del sistema de archivos del ordenador en el que se está ejecutando la aplicación, donde quiere que se almacene el archivo de salida que contendrá el documento *PDF* firmado y la firma que se ha calculado sobre éste.

Dicha ventana aparece en la figura 4-18. En ella se aprecia cómo el campo del nombre de archivo de salida contiene el texto "*SignedForm.pdf_signed.xml*". El nombre *"SignedForm.pdf"* es el nombre que se le da a los documentos *PDF* generados a partir del formulario de ejemplo incluido en la aplicación. La terminación *"_signed.xml"* es la terminación por defecto que se le da a los archivos generados por la aplicación. También en caso de dejar este campo en blanco se utilizará el nombre por defecto. En cualquier caso, el usuario puede cambiar el nombre del archivo de salida si así lo desea.

Figura 4-18: Ventana de selección de destino (Firma de Formulario)

El usuario puede seleccionar en qué unidad desea almacenar el archivo de salida mediante el desplegable de la parte superior izquierda. A la derecha aparecerá una estructura de árbol con el contenido de la unidad seleccionada. Pulsando el botón de *"Atrás"* se retrocede a la interfaz principal de la aplicación donde se puede seleccionar cualquiera de los tres paneles principales (*Firma de Archivo, Firma de Formulario, Validación de Archivo*). En este paso se debe seleccionar un directorio válido (ha de ser directorio, no archivo, con permiso de escritura). En caso de que se pulse el botón

"Aceptar" sin haber seleccionado un directorio válido se muestra el siguiente error.

Figura 4-19: Error de selección de directorio en ventana de selección de destino (Firma)

También puede darse el caso de que el directorio de destino seleccionado no tenga suficiente espacio para almacenar el archivo de salida que se generaría al procesar el archivo de entrada seleccionado. En este caso, el resultado es otro mensaje de error indicando que el fallo se ha dado por problemas de espacio en el sistema de archivos. Pulsando el botón *"Aceptar"* se retrocede nuevamente a la ventana de selección de destino, donde se podrá elegir otro directorio de destino.

Figura 4-20: Error de espacio insuficiente en directorio en ventana de selección de destino (Firma)

Si el directorio de destino seleccionado es válido (es directorio, tiene permiso de escritura y espacio suficiente) se procede a intentar acceder al almacén de certificados de la TII insertada en el lector.

En caso de que no se encuentre ninguna TII insertada en el lector aparecerá el siguiente mensaje de error, indicándole dicha circunstancia al usuario. Una vez que se pulsa el botón *"Aceptar"* en dicho mensaje de error, se vuelve a habilitar la interfaz principal de la aplicación para que el usuario elija qué tarea quiere realizar, si bien ninguna tarea de firma podrá ser completada hasta que se inserte una TII válida en el lector de tarjetas.

Figura 4-21: Error de acceso a la TII – TII no encontrada

Es posible que el módulo criptográfico suministrado por la autoridad emisora, el lector de tarjetas o incluso la propia TII den problemas al intentar establecer

comunicación con el almacén de certificados de la TII. Solventar estos fallos está más allá de las posibilidades de la programación de aplicaciones que interactúen con la TII, ya que estos errores son inherentes al diseño del lector en cuestión, de la TII y del módulo criptográfico necesario para comunicarse con ella. En estos casos la aplicación mostrará alguno de los mensajes de error de las siguientes figuras, indicando a su vez cuál fue el error que provocó el fallo de acceso a la TII.

Figura 4-22: Error de acceso a la TII – Error general

Si se muestra el mensaje de error de la figura 4-22, el error causante del fallo de la aplicación al intentar acceder a la TII es un error general del módulo criptográfico PKCS#11. El módulo ha realizado un autotest de consistencia de pares y no lo ha superado, con lo que ha entrado en estado de error. A partir de ese momento todas las funciones de PKCS#11 devuelven código de error.

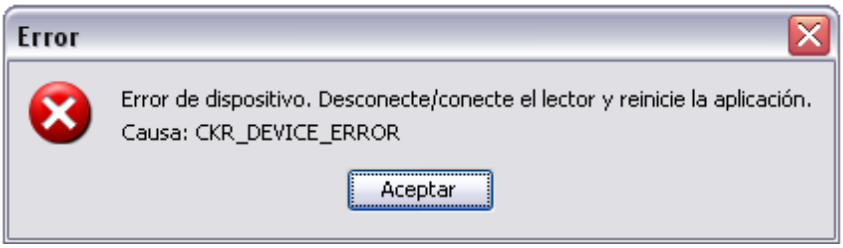

Figura 4-23: Error de acceso a la TII - Error de dispositivo

Cuando el mensaje de error obtenido es el representado en la figura 4-23, el error causante del fallo en la aplicación es, en este caso, un fallo en el módulo criptográfico PKCS#11. El módulo se encuentra en estado de error, generalmente por fallos en un autotest distinto al de consistencia de pares, independiente de la aplicación desarrollada en el ámbito de este proyecto, y hace que todas las funciones de PKCS#11 devuelvan código de error en vez de los resultados esperados.

En ambos casos, tanto en el caso de *CKR_DEVICE_ERROR* como en el de *CKR_GENERAL_ERROR* es conveniente cerrar la aplicación, extraer la TII y desconectar el lector (si es posible) antes de intentar de nuevo realizar el proceso de firma. El problema puede persistir incluso siguiendo estas instrucciones; en esos casos es posible que sea necesario reiniciar el sistema.

La normal ejecución de la aplicación debería sin embargo llegar al siguiente punto. En caso de que no se haya accedido aún a la TII habiendo introducido ya el PIN correctamente en la ejecución actual de la aplicación, se solicita al usuario el PIN de la TII tal y como se muestra en la figura 4-24.

Figura 4-24: Solicitud del PIN de la TII

En la ventana mostrada en la figura 4-24 se ofrecen al usuario un campo de texto donde escribir el PIN de su TII y dos botones, *"Aceptar"* y *"Cancelar"*. El campo de texto donde se debe introducir el PIN oculta mediante caracteres "●" el texto introducido por el usuario, de tal forma que un tercero cercano al monitor del ordenador donde se ejecuta la aplicación no pueda obtener el PIN del usuario observando "por encima del hombro". Pulsando el botón *"Cancelar"* se aborta el proceso de firma y la aplicación vuelve a habilitar la interfaz principal. Pulsando el botón *"Aceptar"* se intenta acceder a la TII con el PIN proporcionado, siempre que se haya introducido algún texto. En caso de que el campo del PIN esté vacío la aplicación no hace nada y la ventana mostrada en la figura permanece en pantalla.

En este punto se pueden dar varios casos de error: el PIN proporcionado no es correcto o bien el PIN de la TII está bloqueado debido a que se ha intentado acceder tres veces a ella con un PIN erróneo. En este último caso es imprescindible que el usuario acuda a un terminal de desbloqueo de la autoridad emisora de la TII para desbloquearla, ya que este desbloqueo únicamente se puede realizar en las máquinas instaladas a tal efecto.

Figura 4-25: Error a) PIN erróneo, b) PIN bloqueado

Si el PIN introducido es el correcto entonces la aplicación accederá al almacén de certificados de la TII sin problemas. En este caso, si se sigue trabajando con la misma TII para firmar otros archivos, no se volverá a solicitar el PIN para acceder a ella. De esta forma resulta más sencillo firmar distintos archivos en una sola sesión con la aplicación desarrollada. La ventana que aparece a continuación en el proceso de firma se muestra en la figura 4-26.

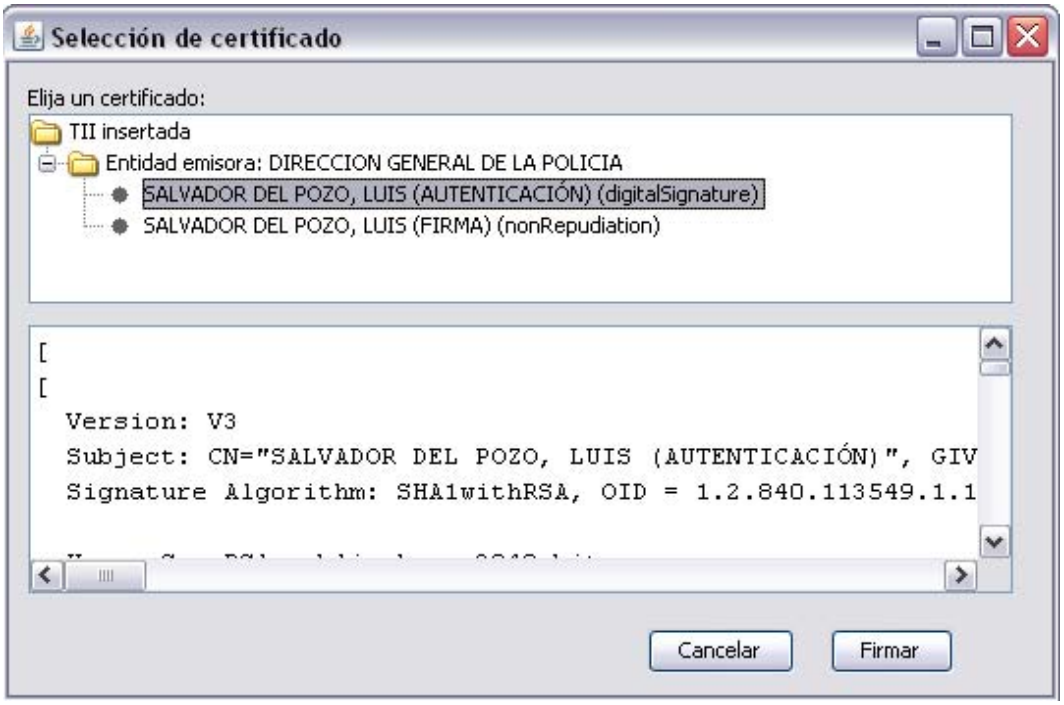

Figura 4-26: Ventana de selección de certificados

Al intentar acceder al almacén de certificados, puede parecer que la aplicación ha quedado bloqueada o que no responde. Esto se debe a que el proceso de establecer un canal de comunicación con la TII es relativamente lento. En unos instantes se mostrarán por pantalla los certificados almacenados en la TII para que el usuario elija con cuál desea realizar la firma digital, tal y como se muestra en la figura 4-26. Los distintos certificados almacenados en la TII aparecen en forma de árbol. Cuando se selecciona uno de los dos certificados de usuario aparece en el recuadro inferior la información de dicho certificado.

En la ventana de *Selección de Certificado* se muestran también dos botones, *"Cancelar"* y *"Firmar"*. Pulsando el botón cancelar se aborta el proceso de firma y se vuelve a activar la interfaz principal de la aplicación. Si se ha seleccionado un certificado, pulsando el botón *"Firmar"* la aplicación creará un nuevo hilo de forma transparente al usuario para realizar la firma con el certificado elegido. En este punto pueden surgir dos problemas, aunque no son comunes.

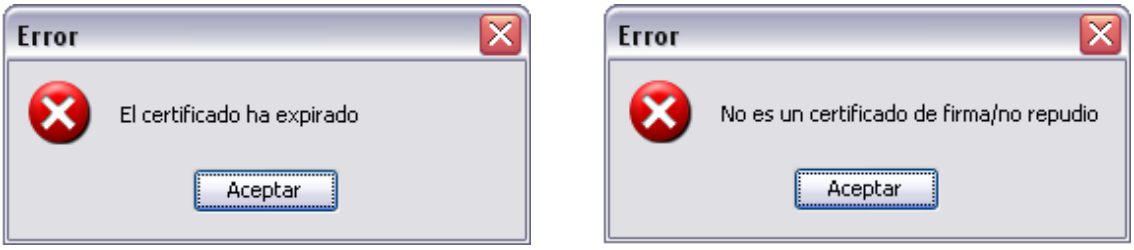

Figura 4-27: Error a) Certificado expirado, b) Certificado no válido

El error representado en la figura 4-27-a aparecerá cuando el certificado almacenado en la TII haya superado su periodo de validez y deba ser renovado en algún terminal de la autoridad emisora. El error representado en la figura 4-27-b es altamente improbable que ocurra y no debería suceder, pero cubre el caso en que el certificado seleccionado almacenado en la TII no esté definido como de firma digital o de no repudio. En ambos casos el proceso de firma se abortará y se retornará al panel de la interfaz principal de la aplicación.

En caso de que el certificado seleccionado por el usuario sea el que provee no repudio del documento firmado aparecerá tres veces la advertencia mostrada en la figura 4-28 en pantalla. Dicha advertencia es ajena a la aplicación desarrollada en el ámbito de este proyecto; se lanza a través del módulo criptográfico suministrado por la autoridad emisora de la TII utilizada.

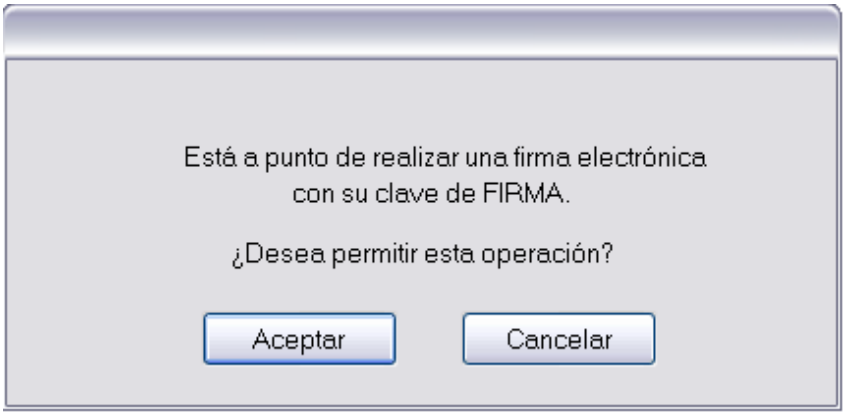

Figura 4-28: Advertencia - Firma con certificado de no repudio

Dicha advertencia al usuario se realiza por el hecho de que, al ser un certificado que provee de no repudio del contenido firmado, se identificará de forma inequívoca al firmante con el documento firmado. En caso de que el usuario no quiera que se le asocie con dicho documento puede abortar el proceso de firma pulsando el botón *"Cancelar"*. El hecho de que esta advertencia aparezca tres veces en vez de una, como sucede en el caso de firma de un archivo presente en el sistema de archivos del ordenador que ejecuta la aplicación, se debe a que en el proceso de firma del documento *PDF* se debe utilizar dos veces más el certificado del usuario para incluir una firma *dentro* de aquél.

