



UNIVERSIDAD POLITÉCNICA DE MADRID
FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA
Soluciones PKI basadas en Cryptlib

AUTOR: Daniel Plaza Espí

TUTOR: Jorge Dávila Muro

Esta obra está bajo una licencia Reconocimiento-No comercial-Sin obras derivadas 2.5 España de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc-nd/2.5/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

Índice general

1. OBJETIVOS	1
2. ESTADO DEL ARTE	3
2.1. Orígenes de las infraestructuras de clave pública	3
2.1.1. Criptografía de clave pública	3
2.1.1.1. Generación de claves	4
2.1.1.2. Cifrado de datos	5
2.1.1.3. Firma digital	7
2.1.2. Necesidad de las PKIs	8
2.1.3. Definición de PKI	8
2.1.4. Certificados digitales	9
2.2. Elementos de una PKI	9
2.2.1. Autoridad de Certificación	9
2.2.2. Agencia de Registro	10
2.2.3. Dispositivos de usuario	10
2.3. Servicios de una PKI	11
2.3.1. Verificación de certificados	11
2.3.2. Gestión de certificados	11
2.3.3. Comprobación del estado de un certificado	12
2.3.4. Publicación de certificados	13
2.4. Soluciones de PKI existentes	13
2.4.1. OpenSSL	13
2.4.2. OpenCA PKI	14
2.4.3. EJBCA	14
2.4.4. Entrust	15
2.4.5. VeriSign	15
3. TRABAJO PREVIO	17
3.1. Certificados digitales X.509	17
3.1.1. Definición	17
3.1.2. Estructura básica	18

3.1.3.	Extensiones de los certificados	19
3.1.3.1.	Identificador de clave del emisor	20
3.1.3.2.	Identificador de clave del sujeto	20
3.1.3.3.	Uso de la clave	20
3.1.3.4.	Políticas del certificado	21
3.1.3.5.	Restricciones básicas	22
3.1.3.6.	Uso extendido de la clave	22
3.1.3.7.	Puntos de distribución de CRLs	22
3.2.	Almacenamiento de identidades	23
3.2.1.	PKCS #12: Personal Information Exchange Syntax Standard . . .	23
3.2.2.	PKCS #11: Cryptographic Token Interface Standard	24
3.2.3.	PKCS #15: Cryptographic Token Information Format Standard . .	25
3.3.	Protocolos de gestión de certificados	26
3.3.1.	Solicitudes PKCS #10	26
3.3.2.	Certificate Management Messages over CMS	27
3.3.3.	CMP: Certificate Management Protocol	28
3.4.	Protocolos de comprobación del estado de certificados	31
3.4.1.	CRL: Certificate Revocation List	31
3.4.2.	OCSP: Online Certificate Status Protocol	32
3.4.3.	RTCS: Real-time Certificate Status Facility for OCSP	34
3.5.	Protocolos de publicación de certificados	34
3.5.1.	Publicación Web	35
3.5.2.	LDAP: Lightweight Directory Access Protocol	35
4.	DESARROLLO SOFTWARE	39
4.1.	Características de la solución	39
4.2.	Tecnologías utilizadas	40
4.2.1.	Lenguajes de programación	40
4.2.2.	Bibliotecas	40
4.2.3.	Herramientas	41
4.3.	Análisis del protocolo CMP	43
4.3.1.	Estructura general de los mensajes	43
4.3.2.	Cabecera de los mensajes	43
4.3.3.	Estructuras de datos	45
4.3.3.1.	Valores solicitados para el certificado	45
4.3.3.2.	Valores cifrados	45
4.3.3.3.	Códigos de error	46
4.3.3.4.	Estructuras de representación de las pruebas de posesión	47
4.3.4.	Protección de los mensajes	48
4.3.4.1.	Protección basada en secretos compartidos	48
4.3.4.2.	Protección basada en firmas digitales	49