Si la firma se realiza sin problemas, aparecerá el siguiente mensaje indicando que el proceso de firma ha finalizado y el archivo con la firma se ha guardado correctamente en el directorio de destino indicado por el usuario.

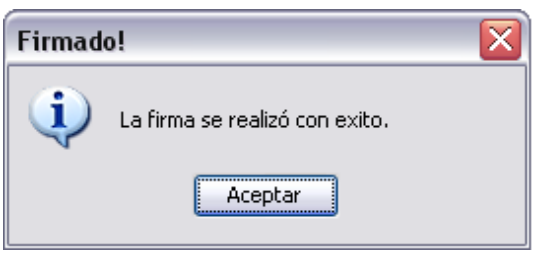

Figura 4-29: Mensaje de firma realizada

En este punto la aplicación regresa al panel de la interfaz principal, habilitado de nuevo, donde el usuario podrá elegir continuar trabajando con el programa para realizar más firmas o bien validar archivos firmados.

4.3.3 Ejemplo: Validación de Archivo Firmado

En este ejemplo se muestran los pasos a seguir para realizar la validación de un archivo generado mediante la aplicación desarrollada en este proyecto. La aplicación comprobará si el archivo tiene un formato válido (el formato se explica en detalle en la sección *4.5 Formato de Salida*) y, en caso afirmativo, comprueba si la firma es válida. Si el archivo analizado pasa el test de validación, se extrae el archivo original contenido dentro al directorio de destino indicado por el usuario. Es el único caso de uso de la aplicación desarrollada en el que no se necesita acceder a la TII, ya que toda la información necesaria para la validación de un archivo firmado con anterioridad se encuentra almacenada en el propio archivo firmado, tal y como impone el formato de firma digital XAdES.

El proceso de *Validación de Archivo Firmado* sigue los pasos indicados a continuación, partiendo siempre de un archivo válido y firmado generado por la aplicación desarrollada en el ámbito de este proyecto.

1) La aplicación analiza sintácticamente el archivo que se quiere validar para obtener los distintos elementos del árbol xml de que consta y que serán utilizados en la validación. Dichos elementos necesarios para la validación son los siguientes:
    a. Las referencias a URIs del documento cubiertas por la firma y los valores de *digest* obtenidos para ellas.
    b. El valor de la firma obtenido de aplicar el algoritmo de firma (indicado también en el archivo a validar) sobre los *digest* indicados en 1) a. Dicha firma no es otra cosa que esos *digest* canonizados y codificados con la clave privada del firmante del archivo a validar.
    c. La clave pública del firmante del archivo a validar.
2) Se obtiene el valor de los *digest* indicados en 1) a., usando para ello la clave pública del firmante para decodificar el valor de la firma electrónica indicado en 1) b.
3) La aplicación calcula los *digest* de las URIs del documento referidas en 1) a.
4) Se comprueba si los valores obtenidos en 3) se corresponden con los obtenidos en 1) a. del archivo a validar y si coinciden con los obtenidos en 2). La validación sólo es satisfactoria si se cumple lo anterior.

Este ejemplo parte de la pantalla principal de la interfaz gráfica de la aplicación. En ella se debe seleccionar el panel etiquetado como *"Validación de Archivo"*. De esta forma, la aplicación tendrá un aspecto similar al mostrado por la figura 4-30, si bien puede diferir ligeramente, ya que como se ha comentado anteriormente, la aplicación está programada para obtener el *Look & Feel* del sistema en el que se ejecuta.

En dicho panel se muestra a la izquierda un desplegable con todas las unidades válidas en las que se tiene permiso de lectura, y a la izquierda se muestra el árbol de archivos de la unidad seleccionada. No se muestran las carpetas ocultas para evitar problemas con la estructura de directorios del sistema en el que se ejecuta la aplicación. El usuario debe seleccionar un archivo para el que desee comprobar la validez de la firma. Si se pulsa el botón *"Aceptar"* antes de haber seleccionado un archivo válido (debe ser un archivo, no un directorio, y debe tener permiso de lectura) se mostrará el error mostrado en la figura 4-31.

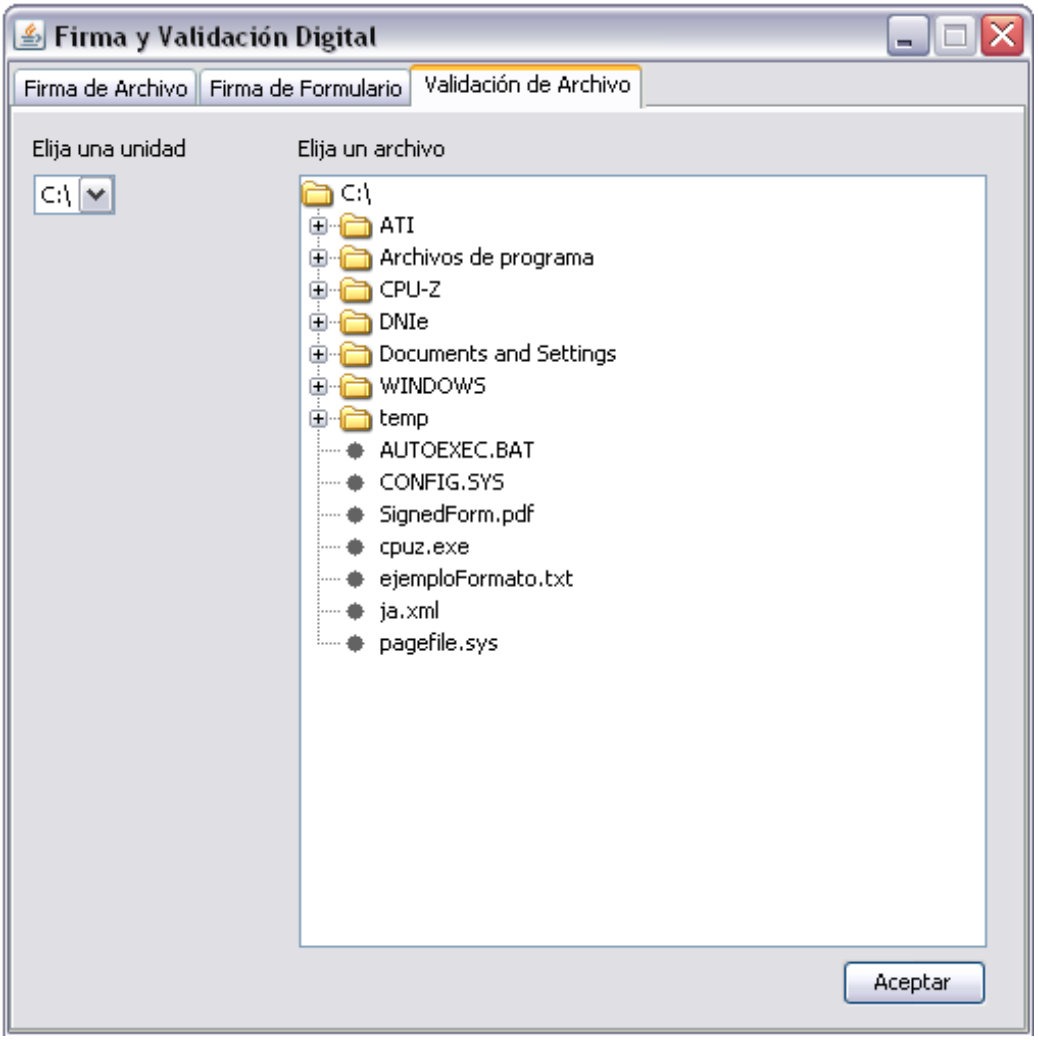

Figura 4-30: Panel de Validación de Archivo

Figura 4-31: Error en Panel de Validación de Archivo

Una vez que el usuario ha seleccionado un archivo válido para la comprobación de validez de firma, la interfaz principal de la aplicación queda deshabilitada hasta que se complete el proceso de validación, se aborte por parte del usuario o se produzca un error en el proceso. La interfaz principal queda por tanto como se muestra en la figura 4-32.

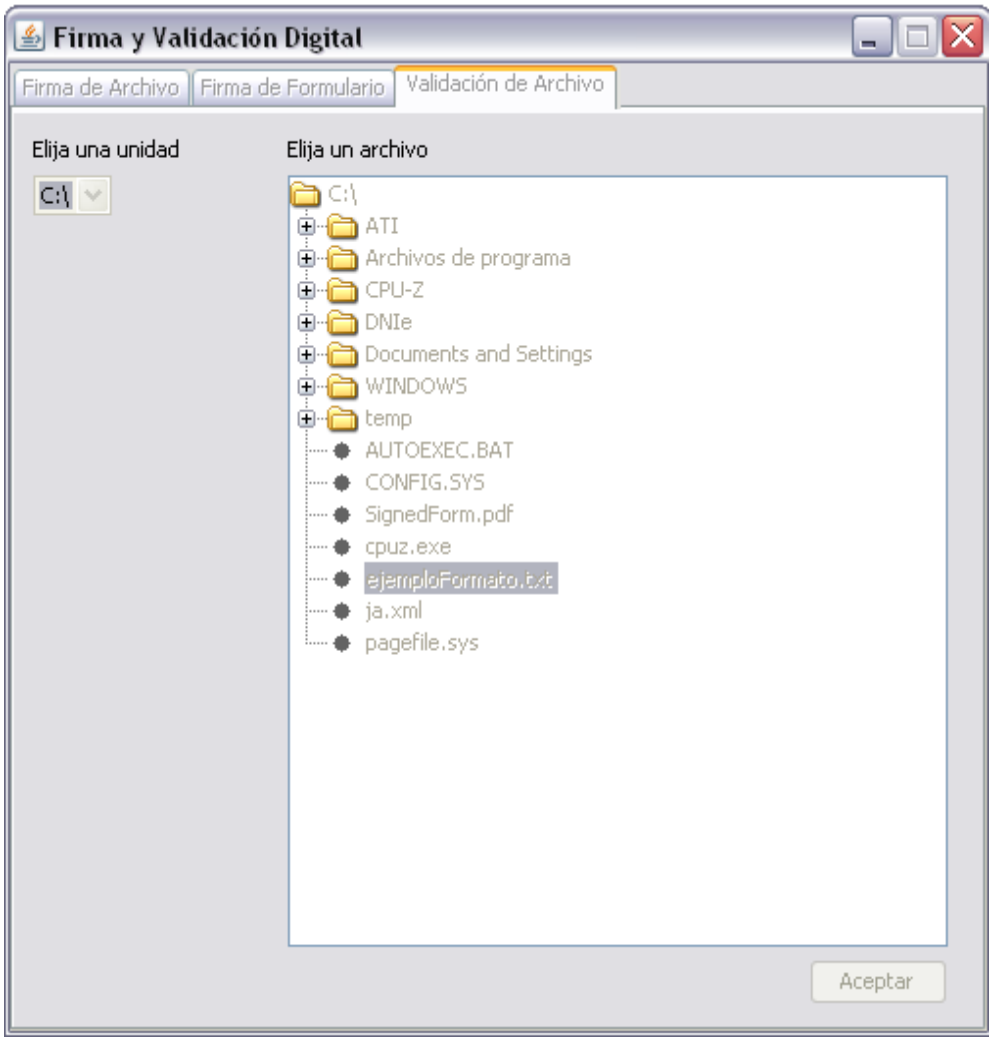

Figura 4-32: Panel de Validación de Archivo deshabilitado

De esta forma, el usuario no puede generar comportamientos no deseados mientras se ejecuta el proceso de firma pulsando por error o intencionadamente algún botón de la interfaz gráfica principal, ya sea por pulsar repetidamente el botón *"Aceptar"* recién pulsado o cualquier otro botón de los otros dos paneles.

Tras pulsar el botón de *"Aceptar"* en el panel de *Validación de Archivo* se muestra la ventana que aparece en la figura 4-33, la ventana de selección de directorio de salida. En ella, el usuario puede seleccionar el destino dentro del sistema de archivos donde quiere que se almacene el archivo de salida que contendrá el archivo original contenido en el archivo de firma que está siendo validado. Esta ventana se parece mucho al panel del que se parte en el proceso de *Validación de Archivo*. Sin embargo, en este desplegable de las unidades del sistema se muestran tan sólo las unidades en las que se

tiene permiso de escritura y además no están llenas. Posteriormente se comprobará que exista espacio suficiente en la unidad de destino para albergar el archivo de salida generado por la aplicación. También es muy similar a la ventana mostrada en los dos casos de firma, pero difiere, además de en el texto informativo mostrado en el recuadro de texto, en que esta vez no hay ningún campo a la vista en el que introducir el nombre con el que se desea que se guarde el archivo. Esto se debe a que el nombre del archivo de salida será igual que el nombre que tenía el archivo original al calcular la firma sobre él e incluirlo en el archivo que está siendo validado.

Figura 4-33: Ventana de selección de destino (Validación de Archivo)

Esta ventana también cuenta con dos botones, el de *"Atrás"* y el de *"Aceptar"*. Pulsando el botón de *"Atrás"* se retrocede a la interfaz principal de la aplicación donde se puede seleccionar cualquiera de los tres paneles principales (*Firma de Archivo, Firma de Formulario, Validación de Archivo*). Pulsando el botón *"Aceptar"* la aplicación pasa a iniciar el proceso de validación del archivo indicado, siempre que se haya escogido un directorio de destino válido. En caso de no haber seleccionado ningún directorio válido (un archivo, un directorio sin permiso de escritura o no haber seleccionado nada) se muestra el siguiente error.