4.4.	Autoridad de certificación	49
4.4.1.	Inicialización del sistema	50
4.4.1.1.	Configuración previa	50
4.4.1.2.	Generación de la identidad digital	50
4.4.1.3.	Sistema de almacenamiento	51
4.4.2.	Servidor CMP	53
4.4.2.1.	Solicitud inicial de certificación	53
4.4.2.2.	Revocación de certificados	54
4.4.2.3.	Solicitud de recertificación	55
4.4.2.4.	Solicitud de renovación de clave	56
4.4.2.5.	Mantenimiento de la base de datos	56
4.5.	Agencia de registro y revocación	57
4.5.1.	La agencia de registro	57
4.5.1.1.	Alta de usuarios	58
4.5.1.2.	Baja de usuarios	61
4.5.2.	La agencia de revocación	61
4.6.	Aplicación de usuario	64
4.6.1.	Solicitud inicial	65
4.6.2.	Revocación	66
4.6.3.	Recertificación	66
4.6.4.	Actualización de la clave	67
4.6.5.	Verificación de certificados utilizando RTCS	67
4.7.	Servicio de publicación LDAP	67
4.7.1.	Modificaciones sobre la biblioteca <i>Cryptlib</i>	69
4.8.	Servicios de validación	70
4.8.1.	Generación de CRLs	70
4.8.2.	Servidor de validación RTCS	71
4.9.	Esquema general de la solución	73
5.	CONCLUSIONES	75
	Bibliografía	77
	Referencias	79
A.	MANUALES DE INSTALACIÓN	81
A.1.	Manual de instalación de la autoridad de certificación	81
A.1.1.	Requisitos de la aplicación	81
A.1.2.	Instalación de la aplicación	81
A.1.3.	Configuración previa	82
A.1.3.1.	Acceso a dispositivos PKCS #11	82

A.1.3.2.	Acceso a la base de datos de la autoridad de certificación	83
A.1.3.3.	Generación de la identidad de la autoridad de certificación	84
A.1.4.	Ejecución de los servicios	84
A.1.4.1.	Servidor CMP	85
A.1.4.2.	Generación de CRLs	86
A.1.4.3.	Servicio de validación RTCS	87
A.1.4.4.	Servicio de publicación LDAP	88
A.2.	Manual de instalación de la agencia de registro	88
A.2.1.	Configuración previa	88
A.2.2.	Instalación de la aplicación	89
A.3.	Manual de instalación de la aplicación de usuario	90

Índice de figuras

2.1. Generación de claves asimétricas	5
2.2. Cifrado de datos	6
2.3. Firma digital	7
3.1. Estructura lógica de un dispositivo PKCS #11	25
3.2. Modelo de una PKI	29
4.1. Generación de la identidad de una autoridad de certificación raíz	52
4.2. Creación de la estructura de la base de datos	53
4.3. Fichero de configuración de la autoridad de certificación	54
4.4. Ventana de conexión con la base de datos	58
4.5. Ventana de inicialización del usuario	59
4.6. Ventana de datos de inicialización de un usuario final	59
4.7. Ventana de datos de inicialización de una autoridad subordinada	60
4.8. Ventana de baja de usuarios	62
4.9. Ventana de búsqueda de certificados para revocar	63
4.10. Ventana de selección de certificados a revocar	63
4.11. Ventana de operaciones de la aplicación de usuario	65
4.12. Ventana de generación de identidades	66
4.13. Fichero de configuración del publicador	68
4.14. Fichero de configuración del generador de CRLs	71
4.15. Fichero de configuración del servidor RTCS	72
4.16. Esquema general de la solución software	74
A.1. Compilación e instalación de Cryptlib	82
A.2. Compilación de las aplicaciones de la autoridad de certificación	82
A.3. Ejemplo de fichero de configuración .odbcinst.ini	83
A.4. Ejemplo de fichero de configuración .odbc.ini	84
A.5. Ejemplo de fichero de configuración del servidor CMP	85
A.6. Ejemplo de fichero de configuración del generador de CRLs	87
A.7. Ejemplo de fichero de configuración del validador RTCS	88
A.8. Ejemplo de fichero de configuración del publicador LDAP	89

A.9. Ejemplo de fichero de configuración de la agencia de registro	89
--	----

Capítulo 1

OBJETIVOS

El objetivo de este proyecto es realizar un estudio sobre el funcionamiento y la construcción de una infraestructura de clave pública, en adelante PKI¹. Para ello se llevarán a cabo dos tareas:

1. Análisis y estudio de los estándares y protocolos más actuales y avanzados que permitan el establecimiento de PKIs jerárquicas siguiendo el esquema de identidades X509v3.
2. Desarrollo software de un conjunto de herramientas que implementen los estándares estudiados y permitan la construcción de dichas PKIs. Para este desarrollo se utilizará la biblioteca criptográfica *Cryptlib* desarrollada por Peter Gutmann.