Figura 4-34: Error de selección de directorio en ventana de selección de destino (Validación)

También puede darse el caso de que el directorio de destino seleccionado no tenga suficiente espacio para almacenar el archivo de salida que se generaría al extraer el archivo original del archivo de entrada seleccionado. En este caso el resultado es otro mensaje de error indicando que el fallo se ha dado por problemas de espacio en el sistema de archivos. Pulsando el botón *"Aceptar"* se retrocede nuevamente a la ventana de selección de destino, donde se podrá abortar el proceso de validación o elegir otro directorio de destino que sí tenga espacio suficiente.

Figura 4-35: Error de espacio insuficiente en directorio en ventana de selección de destino (Validación)

Si el directorio de destino seleccionado es válido (es directorio, tiene permiso de escritura y espacio suficiente) se inicia el proceso de validación de firma. En primer lugar, la aplicación comprueba que el archivo tenga el formato necesario. De no ser así se mostrará el siguiente error y se abortará el proceso.

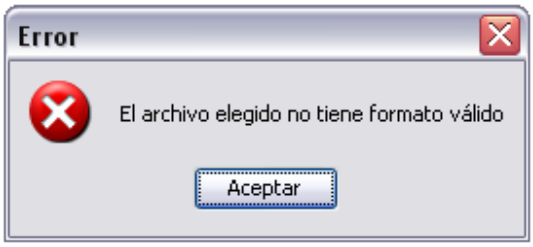

Figura 4-36: Error de Formato del Archivo Validado

El siguiente mensaje de error aparece si el archivo indicado no pasa el test de validación de firma o tras haber saltado el error mostrado en la figura 4-36. Como se indica en el propio mensaje de error, al no haberse superado el test de validación de firma no se extraerá ningún archivo al sistema de archivos de la máquina que está ejecutando la aplicación. La aplicación aborta el proceso de validación y devuelve al usuario a la interfaz principal de la aplicación, nuevamente desbloqueada para permitir las acciones del usuario.

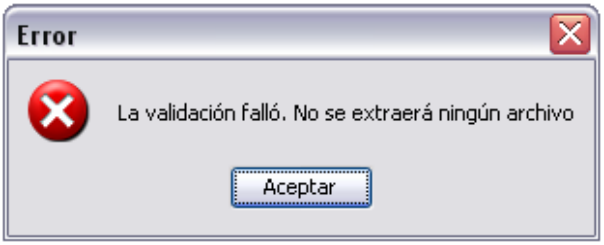

Figura 4-37: Error - Test de validación de firma no superado

Si el archivo indicado por el usuario tiene el formato adecuado y supera el test de validación de firma se extrae el archivo original sobre el que se aplicó la firma y se guarda en el destino indicado del sistema de archivos del ordenador en el que se está ejecutando la aplicación. También se indica al usuario mediante un mensaje informativo que la validación ha sido superada junto con el nombre del firmante, el tipo de certificado utilizado para la firma y un identificador único de usuario. Este mensaje informativo es el que se muestra en la figura 4-38.

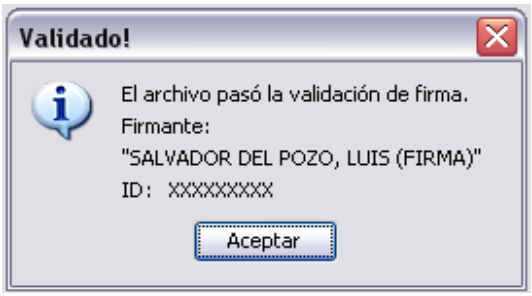

Figura 4-38: Validación superada con éxito

En este punto la aplicación vuelve a habilitar la interfaz principal de la aplicación y se da por concluido el proceso de validación de archivo.

4.4 Diagramas de flujo de la aplicación

Una vez explicado el funcionamiento de la aplicación desarrollada mediante los ejemplos anteriores, es posible y conveniente representar los diagramas de flujo de la ejecución. Estos diagramas de flujo pueden ser utilizados como una sencilla guía del manejo de la aplicación, a pesar de la ya de por sí intuitiva interfaz de usuario desarrollada. Además plantean una visión general del funcionamiento de la aplicación, necesaria no sólo para los posibles usuarios de la misma, sino también para futuros desarrolladores que utilicen esta aplicación como base para adaptarla a sus requerimientos.

En los siguientes apartados se muestran los tres diagramas de flujo para las distintas situaciones de uso de la aplicación mostradas en los ejemplos anteriores.

4.4.1 Diagrama de flujo: Firma de Archivo

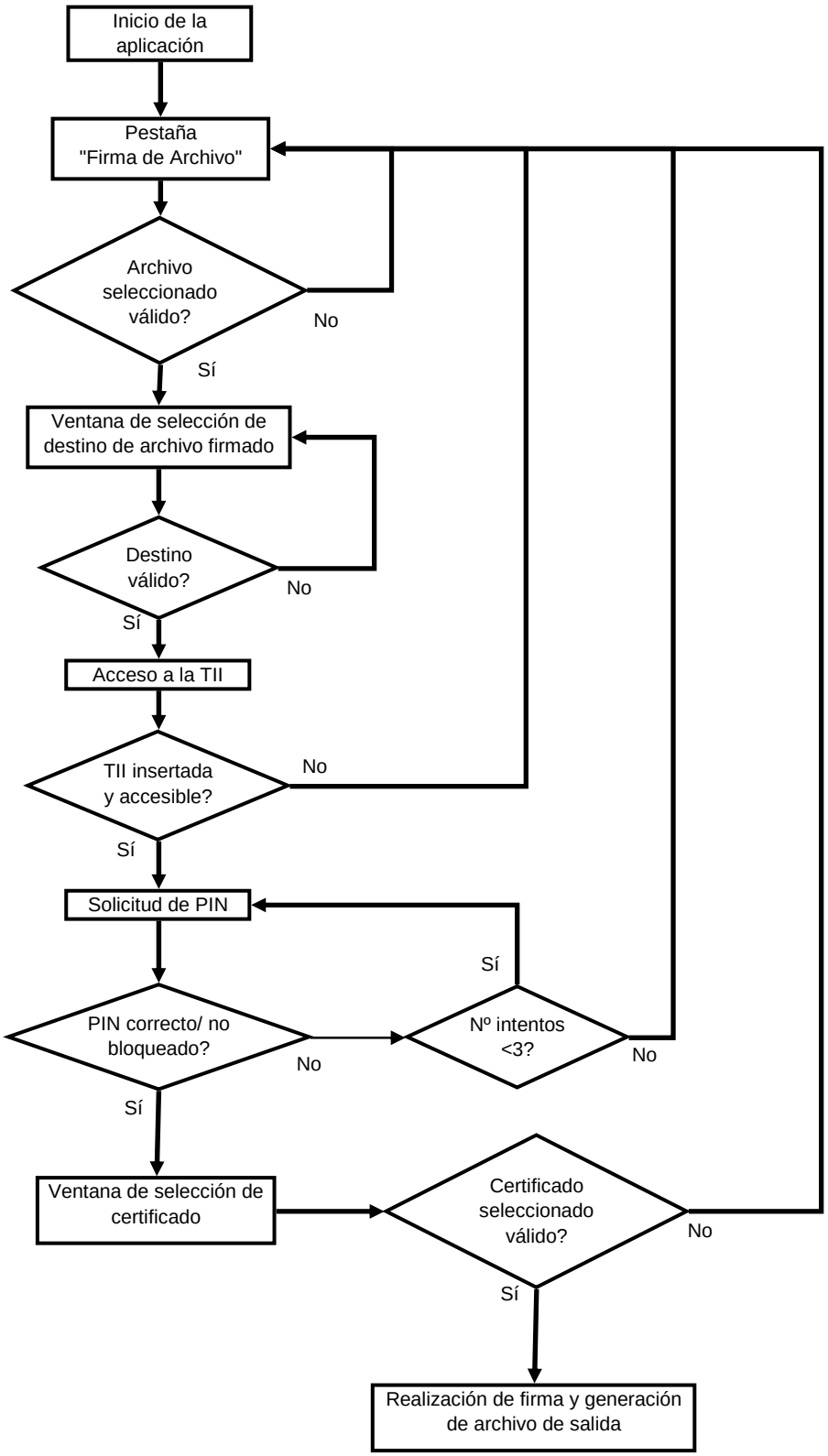

Figura 4-39: Diagrama de flujo: Firma de Archivo

4.4.2 Diagrama de flujo: Firma de Formulario

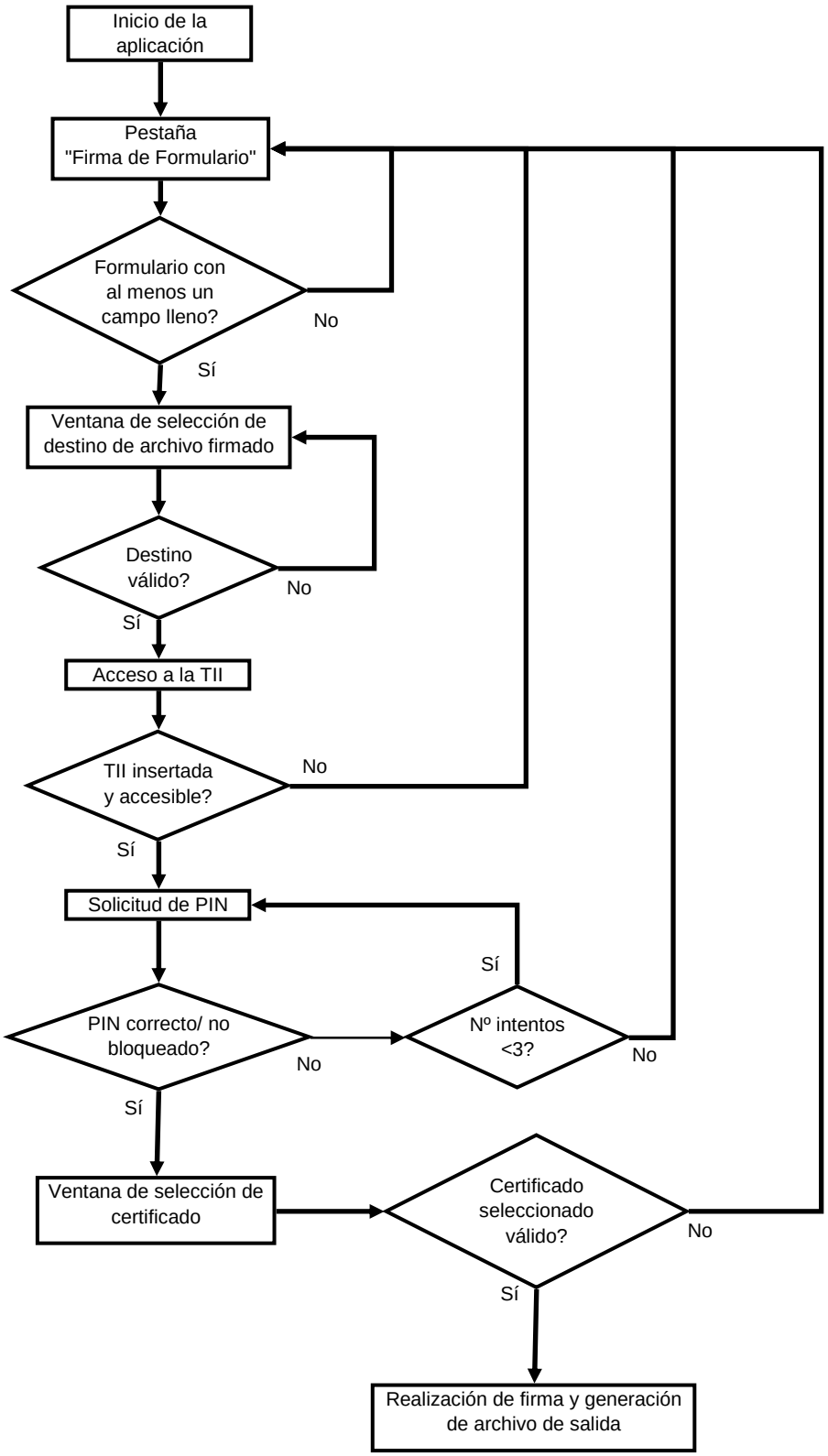

Figura 4-40: Diagrama de flujo: Firma de Formulario

4.4.3 Diagrama de flujo: Validación de Archivo Firmado

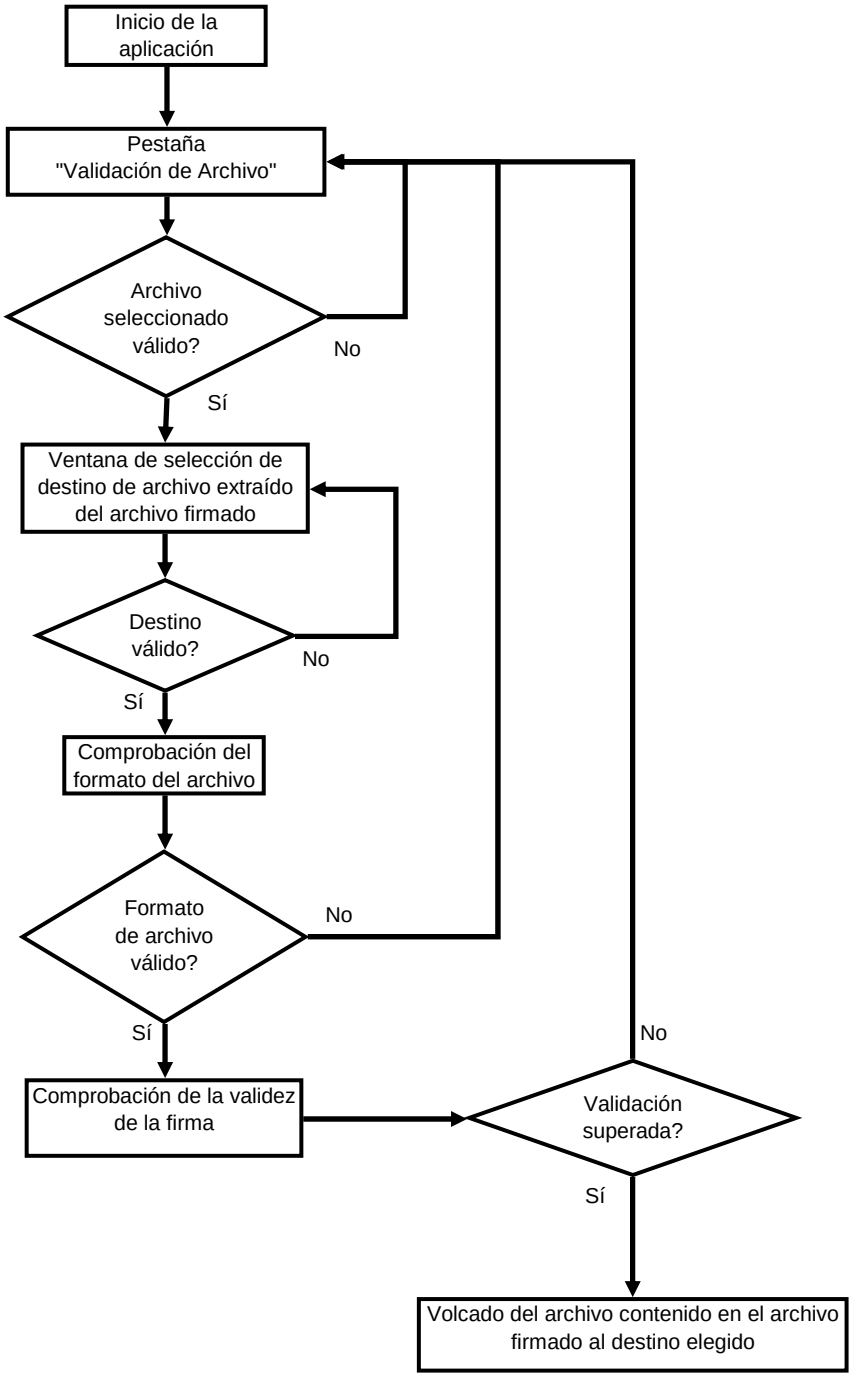

Figura 4-41: Diagrama de flujo: Validación de Archivo Firmado

4.5 Salida de la aplicación

En este apartado se explica con más detalle el formato de salida de la aplicación, no sólo respecto al archivo final de salida sino también el formato del archivo *PDF* generado en el proceso de *Firma de Formulario*.

4.5.1 Formato del archivo de salida

La salida generada por la aplicación en los dos procesos de firma que el usuario puede usar, *Firma de Archivo* y *Firma de Formulario,* es un archivo que contiene texto en formato *xml*. Este texto conforma un árbol *xml* en el que se almacena:

- El nombre del archivo a partir del que se ha generado el archivo de salida.
- El contenido del archivo a partir del que se ha generado el archivo de salida, almacenado en formato Base64.
- La firma digital que garantiza que no se ha modificado el archivo de salida.
- La información necesaria para comprobar la validez de la firma por cualquier usuario de la aplicación desarrollada sin necesidad de tener acceso a la TII del firmante.

Cabe indicar que, como ya se ha comentado con anterioridad en este documento, el proceso de *Firma de Formulario* se trata como un proceso especial de *Firma de Archivo*. Al firmar el formulario se genera un documento *PDF* en el que se incluye una firma digital con el certificado elegido. Es este documento *PDF* firmado el que se incluirá en el archivo de salida de la aplicación, si el proceso de firma se completa satisfactoriamente, con un nombre predefinido (“*SignedForm.pdf*”) y como contenido, el obtenido al pasar el formulario a *PDF* y firmarlo. Por ello el formato de salida en los casos de *Firma de Archivo* y *Firma de Formulario* es idéntico.

La firma digital y la información necesaria para su validación se incluyen en el archivo siguiendo la estructura *xml* necesaria para respetar el formato de firma XAdES-BES, la forma más básica de firma digital aceptada legalmente según la directiva europea de firma electrónica.

La estructura *xml* de un archivo de salida de la aplicación comienza creando un documento *xml* en memoria (la aplicación no vuelca nada al sistema de archivos salvo si el proceso de firma finaliza correctamente) en el que se almacena la información referente al archivo sobre el que se calcula la firma. La siguiente figura muestra un ejemplo del árbol inicial del documento, únicamente con la información del archivo sobre el que se calcula la firma.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<archivo Id="idSignedDoc">
      <nombre>ejemploFormato.txt</nombre>
      <datos64>RXN0byBzZSBmaXJtYXLh</datos64>
</archivo>
```

Figura 4-42: Estructura inicial del árbol *xml* del archivo de salida

Este ejemplo se obtiene de firmar un documento de texto generado con el programa *Notepad* de *Windows* con el texto *"Esto se firmará"*. Se puede observar en la figura 4-40 cómo en primer lugar aparece la declaración de qué versión de *xml* se sigue en el documento. A continuación comienza el árbol *xml* que contiene la información del archivo sobre el que se va a calcular la firma.

El nodo definido con la etiqueta "*archivo*" contiene en este momento dos elementos etiquetados como "*nombre*" y "*datos64*". El elemento "*nombre*" almacena el nombre del archivo a partir del cual se va a generar el archivo de salida tras calcular la firma. El elemento "*datos64*" contiene una representación en base 64 del contenido binario del archivo. De esta forma se evitan problemas a la hora de interpretar el árbol *xml* por caracteres no deseados, y además asegura que el contenido binario no será modificado al pasar por distintos sistemas que pudieran interpretar la secuencia de datos de forma errónea.

Al árbol *xml* creado con la información del archivo que se pretende firmar se le añade posteriormente la información necesaria de la firma: la firma en sí y la información necesaria para validarla posteriormente. La estructura de los nuevos elementos añadidos al árbol *xml* es la mostrada en la figura 4-41.

```
<ds:Signature
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="idSignature">

 <ds:SignedInfo>

  <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
  <ds:SignatureMethod
   Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>

  <ds:Reference URI="">
   <ds:Transforms>
    <ds:Transform
     Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
   </ds:Transforms>
   <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <ds:DigestValue>xBru1KmA71RYOtcYc0730m1/tm4=</ds:DigestValue>
  </ds:Reference>

  <ds:Reference URI="#idKeyInfo">
   <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <ds:DigestValue>c42L2CUI1aV/BXfflhUA+kf9M7Y=</ds:DigestValue>
  </ds:Reference>

  <ds:Reference Type="http://uri.etsi.org/01903/v1.3.2#SignedProperties"
   URI="#idSignedProp">
   <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <ds:DigestValue>CAmf/+GGusyDBT1cMxSanih0OWY=</ds:DigestValue>
  </ds:Reference>

 </ds:SignedInfo>

 <ds:SignatureValue Id="idSignatureValue">zBOI...M4sA==</ds:SignatureValue>

 <ds:KeyInfo Id="idKeyInfo">

  <ds:KeyValue>
   <ds:RSAKeyValue>
    <ds:Modulus>0RFRF63B5MGe...9r1iuQiww==</ds:Modulus>
    <ds:Exponent>AQAB</ds:Exponent>
   </ds:RSAKeyValue>
  </ds:KeyValue>

  <ds:X509Data>
   <ds:X509Certificate>MIIF2zCCBMO...zCcQCYE3zP2z5j0</ds:X509Certificate>
  </ds:X509Data>

  <ds:X509Data>
   <ds:X509SubjectName>CN=”SALVADOR DEL POZO, LUIS (FIRMA)” ,...,
    C=ES</ds:X509SubjectName>
  </ds:X509Data>

 </ds:KeyInfo>

 <ds:Object Id="idObject">
  <QualifyingProperties Target="idSignature">
   <SignedProperties id="idSignedProp">
    <SignedSignatureProperties/>
   </SignedProperties>
  </QualifyingProperties>
 </ds:Object>
```

Figura 4-43: Ejemplo de elemento *Signature* del archivo de salida

El elemento *Signature* define el espacio de nombres utilizado y su *Id* en su declaración:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/XML-DSig#" Id="idSignature">
```

Todos los elementos que siguen la nomenclatura de la firma *XML-DSig* tienen *ds* como prefijo, tal y como se indica en la definición de XAdES-BES (apartado *2.5.4.1* del presente documento).

Dentro del árbol *xml* del elemento *Signature* nos encontramos con cuatro elementos principales, algunos de ellos con sus propios subárboles:

- *SignedInfo*
- *SignatureValue*
- *KeyInfo*
- *Object*.

El elemento *SignedInfo* contiene la siguiente información

- El método de canonización utilizado: en este caso se utiliza el algoritmo definido en:
  `http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments`
- El método de firma: en este caso se utiliza el algoritmo *RSA-SHA1* para obtener la firma digital.
- Los elementos *Reference*: indican mediante referencias URI qué recursos del documento van a ser firmados. Existen tres elementos *Reference* en el archivo de salida. El primero de ellos tiene como URI simplemente “”. Esto indica que se va a firmar todo el documento. Se añaden además dos referencias a los elementos *KeyInfo* y *SignedProperties*. De esta forma se cubre también la información necesaria para validar la firma. En las tres referencias aparecen los elementos *DigestMethod* y *DigestValue*. *DigestMethod* indica el algoritmo utilizado para calcular el *digest* del elemento al que se hace referencia y *DigestValue* contiene dicho *digest*. En el caso del elemento *Reference,* que apunta a todo el documento, se indica además qué tipo de transformación se aplica al documento firmado para incluir la firma. En este caso se usa la transformación necesaria para obtener una *enveloped Signature*, es decir, una firma envuelta en el propio documento firmado.

El elemento *SignatureValue* aparece acortado en el ejemplo mostrado en la figura 4-41 para facilitar su inclusión en este documento. Este elemento contiene la firma, es decir, contiene el elemento *SignedInfo* en forma canonizada, resumida y encriptada con la clave privada del firmante.

El elemento *KeyInfo* del archivo de salida de la aplicación aporta la información necesaria para poder validar la firma. Contiene tres elementos:

- Elemento *KeyValue*: especifica la clave pública con la que validar la

firma. Al tratarse de una firma *RSA* este elemento se compone de una clave *RSA*, *RSAKeyValue*, compuesta por un módulo y un exponente.

- Certificado X509 del firmante: Este elemento se incluye como un elemento de datos *X509Data*. Permite comprobar la información del firmante.
- *X509SubjectName* del firmante: contiene un nombre X509 distinguido del sujeto titular de la TII utilizado en la firma.

El elemento *Object* aporta los campos necesarios para completar el formato de firma XAdES-BES. Los elementos contenidos en *Object* no aportan información adicional pero han de estar presentes para cumplir dicho formato. Queda a discreción de los futuros administradores de la aplicación la decisión de completar o no estos campos según se indica en *2.5.3.1 Basic Electronic Signature, XAdES-BES*, en este mismo documento.

Respecto al tamaño del archivo de salida, la siguiente relación entre el tamaño del archivo original sobre el que se quiere calcular la firma y el archivo final de salida, expresados ambos tamaños en *kilobytes*, es una buena aproximación con un margen de error muy pequeño:

$$tamaño\_salida = 5 + 1.334 \cdot tamaño\_entrada$$

Como punto final de este apartado, aunque no tenga que ver directamente con el formato "interno" de la salida de la aplicación, cabe añadir que el hecho de haber elegido la extensión *.xml* en el nombre por defecto de los archivos de salida (mostrados en las figuras 4-4 y 4-17) no ha sido aleatorio. Con esta extensión se pueden utilizar distintos navegadores (Internet Explorer, Mozilla Firefox, etc.) o algún editor *xml* para visualizar de forma rápida y sencilla el contenido de un archivo de salida generado mediante la aplicación desarrollada. Sin extensión *xml* los navegadores mencionados siguen pudiendo mostrar correctamente la información contenida en el archivo; sin embargo no representan la información en forma de árbol *xml*, sino como texto plano, con lo que se pierde la capacidad de comprobar la estructura de árbol del archivo de salida.

En todo caso queda a discreción del usuario cambiar la extensión del archivo de salida a su gusto, ya que el proceso de validación es independiente del nombre que tenga el archivo que se valida.

4.5.2 Formato del documento PDF

Cuando se utiliza la función *Firma de Formulario*, la aplicación convierte los campos del formulario a un árbol *XML*. Tras esto, una vez que se tiene el documento *XML* cargado en memoria, se realiza un análisis sintáctico del árbol generado a partir del documento y se crea un documento *PDF* a partir de él, utilizando las herramientas provistas por la librería Java *iText*. La forma en que se genera el documento *PDF* y sus

posibles variaciones de configuración se explican en el apartado 5.4 de este mismo documento.

Este documento *PDF* se firma posteriormente incluyendo una firma digital *dentro* del documento, de forma independiente de la firma que se calcula para incluir en el archivo de salida. En la siguiente figura se muestra un ejemplo del aspecto que tendrán los documentos *PDF* generados.

Figura 4-39: Ejemplo de documento *PDF* de salida

En la parte superior del documento *PDF* generado aparecen los distintos campos del formulario, alineados según su posición dentro del formulario de la aplicación (ver figura 4-16) de izquierda a derecha y de arriba a abajo. Los separadores son interpretados como saltos de línea. La firma aparece en la parte inferior izquierda. El formato de firma de Adobe (derivado de PKCS#7) queda embebido dentro del formato PDF y permite asociar una imagen, por lo que es uno de los más adecuados para su visualización. La especificación del formato es la 1.6 y para la visualización se emplea Adobe Acrobat Reader v7 o Foxit PDF Reader.

En el ejemplo, el documento se visualiza mediante *Adobe Reader 8*. En este programa (Adobe Reader) se ha incluido el certificado de usuario obtenido del documento, como un certificado de confianza tras haber comprobado la cadena de certificados, por lo que la firma aparece claramente marcada como válida. De no haber añadido el certificado de usuario a los certificados de confianza, la firma aparecería con un signo de interrogación, si bien se podría obtener del documento *PDF* toda la información del certificado con el que se firmó.

5 Aplicación de Firma y Validación: Configuración

A continuación se muestran distintas consideraciones necesarias, así como varias posibilidades, para el correcto despliegue de la herramienta de validación de certificados en PKI con tarjeta inteligente. De notable importancia es el requisito de memoria que se muestra en el siguiente apartado.