El primer objetivo del proyecto proporcionará una amplia documentación de los elementos y procedimientos que componen una PKI, así como las interacciones entre los mismos, para proporcionar todos los servicios que debe ofrecer una PKI.

El segundo objetivo dará como resultado un conjunto de aplicaciones simples de usar y portables que permitan construir una solución software de PKI completa y lo más estándar posible.

¹Public Key Infrastructure

Capítulo 2

ESTADO DEL ARTE

2.1. Orígenes de las infraestructuras de clave pública

2.1.1. Criptografía de clave pública

Hasta hace poco menos de cuarenta años los sistemas de cifrado estaban basados en criptografía simétrica. En estos sistemas, las dos partes que desean intercambiar un mensaje deben compartir, y mantener en absoluto secreto, una clave acordada de antemano, mediante un método no criptográfico; por ejemplo comunicándosela cara a cara o usando un método de entrega convencional certificado. Este tipo de sistemas presentan un gran número de dificultades prácticas a la hora de realizar la distribución de claves. La invención de la criptografía de clave pública solucionó estos problemas; con la criptografía de clave pública los usuarios pueden comunicarse de manera segura a través de un canal inseguro sin necesidad de acordar una clave de antemano.

En 1976 Whitfield Diffie y Martin Hellman, influenciados por los trabajos de Ralph Merkle en el campo de la distribución de claves publicaron un método para la negociación de claves. Este método, conocido como *Diffie-Hellman key exchange* [DH76] fue el primer método de aplicación práctica para compartir una clave secreta sobre un canal inseguro de comunicaciones sin la necesidad de disponer de un secreto compartido de antemano.

Dos años después, en 1977, Rivest, Shamir y Adleman trabajando en el MIT¹ publicaron el algoritmo RSA [RSA78] que fue el primer algoritmo conocido que permite realizar tanto cifrado como firmas digitales, y que supuso uno de los mayores avances en la criptografía de clave pública.

Desde los años setenta se han publicado una gran variedad de algoritmos de cifrado, firma digital y negociación de claves dentro del campo de la criptografía de clave pública:

- en 1984 *Taher Elgamal* [ElG85] publicó el sistema de cifrado *ElGamal*, basado en el algoritmo de negociación de claves de *Diffie-Hellman*, que permite realizar cifrado

¹Massachusetts Institute of Technology

de datos con clave pública.

- en 1985 *Neal Koblitz* [Kob87] y *Victor S. Miller* [Mil85] publicaron, de manera independiente, varios artículos en los que sugerían el uso de curvas elípticas en criptografía. La criptografía de curva elíptica, generalmente conocida como ECC², es una propuesta para utilizar la estructura algebraica de las curvas elípticas sobre campos finitos como algoritmos de clave pública.
- en 1991 el NIST³ publicó el algoritmo DSA⁴ [U.S94] para su uso en su estándar de firma digital.

Los algoritmos de clave pública, como RSA o DSA, se basan en que cada usuario utiliza un par de claves relacionadas matemáticamente, en la que una de ellas descifra el cifrado que se realiza con la otra. Estos algoritmos tienen la propiedad adicional de que conociendo una de las claves del par es computacionalmente imposible deducir la otra. Una de estas claves, que debe permanecer siempre en secreto, se conoce como privada y la otra como pública. Estos algoritmos permiten realizar dos operaciones:

- Cifrado: Un mensaje cifrado con la clave pública de un destinatario no puede ser descifrado por nadie excepto por el destinatario que está en posesión de la correspondiente clave privada. Este mecanismo proporciona confidencialidad.
- Firma digital: Un mensaje cifrado con la clave privada del emisor puede ser descifrado por cualquiera que tenga la clave pública de dicho emisor, probando de esa manera que sólo ese emisor pudo cifrar el mensaje y que no ha sido modificado. La firma digital proporciona autenticación.

2.1.1.1. Generación de claves

Para utilizar los algoritmos de clave pública, cada una de las dos entidades que desean intercambiar información deben disponer de un par de claves relacionadas matemáticamente: una privada, que sólo conoce su propietario y otra pública que debe ser conocida por cualquiera que desee comunicarse con él.

La fortaleza de los algoritmos más utilizados, RSA y DSA, reside en la dificultad de factorizar números grandes ya que la generación de claves se basa en elegir dos números primos pseudoaleatorios. En la figura 2.1 en la página siguiente se puede observar el esquema básico de generación de claves para un usuario.

²Elliptic curve cryptography

³National Institute of Standards and Technology

⁴Digital Signature Standard

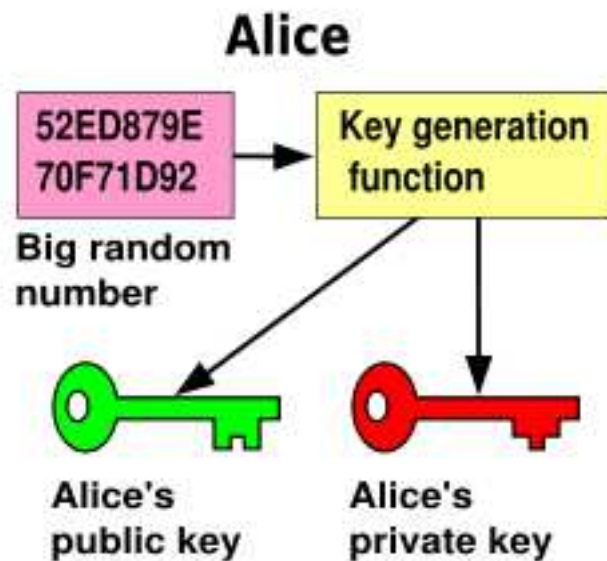


Figura 2.1: Generación de claves asimétricas

2.1.1.2. Cifrado de datos

El cifrado de datos con algoritmos de clave pública se realiza de la siguiente forma: supongamos que Bob desea enviarle un mensaje a Alice; Bob cogerá el mensaje que desea enviar y lo cifrará con la clave pública de Alice; cuando Alice reciba el mensaje utilizará su clave privada para descifrarlo y leer el mensaje original que Bob deseaba enviarla. En la figura 2.2 en la página siguiente se puede ver como funciona este mecanismo. Como Alice es la única que conoce su propia clave privada, sólo ella podrá descifrar el mensaje que le envía Bob. Mediante este procedimiento conseguimos realizar envío de datos con confidencialidad.

Debido a que los algoritmos asimétricos son bastante más lentos que los simétricos, lo que se hace realmente en este tipo de cifrados es:

1. Sortear una clave para un algoritmo de cifrado simétrico y cifrar el mensaje que se desea enviar.
2. Cifrar esta clave simétrica con la clave pública del destinatario y enviar el resultado junto con el cifrado anterior.
3. Cuando el destinatario reciba el mensaje descifrará la clave simétrica con su clave privada y la utilizará para descifrar el mensaje original.