5.1 Requisitos de memoria

La aplicación desarrollada no vuelca ningún fichero temporal al sistema de archivos de la máquina en la que se ejecuta la aplicación. De esta forma no se generan archivos residuales en el sistema de archivos que, además de ocupar espacio innecesario, un tercero podría intentar utilizar de forma maliciosa. Tan sólo graba el archivo de salida de la operación realizada, es decir, el archivo original insertado en el archivo firmado en el caso de una operación de validación o el archivo firmado obtenido tras una operación de firma.

Sin embargo, tanto durante el proceso de validación como durante el de firma, es necesario almacenar en memoria una cantidad considerable de datos para no tener la necesidad de volcar archivos temporales al disco duro. La cantidad de datos almacenada en memoria depende directamente del tamaño de los archivos con los que se trabaja. En el caso de firma de formulario esto no suele ser un problema ya que los formularios transformados a *PDF* no tienen un gran peso en volumen de datos.

Si los archivos con los que se trabaja, tanto en validación como en firma de archivos, son grandes, se puede dar el caso de que la pila de memoria disponible para java se agote. Esto se debe a que internamente se deben mantener almacenados en memoria los archivos con los que se trabaja para poder realizar los cálculos necesarios en cada modo de operación (firma o validación). La memoria necesaria puede ser mayor que la que haya disponible en la pila de memoria de java. En estos casos saltará una excepción no controlable mediante cláusulas *throw* y *catch* y la aplicación quedará bloqueada. Este problema es especialmente crítico en el proceso de validación, en el que a la hora de decodificar el texto en Base64 para volcarlo al archivo de salida, se consume mucha más memoria que en el proceso de firma. En los ordenadores utilizados para las pruebas de configuración del tamaño de memoria necesario se ha llegado a desbordar una pila de memoria de 400MB al validar archivos firmados de 51MB. En cualquier caso, no suele ser necesario firmar archivos de esos tamaños.

Por ello es conveniente lanzar la aplicación, siempre que sea posible, con un argumento para la Máquina Virtual de Java que indique que, para esta aplicación, debe usar un límite superior de tamaño de pila de memoria definido, mayor que el límite por defecto.

De esta forma, se puede llegar a evitar completamente el problema de superar el tamaño de la pila de memoria de la *JVM,* siempre que se ajuste adecuadamente el tamaño máximo fijado para la pila. Desgraciadamente, no en todas las configuraciones de despliegue posibles se puede fijar ese tamaño máximo, por lo que unas configuraciones serán más convenientes que otras, como se verá en el apartado *5.3 Configuraciones de despliegue*.

5.2 Necesidad de firmado del jar contenedor

La aplicación de firma accede a recursos del sistema, tales como el sistema de archivos y el almacén de certificados de la TII. Por ello, siempre que se desee utilizar la aplicación partiendo de su *jar*, se debe firmar mediante un certificado válido de usuario para tener los permisos necesarios. A la hora de cargar el *jar* se consultará al usuario si confía en el firmante.

Actualmente el *jar* está firmado con un certificado autogenerado con una validez de 6 meses y se deja a discreción del desarrollador que vaya a hacer uso de la aplicación la firma del *jar* con un certificado de su propiedad de mayor duración. Se podría utilizar los propios certificados contenidos en una TII de una PKI de confianza para firmarlo.

5.3 Configuraciones de despliegue

La aplicación de firma digital y validación puede ser desplegada de distintas formas. Puede ser iniciada vía línea de comandos tanto en forma de archivo *jar*, como a través de sus clases compiladas, puede ser insertada como un applet en una página web y,

usando *JNLP*, puede ser desplegada al escritorio del usuario sin necesidad de navegador o bien ejecutada a partir de un enlace en una página web en una ventana nueva.

A continuación se muestran los pasos necesarios para cada uno de estos tipos de despliegue. El desarrollador que adopte esta aplicación a sus necesidades deberá decidir cuál de ellas es la más conveniente para sus propósitos.

5.3.1 Despliegue mediante línea de comandos (jar)

Para ejecutar la aplicación desde la línea de comandos utilizando el *jar* firmado se debe introducir lo siguiente:

```
java -jar -Xmx400M PFC_Firma.jar
```

Figura 5-1: Ejecución del *jar* mediante línea de comandos

El argumento *–jar* es el que indica a Java que se va a realizar la ejecución a partir de un contenedor *jar*. El argumento *–Xmx400M* le indica a Java que la máquina virtual sobre la que se ejecute esta aplicación debe utilizar un tamaño máximo de pila de memoria de 400MB. Este número se ha utilizado como ejemplo y es deber del desarrollador que utilice la aplicación el encontrar un tamaño máximo de pila que se ajuste al tamaño máximo de archivo que pretende firmar.

Con este método no es posible asegurar que el *Java Runtime Environment* instalado sea compatible con la aplicación (se debe recordar que no se garantiza su correcto funcionamiento con versiones de Java anteriores a la 1.6.11).

5.3.2 Despliegue mediante línea de comandos (clases)

La instrucción que se debe introducir en la línea de comandos para ejecutar la aplicación a partir de las clases ya compiladas es la siguiente:

```
java -Xmx400M PFC.appletFirma.AppletFirma
```

Figura 5-2: Ejecución de las clases ya compiladas mediante línea de comandos

Esto es así siempre que el usuario se encuentre en el directorio raíz de la aplicación. Nuevamente el argumento *–Xmx400M* indica a Java que se debe utilizar un tamaño de 400MB como tamaño máximo de la pila de memoria de la *JVM*. Este método de despliegue es prácticamente idéntico en funcionamiento al anterior, con lo que nuevamente no es posible asegurar que el *Java Runtime Environment* instalado en la máquina en que se ejecuta la aplicación sea compatible con ésta. Queda de nuevo en manos del usuario la realización de esta comprobación.

5.3.3 Despliegue como Applet

Este método de despliegue presenta varios inconvenientes. En primer lugar, se depende de métodos ajenos a la aplicación para garantizar que la versión instalada sea compatible con la aplicación y, por lo tanto, no se garantiza su correcto funcionamiento. Se puede conseguir una versión de java compatible con la aplicación, sin demasiada dificultad, haciendo uso de herramientas como el script *deployJava.js* facilitado por Sun, que aporta funciones para comprobar la versión de *JRE* instalada y para descargar la versión más reciente en caso de ser necesario.

En segundo lugar, no hay forma de ajustar el tamaño máximo para la pila de memoria de la *JVM*. De esta forma, salvo que el usuario lo tenga configurado de forma local, el tamaño máximo de archivos que se podrán firmar, y de forma más crítica, validar, se reduce considerablemente.

Para cargar la aplicación insertada en una página web como un applet se debe introducir el siguiente código HTML dentro del elemento *BODY* de la página en la que se quiere incluir:

```
<applet id="AppletFirma" code="PFC.appletFirma.AppletFirma"
                width="500" height="485"
                codebase="<almacén del jar>"
                archive="PFC_Firma.jar">
</applet>
```

Figura 5-3: Código HTML para incrustar la aplicación como *applet* en una página web

Si el *jar* de la aplicación no se encuentra en el directorio raíz, se debe incluir en *codebase* el directorio relativo del servidor donde se encuentra. En caso contrario, dicho campo se puede omitir. Los campos *width* y *height* definen la altura y anchura que ocupará la aplicación en la página al ser cargada como applet. El campo *archive* define el nombre del *jar* que contiene la aplicación y el campo *code* indica la clase que se debe invocar al cargar la página. Esta clase, *PFC.appletFirma.AppletFirma* hereda de la clase *javax.swing.JApplet*, con lo que puede ser usada para arrancar como applet, ya que implementa el método *init()* necesario que el navegador invocará al cargar correctamente la página.

5.3.4 Despliegue mediante *JNLP*

La aplicación desarrollada en el ámbito de este proyecto ha sido programada aprovechando las ventajas que aporta el sistema *JNLP* de Java. *JNLP* (*Java Network Launching Protocol*), desarrollada por Sun Microsystems, permite arrancar aplicaciones Java que están en un servidor web de aplicaciones, comprobando previamente si el cliente tiene la versión actualizada de dicha aplicación. Si no es así descargará la última versión y se ejecutará en local. El arranque de dichas aplicaciones puede ser efectuado mediante enlaces en una página web o bien a través de enlaces en el escritorio cliente. Mediante está tecnología se asegura que la aplicación se distribuye

siempre en su última versión. Los ficheros que contienen la información sobre donde se encuentra la aplicación, versión, versión java necesaria, etc. tienen la extensión *.jnlp*.

Un cliente *JNLP* es una aplicación o servicio que puede cargar aplicaciones desde recursos albergados en una red. Si se empaqueta una aplicación con *JNLP* un cliente podrá:

- Detectar, instalar y usar la versión correcta del *Java Runtime Environment*, *JRE*, necesaria para el correcto funcionamiento de la aplicación que se pretende lanzar.
- Lanzar la aplicación desde un navegador o desde el escritorio del usuario.
- Descargar automáticamente nuevas versiones de la aplicación según se vayan publicando.
- Almacenar en cache local las clases usadas por la aplicación para arranques más rápidos de la aplicación.
- Descargar librerías nativas en caso de que sea necesario.
- Usar recursos locales de forma segura, tales como el sistema de archivos local de la máquina que ejecuta la aplicación.
- Pasar argumentos a la línea de comandos o propiedades a la Máquina Virtual de Java.

Estos archivos *.jnlp* también siguen sintaxis *xml*. El archivo *.jnlp* utilizado para cargar la aplicación se muestra en la figura 5-4.

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp codebase="http://192.168.1.2:8084/PFC/" spec="6.0+">

  <information>
    <title>Firma y Validación Digital</title>
    <vendor>Luis Salvador del Pozo</vendor>
    <description>Firma y Validacion mediante TII</description>
    <offline-allowed/>
    <shortcut online="false">
      <desktop/>
      <menu submenu="PFC Firma"/>
    </shortcut>
  </information>

  <security>
    <all-permissions/>
  </security>

  <resources>
    <java version="1.6+" max-heap-size="500M"/>
    <jar href="PFC_Firma.jar"/>
  </resources>

  <application-desc main-class="PFC.appletFirma.AppletFirma"/>

</jnlp>
```

Figura 5-4: *JNLP* para cargar la aplicación

Cada uno de los elementos que parten del nodo raíz puede tener combinaciones mucho mayores de sub-elementos que las mostradas en la figura 5-4, pero no entra dentro del campo de aplicación de este documento cubrirlas todas, con lo que sólo se explican a continuación los elementos utilizados. Para una descripción completa de la sintaxis de *JNLP* se recomienda visitar la página de documentación de *Sun* [3] y la documentación obtenida de [4].

El elemento *jnlp* es el elemento raíz. Tiene cuatro sub-elementos: *information*, *security*, *resources* y *application-desc*. Se podría sustituir este último elemento por uno de tipo *applet-desc* para lanzar la aplicación pero resulta visualmente más agradable hacerlo de esta forma. Tiene los siguientes atributos:

- El atributo opcional *codebase* define una URL base para todas las URLS relativas definidas en el mismo archivo *.jnlp*.
- El atributo opcional *spec* indica la versión de *Java Web Start* con la que se podrá trabajar.

El elemento *information* debe contener al menos los elementos *title* y *vendor*. Adicionalmente se pueden incluir otros elementos, aunque en este documento sólo se documentan los utilizados:

- El elemento t*itle*: indica el nombre de la aplicación
- El elemento *vendor*: indica el nombre del proveedor de la aplicación.
- El elemento opcional *description*: provee una descripción por defecto que *Java Web Start* mostrará en caso de solicitar una descripción de la aplicación.
- El elemento opcional *offline-allowed*: especificando este elemento, optativo, *Java Web Start* comprobará al arrancar si existe una actualización de la aplicación, pero si ésta ya está descargada, la comprobación de actualización caducará a los pocos segundos. En ese caso se usará la aplicación almacenada en caché. De esta forma, dada una conexión razonablemente rápida con el servidor, se usará la última versión de la aplicación, pero no se garantiza que sea así. La aplicación en cualquier caso se puede ejecutar sin conexión a la red.
- El elemento opcional *shortcut*: se usa para indicar las preferencias de la aplicación para integración en el escritorio. El atributo opcional *online="false"* define la preferencia de la aplicación de crear un acceso directo para lanzar la aplicación con conexión u *offline*. Los siguientes sub-elementos opcionales del elemento *shortcut* indican más preferencias de integración en escritorio:
  - Elemento *desktop*: indica la preferencia de crear un acceso directo en el escritorio del usuario.
  - Elemento *menu*: indica la preferencia de crear un valor de menú en los menús de inicio del usuario. Puede contener un atributo opcional, *submenu*, que indica donde se desea incluir el valor de menú.

El elemento *security* es necesario para solicitar acceso no restringido a los recursos del sistema. Al estar especificado el elemento *all-permisions* la aplicación tendrá acceso

completo a la máquina cliente y su red local. Para poder garantizar acceso completo todos los archivos *jar* deben estar firmados por el mismo firmante. El usuario será consultado para aceptar el certificado de firma cuando se lance la aplicación.

El elemento *resources* describe todos los recursos necesarios para la aplicación. Estos recursos pueden ser clases de Java, librerías nativas o propiedades del sistema, por ejemplo. En este caso se definen únicamente los siguientes sub-elementos:

- El elemento *java*: especifica qué versión del *Java Runtime Environment* soporta la aplicación. El atributo opcional *max-heap-size* indica el tamaño máximo de pila de memoria para la *JVM* que corra esta aplicación. De esta forma el problema del tamaño de los archivos con los que se trabaja queda bajo control.
- El elemento *jar*: especifica un archivo *jar* que forma parte de la aplicación mediante un atributo *href* que indica una URL relativa o completa. En este caso sólo se cuenta con un elemento de este tipo, que es el que contiene la aplicación desarrollada, *PFC_FIRMA.jar*.

Finalmente, el elemento *application-desc* indica que el archivo *JNLP* está lanzando una aplicación en lugar de un applet. Este elemento contiene un atributo, *main-class*, usado para especificar la clase que contiene el método *public static void main(String args[])* donde debe comenzar la ejecución.

Como conclusión, cabe decir que este es el método más versátil de despliegue de la aplicación de firma digital. Cambiando ligeramente el archivo *JNLP* se puede adecuar el despliegue a las necesidades puntuales de cada planteamiento, además de que permite controlar de forma sencilla el límite máximo de la pila de memoria de la *JVM*.

La capacidad de crear accesos directos, tanto en el menú de inicio como en el escritorio del usuario, para utilizar la versión en caché de la aplicación es una paso de gigante en facilidad de uso con respecto a las otras opciones de despliegue. De esta forma, se posibilita el uso de la aplicación por parte del cliente, independientemente de si tiene conectividad o no.

5.4 Generación de PDF a partir de formulario

En este apartado se explica con detalle la configuración del *PDF* de salida y los elementos del código fuente que pueden ser modificados para obtener distintos paneles de formulario y/o documentos *PDF* de salida.

En el curso del proceso de *Firma de Formulario*, la aplicación genera un documento *PDF* a partir de los distintos campos del formulario rellenados por el usuario. Esta conversión de formulario a documento *PDF* se realiza en dos pasos:

1) Conversión de panel de formulario a documento *XML*.
2) Conversión de documento *XML* a documento *PDF*

En el primer paso se realiza un barrido del panel del formulario, de arriba a abajo y de izquierda a derecha, buscando componentes interpretables por el conversor del paso 1). Según la configuración por defecto, los campos interpretables por el conversor son campos de texto, de una línea y multilínea, separadores y etiquetas de texto. Tan sólo se reconocerán los componentes del panel que sean campos de texto, y aquellas etiquetas a las que se les haya asignado un nombre mediante el método Java *<Component>.setName("nombre")*.

Los componentes válidos encontrados y ordenados según su posición relativa dentro del panel de *Firma de Formulario*, se usan para crear un documento *XML*, con elemento raíz *<formulario>*. Este documento *XML* sigue una sintaxis específica para posibilitar el segundo paso, la conversión de *XML* a *PDF*. Esta sintaxis viene regida por los requisitos de la librería *iText* y la configuración de etiquetas de conversión, *tagmap*, que se incluye en un archivo dentro del archivo *JAR* de la aplicación.

De esta forma, si se desea cambiar el formato del *PDF* de salida existen tres posibilidades: cambiar el panel de *Firma de Formulario* de la aplicación, cambiar el mapa de etiquetas con el que se define la conversión de *XML* a *PDF*, o una combinación de las dos anteriores.

El cambio del panel de *Firma de Formulario* implica modificar el código fuente de la aplicación para cambiar el panel a gusto del desarrollador. El único requisito para modificar el código fuente es que, si se añaden elementos que se desea incluir en la transformación del formulario a *PDF*, se utilice sobre ellos la sentencia *<Component>.setName("nombre")*. El siguiente ejemplo ilustra lo aquí explicado:

Supongamos que se quieren añadir al formulario los siguientes componentes: un campo de texto editable por el usuario, una etiqueta que lo defina y un separador para mejorar visualmente la interfaz. El componente de campo de texto editable será una instancia de alguna de las siguientes clases: *javax.swing.JTextField* para los campos de texto de una línea y *javax.swing.JTextPane* para los campos de texto editable multilínea. La etiqueta que acompañará al campo de texto será una instancia de la clase *javax.swing.JLabel*. El separador será una instancia de la clase *javax.swing.JSeparator*.

Para este ejemplo supondremos además que el campo de texto y la etiqueta se quieren incluir en el documento *PDF* final, pero no el separador. En consecuencia se deberá ejecutar la sentencia *<Component>.setName("<nombre>")* sobre el campo de texto y sobre la etiqueta.

```
...
//campoTexto debe aparecer en el PDF de salida
campoTexto = new javax.swing.JTextField();
campoTexto.setName("Nombre de Campo de Texto");
...
//etiqueta debe aparecer en el PDF de salida
etiqueta = new javax.swing.JLabel();
etiqueta.setName(“Nombre de Etiqueta”);
...
separador = new javax.swing.JSeparator();
```

Figura 5-5: Código *java* de ejemplo para modificar el *PDF* de salida

En la figura 5-5 se muestra una simple instanciación de las tres clases utilizadas y cómo se debe definir el nombre de los componentes que se desea que aparezcan en el *PDF* de salida. En el caso de las etiquetas únicamente el texto entrecomillado en la sentencia de *setName* aparecerá en el *PDF* de salida. En el caso de los campos de texto editable, en el *PDF* de salida se mostrará el texto entrecomillado en la sentencia *setName* del componente seguido de dos puntos y el texto introducido por el usuario de la aplicación. En el caso de los separadores, el texto entrecomillado se omite, ya que sirve únicamente para indicar a la aplicación que se desea insertar un salto de línea.

El cambio del mapa de etiquetas para su uso por la librería *iText* se ilustrará con el mapa usado por defecto en la aplicación desarrollada y el documento *XML* generado a partir del contenido por defecto del formulario. A continuación se muestra el documento *XML* intermedio obtenido a partir del panel del formulario.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<formulario>
  <textField>NIA: 1000XXXXX</textField>
  <textField>TelÃ©fono: 9XXXXXXXX</textField>
  <textField>MÃ³vil: 6XXXXXXXX</textField>
  <separator/>
  <textField>DirecciÃ³n de contacto</textField>
  <textField>VÃ-a: Calle, nÃºmero, piso, escalera ...</textField>
  <textField>PoblaciÃ³n: poblaciÃ³n</textField>
  <textField>Provincia: provincia</textField>
  <textField>CP: XXXXX</textField>
  <separator/>
  <textField>Asunto: Motivo de la solicitud</textField>
</formulario>
```

Figura 5-6: Documento *XML* intermedio generado a partir del formulario

Se observa en la figura 5-6 que las vocales tildadas no aparecen correctamente; sin embargo, se muestran correctamente en el *PDF* final. Esto no supone ningún problema para el correcto funcionamiento de la aplicación. El documento *XML* recoge todos los elementos que son campos de texto editable por el usuario, los separadores y las etiquetas que tienen un nombre definido mediante *<Component>.setName(“nombre”)*. Se puede comprobar que la lista de componentes del formulario sigue el orden que se obtiene de ordenarlos según su disposición en el panel de arriba a abajo y de izquierda a derecha, como se ha mencionado anteriormente.

A continuación se muestra el mapa de etiquetas usado por defecto en la conversión del documento *XML* intermedio a documento *PDF*.

```
<tagmap>
      <tag name="iText" alias="formulario" />
      <tag name="newline" alias="separator" />
      <tag name="paragraph" alias="textField">
            <attribute name="leading" value="14" />
            <attribute name="align" value="Left" />
      </tag>
</tagmap>
```

Figura 5-7: *tagmap* por defecto para la conversión de *XML* a *PDF*

El mapa de etiquetas se utiliza al realizar un análisis sintáctico del documento *XML* a partir del cual se va a construir un documento *PDF*; le indica a las clases de la librería *iText* qué elemento debe introducir en el documento *PDF* según se encuentre unas u otras etiquetas en el documento *XML*.

La estructura del mapa de etiquetas, o *tagmap*, sigue una sintaxis *XML*, con el elemento *tagmap* como raíz. Consta de distintos elementos de tipo *tag* con dos atributos obligatorios, y la posibilidad de tener sub-elementos en algunos casos. Cada elemento *tag* cuenta con dos atributos:

- Atributo *name*: indica qué tipo definido por la librería debe crearse si se encuentra un elemento del alias indicado en el documento *XML* que se está analizando.
- Atributo *alias*: indica el elemento que hará que se aplique la etiqueta que lo contiene.

Es decir, cada elemento *tag* relaciona un elemento del *XML* analizado, definido por el atributo *alias*, con un componente que se debe crear en el documento *PDF* de salida, definido por la etiqueta *name*. De esta forma, si la aplicación encuentra un elemento de tipo *separator* en el *XML* analizado, se introducirá un elemento de tipo *newline* en el documento *PDF* que se creará.

El caso de la etiqueta con atributo *name="iText"* es especial, ya que indica qué elemento es el que hace que comience el proceso de creación del documento *PDF*. En el documento *XML* generado a partir del formulario, el elemento raíz *formulario* es el que se relaciona mediante un elemento *tag* con la creación del *PDF*.

Teniendo presente lo expuesto anteriormente, es fácil observar en el *tagmap* que los separadores generan un salto de línea y los campos de texto generan un párrafo. El caso del párrafo, de atributo *name="paragraph"*, tiene dos sub-elementos más, que indican características especiales de formato para el párrafo que se crea (en este caso el alineamiento y el *leading*).

El sub-elemento a*lign* indica el alineamiento que tendrá el párrafo en el *PDF* de salida. Éste puede ser *Left*, *Right*, *Center*, *Justified*, es decir, alineado a la izquierda, a la derecha, centrado o justificado (ajustado a ambos márgenes, tanto a la derecha como a

la izquierda). El sub-elemento *leading* indica el espacio entre líneas de un mismo párrafo.

Por tanto, en caso de querer modificar el documento *PDF* generado, se debe cambiar el código de las clases que definen el formulario y que generan el *XML* a partir del panel del formulario. Se deberá adecuar, además, el mapeado de etiquetas incluido en el código fuente a los elementos nuevos que se quieran incluir. Para una documentación más amplia respecto a los mapas de etiquetas se recomienda consultar [5], [6] y [7]. Las clases del código que se deben modificar serán:

- *PFC.appletFirma.TabbedPanel.java*: define el formulario de la aplicación.
- *PFC.appletFirma.Form2xml.java*: define la forma en que se interpreta el panel de formulario para generar el *XML* a partir del que se construirá el documento *PDF*.

Estas clases se explican más a fondo en la estructura del código de la aplicación del capítulo 6 de este documento.

```
1  /**
2   *
3   * @author luis
4   */
5  public class HolaMundo {
6      public static void main(String args[]){
7          System.out.println("Hola, PFC!");
8      }
9  }
10
```

6 Estructura del Código Fuente

Si observamos el contenido del archivo *JAR* que reúne el código de la aplicación observamos una división en cuatro paquetes:

- Paquete *PFC*: este paquete contiene todo el código Java desarrollado por el autor de este documento. Contiene todo el núcleo de la aplicación.
- Paquete *com*: el paquete *com* contiene el código fuente de la librería *iText*, necesaria para la creación, manipulación y firma interna de documentos *PDF*.
- Paquete *org*: contiene el código fuente de las librerías *Commons Codec*, de Apache, y *Bouncy Castle*. La librería *Commons Codec* se usa para codificar y decodificar los archivos a Base64, para trabajar en dicho formato en los documentos *XML*. La librería *Bouncy Castle* es un requisito necesario para el correcto funcionamiento de la librería *iText*.
- Paquete *resources*: este paquete no contiene ninguna clase Java. Únicamente contiene el archivo *tagmap.xml* utilizado para realizar la construcción de un documento *PDF* a partir del contenido del panel de formulario, tal y como se explica en el apartado 5.4 de este documento.

La decisión de incluir los archivos fuente de las librerías necesarias en vez de acompañar el *JAR* de la aplicación de los *JAR* necesarios se debe a criterios de sencillez y comodidad. Resulta más sencillo realizar el despliegue mediante *JNLP*, ya que no hace falta firmar todos los *JAR* necesarios con el mismo certificado que el *JAR* de la aplicación. Esta ventaja de sencillez se paga en una desventaja de tamaño del *JAR* de la aplicación desarrollada. Sin embargo ese tamaño no influye en el tamaño total que el

usuario debe descargar para ejecutar la aplicación, ya que el resto de *JARs* deberían ser descargados igualmente.

Este capítulo se centra en la estructura del paquete *PFC*, que es el desarrollado por el autor de este documento. Para consultar la documentación de las librerías utilizadas se recomienda visitar [5], [6], [7] para iText, [8] para Bouncycastle y [9] para la codificación en Base64.

El código fuente de la herramienta de validación de certificados en PKI con TII, contenido en el paquete *PFC*, está estructurado en distintos subpaquetes, cada uno de ellos orientado a cubrir unas necesidades específicas de la ejecución de la aplicación. Los paquetes de que consta el código desarrollado para la aplicación se definen en los siguientes apartados.

6.1 Paquete PFC.appletFirma

El paquete *PFC.appletFirma* contiene las clases que definen la lógica que coordina el correcto funcionamiento de la aplicación y toda la interfaz gráfica con la que interactúa el usuario. La clase principal de la aplicación, que contiene un método *public static void main(String args[])* y que hereda de la clase *JApplet*, se encuentra dentro de este paquete. A continuación se explican las clases contenidas dentro de este paquete.

6.1.1 PFC.appletFirma.AppletFirma

Esta clase es el punto de entrada a la aplicación. Contiene un método *public static void main(String args[])*, hereda de la clase *javax.swing.JApplet* e implementa el método *INIT()* con lo que puede ser lanzada de múltiples maneras.

Funcionalmente no realiza ninguna tarea salvo la de crear un s*ingleton* de la clase *PFC.appletFirma.AppletHandler*, que es la encargada de manejar la lógica de la aplicación.

6.1.2 PFC.appletFirma.AppletHandler

Esta clase es el "cerebro" de la aplicación, ya que coordina toda la lógica del proceso de firma. La clase *AppletHandler* es la encargada de coordinar el funcionamiento y la correcta disposición de todos los procesos de la aplicación, tanto los de firma como los de validación.

También coordina y ordena los elementos visuales mostrados en la interfaz gráfica que se le presenta al usuario de la aplicación.

6.1.3 PFC.appletFirma.CertSelectionWindow

La clase *PFC.appletFirma.CertSelectionWindow* es la encargada de crear la ventana de selección de certificados de la TII (figura 4-11) en la que el usuario debe elegir uno de los certificados mostrados para utilizarlo en el proceso de firma. Incluye todos los métodos necesarios para la gestión de eventos generados por acciones del usuario.