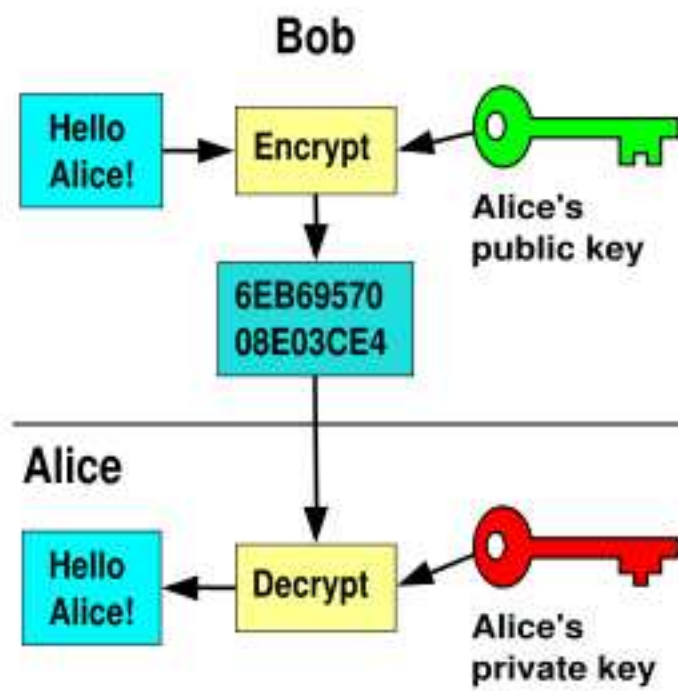


Figura 2.2: Cifrado de datos

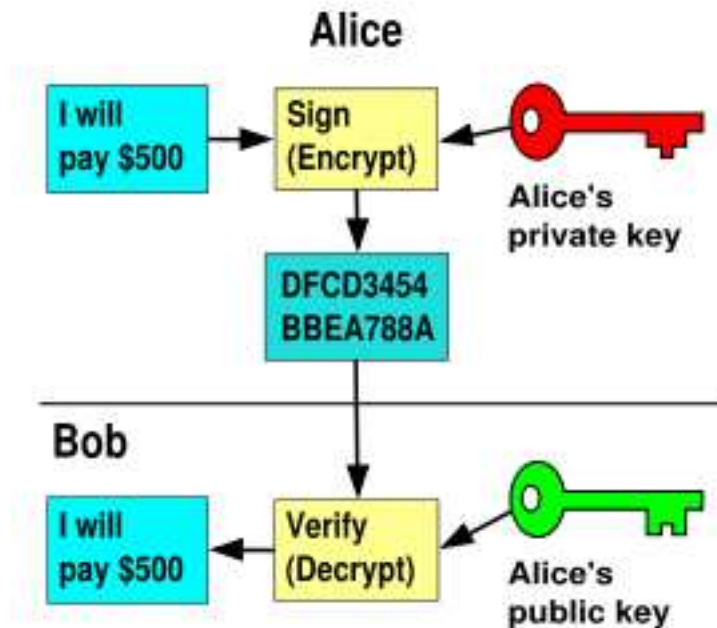


Figura 2.3: Firma digital

2.1.1.3. Firma digital

La firma digital es un procedimiento que permite simular la seguridad que proporciona la firma manuscrita convencional. Como se puede ver en la figura 2.3 realizar una firma digital consiste en lo siguiente: supongamos que Alice desea firmar digitalmente un contrato con Bob; Alice cogerá el contrato, lo cifrará con su clave privada y se lo enviará a Bob; Bob, que tiene la clave pública de Alice será capaz de descifrar el contenido del mensaje; como sólo Alice está en posesión de la clave privada, sólo ella pudo realizar el cifrado. Con este mecanismo se obtiene autenticación del emisor.

Como hemos visto en el punto anterior los algoritmos asimétricos son lentos por lo que si el mensaje que se desea firmar es muy grande, el proceso de firma y verificación puede llevar bastante tiempo. Debido a esto, lo que se hace en lugar de cifrar todo el mensaje es aplicar una función *Hash* al mensaje original y cifrar el resultado.

Una función *hash* es una transformación que toma como entrada una secuencia de longitud arbitraria y devuelve una secuencia de longitud fija que se denomina valor *hash* o resumen. El *hash* es un tipo de "huella digital" del documento original. Una función *hash* debe tener las siguientes propiedades:

- La función debe ser no invertible: dado un valor *hash* h debe ser computacionalmente imposible encontrar un valor m tal que $h = \text{hash}(m)$.

- Dado un valor m_1 debe ser difícil encontrar un valor distinto m_2 tal que $hash(m_1) = hash(m_2)$.
- Debe ser difícil encontrar dos valores m_1 y m_2 tal que $hash(m_1) = hash(m_2)$.