Utiliza una instancia de la clase *PFC.cripto.JTreeCertificateBuilder* (ver 6.2.2) para generar el árbol de certificados que muestra al usuario.

6.1.4 PFC.appletFirma.PasswordDialog

PFC.appletFirma.PasswordDialog es la clase encargada de generar la ventana en la que se solicita al usuario su PIN para acceder al almacén de certificados de su TII. Contiene la gestión necesaria de eventos generados por el usuario.

6.1.5 PFC.appletFirma.SaveSelectionWindow

PFC.appletFirma.SaveSelectionWindow genera la ventana que se muestra al usuario para solicitarle que seleccione un directorio válido de destino en el que salvar el archivo de salida del proceso que se está ejecutando (figuras 4-4, 4-17 y 4-31). La ventana mostrada, tanto en el caso de firma como en el de validación, se genera con esta única clase, aunque cambia ligeramente su aspecto si se trata de uno u otro caso. Contiene toda la gestión de eventos necesaria. En caso de que se desee cambiar el mensaje informativo que se le muestra al usuario en esa ventana basta con cambiar dos *Strings* en el código, el *String helpTextFirma* y el *String helpTextValidación*.

6.1.6 PFC.appletFirma.TabbedPanel

Se trata del componente principal de la interfaz gráfica de la aplicación. Dependiendo de cómo se haya iniciado la aplicación este panel se mostrará enventanado (iniciado como una aplicación vía método *main*) o como un panel dentro de un contenedor adecuado (iniciado como applet vía método *init*), tales como una página web o un visor de applets.

Tal como se ha descrito en apartados anteriores, consta de tres paneles seleccionables mediante etiquetas, donde cada uno de ellos muestra al usuario los elementos necesarios para cada una de las tres funciones de la aplicación, *Firma de Archivo*, *Firma de Formulario* y *Validación de Archivo*.

Cuando se desee adecuar el panel de *Firma de Formulario* a las necesidades propias del proyecto en el que se integre esta aplicación se deberá cambiar el código de esta clase. Únicamente se deben seguir las directrices indicadas en el apartado 5.4 de este documento.

6.2 PFC.cripto

En el paquete *PFC.cripto* se incluyen todas las clases que tienen que ver con el acceso y manejo de certificados y con la generación y validación de firmas digitales.

6.2.1 PFC.cripto.TII

Clase creada para facilitar el acceso al almacén de certificados de la TII utilizada. Gestiona todo el acceso al almacén de certificados y provee de métodos para comprobar si la TII está insertada.

6.2.2 PFC.cripto.JTreeCertificateBuilder

Clase encargada construir el árbol de certificados que se muestra en la ventana de selección de certificados (ver apartado *6.1.3*). Los ordena y agrupa en forma de árbol con la Autoridad de Certificación emisora como nodo raíz.

6.2.3 PFC.cripto.SignatureThread

Hereda de la clase *java.lang.Thread*. Una instancia de esta clase será lanzada cada vez que se ejecute un proceso de firma, tanto de archivo como de formulario. Contiene toda la lógica del proceso de firma.

Si el objetivo por el que ha lanzado este hilo es el de firmar un formulario, entonces se hará uso de la funcionalidad aportada por la librería *iText* para firmar de forma interna el documento *PDF* creado a partir del formulario.

6.2.4 PFC.cripto.Validator

Clase que se ocupa del proceso de validación de archivos. En caso de que el formato del archivo analizado sea correcto y supere el proceso de validación se extraerá el archivo original contenido en el archivo de firma. También mostrará en pantalla mediante un aviso informativo que la validación fue positiva junto con el nombre y el identificador único de usuario del firmante (ver figura 4-36).

6.2.5 PFC.cripto.X509CertificateHandler

La clase *PFC.cripto.x509CertificateHandler* provee de métodos que facilitan la interacción con los certificados de tipo *java.security.cert.X509Certificate*. Esta clase se usa por el resto de clases del paquete al que pertenece, *PFC.cripto*.

6.3 PFC.util

En este paquete se han incluido todas las clases que por su función no encajan dentro de las definiciones de los otros dos paquetes pero que son necesarias para llevar a cabo distintas tareas no tan específicas.

6.3.1 PFC.util.FileTree

Clase tomada parcialmente de los ejemplos de *Core SWING Advanced Programming*, de Kim Topley (Ed. Prentice Hall). Hereda de la clase *javax.swing.JTree*. Sirve para obtener una representación del sistema de archivos de la máquina en que se ejecuta la aplicación en forma de árbol. Utilizado por *PFC.appletFirma.TabbedPanel* en los paneles de *Firma de Archivo* y *Validación de Archivo*, así como por *PFC.appletFirma.SaveSelectionWindow* para mostrar el contenido de la unidad seleccionada en cada caso.

6.3.2 PFC.util.Form2xml

Clase que provee de los métodos necesarios para transformar el panel del formulario en un documento *XML* de tipo *org.w3c.dom.Document*. Tal como se ha explicado previamente, el contenido del panel del formulario se utiliza para crear un *org.w3c.dom.Document* que más adelante se interpretará, haciendo uso de las librerías de *iText* junto con un *tagmap* almacenado en la carpeta *resources*, para crear un archivo *PDF*.

El ordenamiento de los componentes según su posición relativa dentro del panel del formulario se realiza haciendo uso del comparador *PFC.util.PointComparator* descrito en el apartado *6.3.4*.

6.3.3 PFC.util.Utils

En esta clase se incluyen varios métodos estáticos que no tienen especial cabida en otras clases, pero que sin embargo es útil tener a disposición desde cualquier punto de la aplicación.

6.3.4 PFC.util.PointComparator

Comparador necesario para determinar el orden en el que se van a incluir los campos de un formulario en el *PDF* que se crea a partir del panel que contiene el formulario. Se tiene en cuenta el atributo *java.awt.Point* de cada componente del formulario para la comparación entre distintos componentes. Este *Point* indica la esquina superior izquierda de cada componente. El resultado de la comparación entre

dos puntos *Point* será -1 si el primer punto está más arriba o a misma altura pero más a la izquierda que el otro. La comparación será 0 si ambos puntos tienen las mismas coordenadas. La comparación será 1 si el primer punto está más abajo o a la misma altura pero más a la derecha que el segundo.

En caso de querer establecer otro criterio de comparación para obtener un formato *PDF* de salida distinto, se debe cambiar el código de este comparador.

6.3.5 PFC.util.Xml2pdf

Esta clase se encarga de transformar un documento *XML* de tipo *org.w3c.dom.Document* (en la práctica obtenido de la interpretación del panel de firma de formulario) en un documento *PDF* contenido en un array de bytes. De esta forma, se podrá acceder más adelante en el proceso de firma a dicho *PDF*, para incluir una firma en él sin necesidad de volcarlo a disco. Utiliza de forma intensiva clases de la librería *iText*.

6.3.6 PFC.util.XmlDocHelper

La clase *PFC.util.XmlDocHelper* facilita la construcción de objetos de tipo org.w3c.dom.Document, que no son otra cosa que documentos *XML*. Provee métodos sencillos e intuitivos para construir este tipo de objetos de forma que, a la hora de firmarlos, sean fácilmente maleables para incluir los campos necesarios de firma.

7 Conclusiones y Líneas Futuras

El objetivo de este capítulo es mostrar las conclusiones a las que se ha llegado tras la realización del proyecto desarrollado en el ámbito de *Herramientas de validación de certificados en PKI con tarjeta inteligente*, y a su vez se indica cuáles han sido los puntos críticos de diseño en dicho desarrollo. También se exploran algunas propuestas de posibles mejoras aplicables en un futuro para complementar y/o ampliar la funcionalidad y alcance de este proyecto.

7.1 Conclusiones

El objetivo de este proyecto ha sido el desarrollo de una herramienta informática que, haciendo uso de una PKI existente, provea de capacidad de firma y validación mediante certificados de usuario almacenados en una TII de dicha PKI. Dicha herramienta consta del motor lógico de la aplicación y de una interfaz gráfica que facilita la interacción con la TII durante el uso de sus funciones de firma o validación.

La posibilidad de utilizar TIIs de PKIs ya existentes para firma y validación supone un gran ahorro debido al elevado coste que supone implantar una nueva infraestructura desde cero, que provea de todos los elementos necesarios para completar esas funciones. Sin embargo presenta, como es natural, ciertas dificultades.

En el desarrollo de este proyecto, la mayor dificultad encontrada ha sido debida a la falta de información provista por parte de las entidades responsables de la emisión de

las TII utilizadas. Esta escasez de información técnica ha supuesto retrasos en el plan de desarrollo, además de múltiples modificaciones en el comportamiento de la herramienta desarrollada, necesarias para adecuar la herramienta al comportamiento de las TIIs utilizadas. Una vez solventado el problema de la información técnica incompleta a base del más puro estilo de prueba/error, se ha podido acceder adecuadamente a los almacenes de certificados de las TIIs. Ha sido necesario modificar el acceso y el uso de los almacenes de certificados de las TII para tener en cuenta el comportamiento de las TII en determinadas situaciones, no especificadas por el proveedor. El principal problema encontrado fue el de que las TIIs no cierran correctamente el canal de comunicación con el ordenador en el que se ejecuta la aplicación una vez finalizada la interacción con el usuario una vez introducido el PIN correctamente, a pesar de cerrar por software la comunicación con ellas. De esta forma la información básica del almacén de certificados sigue siendo visible, e incluso puede ser utilizada si no se extrae la tarjeta del lector. Esto es un gran inconveniente si se pretende solicitar el PIN de la TII cada vez que se pretenda realizar una operación de firma.

Tras conseguir acceder de forma correcta al almacén de certificados de las TII, se pasa al diseño del formato de salida. En este aspecto, una vez decidido el formato que se usará, la especificación XAdES-BES, el único problema es el de conformar la salida de la aplicación a dicha especificación. Dicha conformancia no es difícil; únicamente presenta la necesidad de elegir qué información incluir en la salida de la aplicación, ya que la especificación XAdES-BES es muy flexible respecto a la información adicional que se desee incluir.

El siguiente paso en el desarrollo de la herramienta es el del diseño de la interfaz gráfica que se le presenta al usuario que interactúa con ella. El objetivo de esta interfaz es el de proveer al usuario de un entorno sencillo e intuitivo con el que manejar las funciones de la aplicación. Como se ha indicado anteriormente, ha sido necesario adecuar el modo en que se accede y se utilizan los certificados de usuario almacenados en las TIIs por su comportamiento en determinadas condiciones. Es por ello que dicho comportamiento se ha tenido en cuenta en el desarrollo de la interfaz gráfica de la herramienta. Sin embargo, una vez conocido dicho comportamiento desde las primeras fases del desarrollo de la herramienta, esta adecuación de la interfaz gráfica no ha supuesto mayor problema.

En último lugar se ha tratado de favorecer la adopción y despliegue de la herramienta desarrollada mediante la inclusión de distintos sistemas de despliegue. Las diferentes maneras en que la herramienta desarrollada puede ser desplegada aportan una flexibilidad muy necesaria para adecuarse a las distintas necesidades de posibles usuarios, tal y como se muestra en el apartado 5.3 de este documento.

De esta forma el desarrollo final de la herramienta permite el trabajo con TIIs de PKIs ya existentes, aportando suficiente flexibilidad en el diseño como para permitir su implantación en distintos ámbitos de trabajo. No está orientada específicamente a un entorno concreto, y se ha dejado la puerta abierta a futuras modificaciones de la herramienta, como se indica en el apartado 7.2 de este documento.

7.2 Decisiones de Diseño

A lo largo del presente documento se han comentado varios criterios de diseño empleados en el desarrollo de la herramienta de validación. En este apartado se comentan los puntos más importantes en cuanto a decisiones de diseño se refiere.

La primera decisión de diseño de peso fue la elección del lenguaje de programación sobre el que desarrollar la aplicación. Finalmente se optó por utilizar *Java*, ya que, además de que prácticamente cualquier ordenador actual tiene una *JVM* instalada, su soporte para trabajo con tarjetas inteligentes y criptografía ha mejorado mucho. Actualmente siguen existiendo ciertos problemas con la comunicación con las TIIs, pero son en su mayoría debidos a comportamientos no esperados por parte de éstas, como se ha indicado en el apartado 7.1. Otra ventaja de elegir *Java* como lenguaje de programación para el desarrollo de la aplicación es la posibilidad de utilizar algunas de las abundantes librerías públicas disponibles, que simplifican mucho la tarea del programador.

Una vez elegido *Java* como lenguaje de programación a utilizar, se planteó la necesidad de elegir entre distintas opciones para generar los componentes de que forma parte la interfaz gráfica de usuario. Se optó por utilizar elementos del paquete *javax.swing*, que presenta mayor versatilidad y amplía la funcionalidad del paquete *java.awt*, más básico.

La siguiente decisión de diseño tiene el panel del formulario como protagonista. Al ser la herramienta desarrollada una base de la que partir para otros proyectos e implantaciones, y no una herramienta final con fines específicos, se debía facilitar el cambio de dicho formulario de forma relativamente sencilla. Por ello se optó por incluir en el panel de formulario de la GUI un formulario genérico a modo de ejemplo, como se muestra en la figura 4-15, a partir del cual se generaría un documento *PDF* siguiendo unas pautas de construcción definidas.

Para que futuros desarrolladores que utilicen esta herramienta como base para sus proyectos no tengan que sumergirse a fondo en el código de la aplicación, se programó dicho panel de formulario para que se pudiese modificar completamente el *PDF* de salida. Tan sólo es necesario eliminar, añadir o mover los elementos que lo conforman y seguir determinadas normas, expuestas en el apartado 5.4 del presente documento. De esta manera se permite la adecuación del *PDF* de salida modificando únicamente el panel de formulario (*6.1.6 PFC.appletFirma.TabbedPanel*), algo que se puede hacer de forma enteramente gráfica mediante algún entorno IDE de desarrollo que lo soporte, como NetBeans (el utilizado para el desarrollo de la herramienta).