2.1.2. Necesidad de las PKIs

La confianza es parte fundamental de cualquier comunicación, sea electrónica o física. En la comunicación física, la confianza es relativamente fácil de conseguir ya que se puede identificar a la persona viéndola cara a cara o mediante marcas identificativas, como firmas manuscritas. Sin embargo, en el caso de comunicaciones electrónicas, confiar en otra entidad puede ser complicado ya que su identidad puede estar oculta y los signos que nos permitirían identificarla en la vida real no están disponibles. La confiabilidad entre dos entidades no se puede establecer a menos que ambas entidades estén seguras de la identidad de la otra y que la información que están transmitiendo a través de una red no está siendo modificada en la transmisión.

Estos problemas se resuelven en parte con el uso de los algoritmos de clave pública ya que nos proporcionan confidencialidad y autenticación, pero nos surge un problema: ¿cómo podemos estar seguros de que una clave pública pertenece realmente a la persona con la que nos estamos comunicando?; es decir, necesitamos algún método que nos relacione, de alguna manera, la identidad de una entidad en el mundo real con su identidad en el mundo electrónico.

Para resolver estos problemas de confiabilidad, autenticación y seguridad de las comunicaciones surgen las PKIs y los certificados digitales. Una PKI proporciona la seguridad y confianza del mundo real en el mundo electrónico.

2.1.3. Definición de PKI

Una PKI es un entorno formado por el hardware, el software, las políticas y los procedimientos de seguridad necesarios que permiten la ejecución, con garantías, de operaciones criptográficas como el cifrado, la firma digital o el no repudio de las transacciones electrónicas. El término PKI se utiliza a veces de manera errónea para referirse al uso de algoritmos de clave pública en comunicaciones electrónicas. Este significado es incorrecto ya que no se requieren elementos propios de una PKI para usar algoritmos de clave pública.

En general, una PKI permite a dos partes disponer de confidencialidad, autenticación e integridad en las comunicaciones sin tener que compartir ninguna información de antemano.

2.1.4. Certificados digitales

Los certificados digitales son los elementos sobre los que trabajan las PKIs. Un certificado digital es un documento firmado, generalmente público, que asocia una clave pública con una identidad. La identidad puede contener, por ejemplo, el nombre de una persona u organización, su país de procedencia o su dirección de correo electrónico. Un certificado digital contiene tres elementos fundamentales:

- La clave pública del sujeto
- Los datos de identificación del sujeto
- Firma digital de una tercera parte que asegura que los dos elementos anteriores están relacionados.

En una PKI la firma del certificado la realiza una tercera parte confiable, que es la que certifica la relación entre una identidad real y una electrónica. En otros sistemas, como los que siguen el estándar OpenPGP [19], se utilizan los llamados esquemas de confianza, en los que son unos usuarios los que certifican la identidad de otros, en lugar de depender una única entidad confiable.

2.2. Elementos de una PKI

Una PKI debe disponer de tres elementos fundamentales: los dispositivos de usuario, que permiten a una entidad hacer uso de su identidad digital, las agencias de registro, que verifican el vínculo entre una clave pública y su propietario y las autoridades de certificación, que certifican la vinculación verificada por las agencias de registro.

2.2.1. Autoridad de Certificación

Como ya hemos visto, el objetivo de una PKI es generar certificados digitales que asocien la identidad real de una persona, organización o incluso un dispositivo hardware con una clave pública. Una vez que una entidad dispone de un par de claves necesita que una tercera parte certifique, y asocie, ese par de claves con su identidad ante otras entidades. Esta tercera parte confiable en el mundo de las PKIs se denomina Autoridad de Certificación.

Una autoridad de certificación es un agente encargado de generar, custodiar y utilizar su propia identidad digital y de emitir los documentos digitales firmados que constituyen los certificados que emite. Un certificado digital es, en esencia, un documento digital firmado, cuya estructura es públicamente conocida y que incluye los datos verificados necesarios para poder afirmar lo que ese documento certifica.

Hasta ahora disponíamos de mecanismos para realizar comunicaciones autenticadas y confidenciales, pero teníamos el problema de no estar seguros de que la clave pública de la otra entidad correspondiera a quién nosotros pensábamos, a menos que se comprobara de manera externa. Ahora en lugar de tener la clave pública de otra entidad, disponemos de un certificado digital que nos asegura que dicha entidad es quien dice ser siempre y cuando confiemos en la Autoridad de Certificación que emitió su certificado.