Respecto al formato de salida de los procesos de firma, hubo que decidir entre las posibilidades de definir un formato único y nuevo para la aplicación desarrollada u optar por uno existente y reconocido. La opción lógica fue la segunda, y dentro de las distintas posibilidades existentes, de las que de las más notables se hace mención en el apartado 2.5, *Formatos de Datos Criptográficos*, se eligió XAdES-BES. El hecho de optar por un formato reconocido radica en la fiabilidad ya probada de dichos estándares

y en la posibilidad de una futura interacción entre la herramienta aquí desarrollada y otras aplicaciones que hagan uso de dichos formatos, más extendidos. La elección de XAdES-BES como estándar de salida se basó en el hecho de que es la forma más básica de firma digital aceptada legalmente según la directiva europea de firma electrónica; es el mínimo formato de firma generable por un firmante que quiera crear una firma reconocida legalmente. De esta forma se dejan abiertas futuras vías de trabajo en torno a la firma digital legalmente reconocida. Existen estándares más completos que XAdES-BES dentro de la familia XAdES, de la cual XAdES-BES es el más básico; sin embargo, para no alejarnos de lo generalista de la herramienta en su estado actual, se optó por utilizar este estándar. Queda a discreción de futuros desarrolladores que adopten esta herramienta como base para futuros trabajos ampliar el formato de salida a otro XAdES más avanzado.

7.3 Líneas Futuras

Existen múltiples mejoras posibles con las que completar la herramienta desarrollada. Una mejora posible sería añadir la opción de elegir en qué formato criptográfico se desea almacenar la salida de los procesos de firma. De esta forma la aplicación ganaría en versatilidad a la hora de complementar otras herramientas o interactuar con otras aplicaciones. La opción podría presentarse al usuario mediante un nuevo diálogo o con una lista desplegable desde el propio panel de la GUI, sin selección múltiple.

Otra vía de mejora que queda abierta es la posibilidad de adaptar la herramienta para ser lanzada sin modo gráfico. De esta forma, otras aplicaciones podrían integrarla en sus procesos de forma invisible para el usuario, salvo por el diálogo necesario de petición de PIN para la TII en los casos que sea necesario, y por la ventana de selección de certificado. En función de las especificaciones del entorno en el que se pretendiera integrar, incluso estas dos pequeñas muestras de interfaz gráfica podrían evitarse.

También cabe la posibilidad de adaptar la herramienta para aceptar como entrada para su firma un formulario web visionado en una ventana de navegador. De esta forma se tendría mayor flexibilidad aún a la hora de adaptar el *PDF* de salida a los requisitos de cada implantación particular de la herramienta. Esta mejora no supone una gran dificultad si se tiene en cuenta que ya se han incluido métodos con los que interpretar el panel de formulario mediante *tagmaps* para construir un documento *PDF* a partir de él. Bastaría con crear un nuevo "intérprete" para la página web similar a lo realizado con dichos métodos del panel de formulario de la GUI.

Actualmente la aplicación desarrollada está programada con un planteamiento de *applet*: se ejecuta o bien de forma local en la máquina del usuario o se ejecuta a partir de una página web cargada en una ventana del navegador de la máquina del usuario. No existe comunicación entre la aplicación desarrollada y la máquina que la aloja más allá de la carga inicial; es fundamentalmente una aplicación *offline*. Así, todos los archivos manejados por la aplicación (los archivos que toma como fuente para firmarlos, los

archivos de salida firmados y los archivos extraídos en el proceso de validación) serán parte del sistema de archivos local del usuario que ejecuta la herramienta. Una variación importante de esta herramienta podría ser añadir funcionalidad *online* y que los archivos de firma generados por la herramienta quedaran directamente almacenados en la máquina servidora. De esta forma se podrían explotar nuevos enfoques de uso de la herramienta, tales como registros de acceso, autenticación, etc. Esta nueva orientación de los objetivos de la herramienta supondría un cambio bastante profundo, sin duda el más complejo de los sugeridos en este apartado. Sin embargo su realización manteniendo grandes partes del código de la aplicación inalteradas sigue siendo relativamente asequible, y, en función de necesidades futuras, puede ser un cambio de gran importancia.

8 Presupuesto

En este capítulo se procede a realizar una proyección económica en base a la cuantificación del esfuerzo realizado y del material utilizado en la elaboración de este proyecto de fin de carrera.

8.1 Tareas del Proyecto

El proyecto ha constado de las siguientes tareas de desarrollo:

- Tarea 1: Estudio previo y documentación. En esta etapa se ha realizado un estudio previo del estado del arte de los formatos criptográficos y las infraestructuras de clave pública, las aplicaciones actuales de las tarjetas inteligentes de identificación de PKI existentes.

- Tarea 2: Definición de requisitos y especificaciones. En esta tarea se han definido los criterios de diseño de los formatos de salida, la elección del lenguaje de programación a utilizar, y la concreción del alcance de los objetivos a cumplir. Todo ello se ha realizado en base a los resultados de la fase de estudio previo y documentación, la tarea 1.

- Tarea 3: Implementación básica. En esta tarea se ha implementado la comunicación básica con las TIIs y el acceso a sus almacenes de certificados. Como se indica en el apartado 7.1 del presente documento, en

esta fase se encontraron determinados problemas que ralentizaron el desarrollo de la herramienta.

- Tarea 4: Implementación de la GUI. En esta tarea se ha desarrollado la interfaz de usuario en torno a la implementación obtenida en la Tarea 3.

- Tarea 5: Implementación de conformidad con XAdES-BES. En esta tarea se ha adecuado el formato de salida en los procesos de firma a las especificaciones del estándar XAdES-BES.

- Tarea 6: Implementación de las distintas formas de despliegue. Durante esta fase del desarrollo se han realizado las modificaciones necesarias en el código obtenido hasta este punto para permitir la flexibilidad a la hora de desplegar la herramienta.

- Tarea 7: Desarrollo de métodos de creación de documento *PDF*. En esta tarea se han implementado los métodos mediante los que se facilita la creación y personalización de documentos *PDF* a partir del panel de formulario. Es también en esta etapa en la que se incluye con éxito la funcionalidad de inserción de firma dentro del propio documento *PDF* generado.

- Tarea 8: Depuración y pruebas. Proceso de testeo intensivo para eliminación de fallos y bloqueos.

- Tarea 9: Trabajo de documentación del proyecto. Esta tarea comprende la realización del presente documento.

A continuación se muestra una tabla con la duración estimada, en horas, de cada una de las tareas aquí descritas.

Tabla i: Carga horaria de las tareas realizadas

Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5	Tarea 6	Tarea 7	Tarea 8	Tarea 9
100	60	220	120	60	60	90	80	190

8.2 Coste de Personal

En estos gastos se incluyen todos aquellos costes relativos a los recursos humanos requeridos en el proyecto en concepto de mano de obra. No todas las tareas realizadas requieren personal con cualificación de ingeniero de telecomunicaciones. Las tareas 1 y 9 pueden, al menos en parte, ser realizadas por personal administrativo. La suma total

de horas de trabajo de uno y otro perfil quedan aproximadamente como se muestra en la siguiente tabla:

Tabla ii: Carga de trabajo de los distintos perfiles

Perfil	Tareas	Horas
Ingeniero de Telecomunicación	1*,2,3,4,5,6,7,8 y 9*	840
Administrativo	1* y 9*	160

Las tareas marcadas en la tabla anterior con un '*' indican que el determinado perfil que participa tan sólo es necesario para una parte de dicha tarea.

Para determinar los sueldos de los distintos perfiles (ingeniero de telecomunicación y administrativo) se han consultado distintas fuentes y ofertas reales en el momento de la realización de este documento. Además cabe indicar que según el Régimen General de la Seguridad Social el grupo de cotización de un ingeniero es el grupo 1 y el de un administrativo el grupo 7. Los salarios brutos anuales considerados quedan por encima de las bases mínimas cotizables que marca la Seguridad Social para cada uno de los perfiles. Se ha considerado una jornada laboral de 8 horas/día, 21 días laborales/mes y 225 jornadas laborables/año. A partir de estos datos se obtienen los siguientes costes:

Tabla iii: Costes salariales por perfil laboral

Costes salariales			
	Concepto	**Grupo 1**	**Grupo 7**
A	Salario bruto anual	24.000,00 €	19.000,00 €
B	Contingencias comunes (23'6%)	5.664,00 €	4.484,00 €
C	Desempleo, F.G.S. y Formación profesional (8,7%)	2.088,00 €	1.653,00 €
E	Coste total de seguridad social (B+C)	7.752,00 €	6.137,00 €
F	Coste salarial anual (A+E)	31.752,00 €	25.137,00 €
G	Coste salarial por hora (F/[225jornadas * 8horas])	17,96 €	13,97 €
H	Número de horas	840	160
I	**Coste total por perfil (G*H)**	**15.085,79** €	**2.235,20** €

En la tabla *iii* se calcula el coste de las contingencias comunes (B) y de desempleo, F.G.S. y F.P. (C) a partir del salario bruto anual considerado para cada perfil. La suma de ambos conceptos es el coste total de seguridad social (E). El coste salarial anual para cada uno de los perfiles (F) se obtiene de la suma del salario bruto anual (A) y el coste de seguridad social anual (E). A partir del coste salarial anual se obtiene un coste salarial por hora de trabajo (G), dividiendo aquel entre el número de horas por jornada (8 horas) y el número de jornadas laborales por año (225 jornadas). Finalmente se calcula el coste total para cada uno de los perfiles multiplicando el coste salarial por

hora estimado (G) por el número de trabajo correspondientes con cada uno de los perfiles.

De esta forma se obtiene el siguiente coste total de personal:

Tabla iv: Coste total de personal

Perfil	Total (€)
Ingeniero de Telecomunicación	15.085,79
Administrativo	2.235,20
Coste Total de Personal	**17.320,99**

8.3 Coste de Material

En este apartado se desglosan los costes correspondientes al material utilizado para la implementación del proyecto.

Tabla v: Coste de material

Equipo	Total (€)
Ordenador personal	800
Licencia Microsoft Windows XP	90
5x TII genérica	75
Lector de Tarjetas Inteligentes	20
Coste Total de Equipo	985

8.4 Coste Total del Proyecto

En la tabla *vi* se resume el coste total del proyecto. La suma del coste de personal y del coste de material necesario para la realización del proyecto se utiliza como base imponible para calcular el gravamen por I.V.A.

Tabla vi: Coste total del proyecto

Coste	Total (€)
Coste de Personal	17.320,99
Coste de Material	985,00
Base Imponible	18.305,99
I.V.A. (16%)	2.928,96
Coste Total del Proyecto	21.234,95

Referencias y Bibliografía

Referencias

[1] Sintaxis de XML-Dsig
http://www.w3.org/TR/xmldsig-core/

[2] Directiva 1999/93/CE del Parlamento Europeo y del Consejo, de 13 de Diciembre de 1999, por la que se establece un marco comunitario para la firma electrónica
http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2000:013:0012:0020:ES:PDF

[3] Sintaxis de Java Network Launching Protocol:
http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/syntax.html

[4] Java Specification Request JSR-00056 Java Network Launching Protocol and API:
http://jcp.org/aboutJava/communityprocess/mrel/jsr056/index3.html

[5] iText API:
http://www.1t3xt.info/api/index.html

[6] iText online tutorial:
http://itextdocs.lowagie.com/tutorial/

[7] iText examples:
http://www.1t3xt.info/examples/browse/

[8] BouncyCastle:
http://www.bouncycastle.org/

[9] Apache Commons Codec:
http://commons.apache.org/codec/

[10] Esquema de funcionamiento de una PKI, Wikipedia
http://en.wikipedia.org/wiki/File:Public-Key-Infrastructure.svg

Bibliografía

- ITU-T Recommendation X.509 | ISO/IEC 9594-8
- RFC 3280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- RFC 3852 Cryptographic Message Syntax (CMS)
- RFC 2315 PKCS#7
- ETSI TS 101 903 XML Advanced Electronic Signatures (XAdES).
- http://en.wikipedia.org

Glosario de Términos

API - *Application Programming Interface*, interfaz de programación de aplicaciones
BER - *Basic Encryption Rules*, reglas básicas de codificación
CA - *Certification Authority*, autoridad de certificación
CMS - *Cryptographic Message Syntax*, sintaxis de mensaje criptográfico
CRL - *Certificate Revocation List*, lista de revocación de certificados
CSP - *Cryptographic Service Provider*, proveedor de servicios criptográficos
DER - *Distinguished Encryption Rules*, reglas avanzadas de codificación
DES - *Data Encryption Standard*, estándar de codificación de datos
DN - *Distinguished Name*, nombre distintivo
DPC - Declaración de prácticas y políticas de certificación
GUI - *Graphical User Interface*, interfaz gráfica de usuario
IETF - *Internet Engineering Task Force*
ISO/IEC - *International Organization for Standardization/ International Electrotechnical Commission*
ITU-T - *International Telecommunication Union, Telecommunication Standardization Sector*
TII – Tarjeta Inteligente de Identificación
OCSP - *Online Certificate Status Protocol*, protocolo en línea de estado de certificados
PAA - *Policy Approval Authority*, autoridad de aprobación de políticas
PGP - *Pretty Good Privacy*
PKCS - *Public Key Cryptography Standard*, estándar criptográfico de clave pública
PKI - *Public Key Infrastructure*, infraestructura de clave pública
RA - *Registration Authority*, autoridad de registro
RFID - *Radio-frequency Identification*
RSA - Algoritmo de cifrado de clave pública, creado por Rivest, Shamir y Adleman
SAML - *Security Assertion Markup Language*
SML - *Service Modeling Language*
SOAP - *Simple Object Access Protocol*
SSL - *Secure Socket Layer*
TSA - *Time Stamping Authority*, autoridad de sellado de tiempo
URI - *Uniform Resource Identifier*, identificador uniforme de recurso, definido en RFC 2396
VA - *Validation Authority*, autoridad de validación

XAdES - *XML Advanced Electronic Signatures*

XAdES-BES - *XAdES Basic Electronic Signature*, firma electrónica XAdES básica

XML - *Extensible Markup Language*

XML-Dsig - *XML Signature*