Existen dos tipos de autoridades de certificación en función de quién firma su propio certificado:

- Si la autoridad de certificación firma su propio certificado con su clave privada se la denomina autoridad de certificación raíz. Este tipo de autoridades de certificación no disponen de una tercera parte confiable que asegure que son quien dicen ser. Es cada una de las entidades que vayan a utilizar el certificado de dicha autoridad la que debe decidir si confía o no en el mismo. La confianza o no en este tipo de autoridades suele venir dada por su reputación o conocimiento personal, por ejemplo, algunas grandes empresas dedicadas al negocio de las PKIs disponen de autoridades de certificación raíz generalmente aceptadas, e incluso instaladas como confiables por defecto en muchos navegadores *Web*.
- El otro tipo de autoridades de certificación, llamadas autoridades de certificación subordinadas, disponen de un certificado que no está firmado por si mismas, sino por otra autoridad de certificación, ya sea raíz o no.

Con estos dos tipos de autoridades de certificación vemos que se pueden crear estructuras jerárquicas arborescentes en las que unas autoridades de certificación pueden firmar el certificado de otra autoridad o bien de un usuario final.

2.2.2. Agencia de Registro

Una agencia de registro es el agente encargado de recibir las solicitudes de emisión de certificados para una determinada autoridad de certificación. Estos agentes deben verificar que todos los datos y documentos (digitales o no) que aporta una entidad son auténticos y que permiten identificar unívocamente al solicitante del certificado. También deben deducir de ellos si la solicitud es pertinente o no, y si puede emitirse el correspondiente certificado. Las agencias de registro son una parte fundamental de una PKI ya que son las que comprueban la relación entre una clave pública y su propietario antes de que la autoridad de certificación la plasme en un certificado digital.

2.2.3. Dispositivos de usuario

Los usuarios deben disponer de herramientas software y/o hardware que actúen en su nombre y le permitan hacer uso de sus identidades digitales, en concreto deben ser capaces de:

- Generar identidades digitales: Los dispositivos deben contener un generador de secuencias aleatorias que le permitan generar el material de partida de las claves que constituyan una identidad digital, así como almacenarlas de forma segura.
- Solicitar certificados digitales: Una vez autorizado por la agencia de registro, el dispositivo debe permitir solicitar a la autoridad de certificación un certificado digital para un par de claves.
- Realizar operaciones criptográficas, como firmas digitales o cifrado de datos, con sus identidades digitales que es fin último de una PKI.

2.3. Servicios de una PKI

2.3.1. Verificación de certificados

Una vez descritos los elementos de una PKI y como se generan los certificados digitales debemos ver como se utilizan. Llegados a este punto sabemos que en lugar de disponer simplemente de la clave pública de otra entidad, disponemos de su certificado. Antes de utilizar un certificado es necesario verificar que está firmado por una autoridad de certificación confiable para nosotros. Para ello se deben realizar los siguientes pasos:

1. Si el certificado está auto firmado, es decir, la clave pública está firmada con la clave privada del mismo par de claves, entonces el usuario debe decidir si confía en el certificado o no.
2. Si el certificado está firmado por una autoridad de certificación raíz, el certificado será confiable si la autoridad de certificación que lo firmó es confiable para nosotros.
3. Si el certificado está firmado por una autoridad de certificación no raíz, entonces: si la autoridad de certificación es confiable el certificado lo será y sino se debe verificar el certificado de dicha autoridad desde el paso 2. A este procedimiento se le denomina verificación de la cadena de certificación.

Viendo el procedimiento de verificación de certificados queda clara la estructura jerárquica de las PKIs a la que hacíamos referencia en apartados anteriores.

2.3.2. Gestión de certificados

Hasta ahora hemos visto el proceso que se debe seguir para solicitar un certificado, pero no es el único servicio que debe proporcionar una PKI. Los certificados digitales tienen una fecha de validez, que impone la autoridad de certificación, durante la que se compromete a certificar que un usuario está vinculado a un par de claves. Una PKI debe

disponer de protocolos para realizar otro tipo de trámites distintos de la solicitud de certificados entre los que se encuentran:

Revocación: Un usuario o autoridad de certificación puede querer rescindir la asociación entre un individuo y su par de claves. Este proceso se conoce como revocación de certificados. La necesidad de revocar un certificado puede deberse a varios factores, por ejemplo, la pérdida de la clave privada por parte de un usuario o el cese de afiliación de un individuo dentro de la organización reflejada en su certificado.

Recertificación: Los certificados digitales tienen unas fechas de validez que indican desde cuando y hasta cuando un certificado es válido. Una vez finalizado el plazo de validez de un certificado puede ser útil que éste se pueda renovar durante un nuevo periodo de tiempo siempre que los datos de identificación del usuario y su par de claves sigan siendo válidos.

Renovación: La renovación de un certificado consiste en solicitar un nuevo certificado con los mismos datos que el actual pero para un nuevo par de claves. Este procedimiento se realiza cuando un usuario desea cambiar el par de claves que utiliza, pero todos los datos de identificación siguen siendo válidos. La recertificación de certificados se puede ver como un caso particular de la renovación en el que el usuario mantiene el mismo par de claves.

La revocación de certificados debe ser un servicio obligatorio que debe proporcionar toda PKI, mientras que la recertificación y la renovación son opcionales. Algunas PKI pueden, además, obligar al usuario a volver a pasar por la agencia de registro para volver a demostrar su identidad antes de permitir la realización de operaciones de recertificación y renovación. Todas las reglas que se deben cumplir para realizar estas operaciones deben estar recogidas en el documento de políticas de certificación que debe disponer toda agencia de certificación y que debe estar disponible para todos sus usuarios.

2.3.3. Comprobación del estado de un certificado

En apartados anteriores hemos visto como son los certificados, como se generan y como se verifica su confiabilidad, pero en el punto 2.3.2 hemos visto que algunos certificados pueden no ser válidos aunque estén dentro de su periodo de validez, ya que alguna circunstancia extraordinaria ha podido provocar que se hayan revocado. Por tanto es necesario disponer de mecanismos adicionales que nos permitan comprobar si un certificado ha sido revocado, y por tanto ya no es válido lo que certifica, o no. Existen dos tipos de servicios que permiten realizar estas comprobaciones:

- Servicios *offline*: Las agencias de certificación pueden poner a disposición de los usuarios, cada cierto tiempo, una lista con los certificados que están dentro de su

periodo de validez pero que han sido revocados. Es tarea de los dispositivos de usuario obtener una copia actualizada de estas listas periódicamente para asegurarse de que un certificado no ha sido revocado antes de utilizarlo.

- Servicios *online*: Algunas PKIs disponen de servicios *online* que permiten a un usuario consultar el estado que tiene un certificado en el instante en el que va a ser utilizado.

La manera de acceder a este tipo de servicios suele venir definida en el propio certificado digital que se desea verificar.

2.3.4. Publicación de certificados

Una vez que un usuario dispone de un certificado digital le surge la necesidad de compartirlo con otras entidades para poder comunicarse de manera segura con ellas. Supongamos que un usuario desea enviar un mensaje cifrado a otro. Para ello el usuario debe disponer del certificado digital del destinatario. Con lo que hemos visto hasta ahora el emisor debería solicitar al destinatario su certificado digital de manera convencional y después enviarle el mensaje cifrado que deseaba. Para evitar este inconveniente, las PKIs suelen proporcionar servicios de publicación de certificados, en los que los usuarios de una PKI, o incluso a veces cualquiera que lo necesite, pueda obtener los certificados emitidos para un usuario.

2.4. Soluciones de PKI existentes

Con la invención de la criptografía de clave pública y los certificados digitales algunas empresas predijeron la creación de un gran mercado en torno a las PKIs, el comercio electrónico y otros servicios que se podrían comenzar a desarrollar a través de Internet mediante el uso de protocolos seguros de comunicaciones. En el mercado podemos encontrar un gran número de soluciones PKI, algunas libres y gratuitas, otras de pago e incluso algunas desarrolladas a medida para soluciones concretas. A continuación mencionaremos algunas de las soluciones más conocidas, así como sus características principales.

2.4.1. OpenSSL

El proyecto OpenSSL [20] es un esfuerzo colaborativo para desarrollar un conjunto de herramientas de código abierto, robustas, de nivel comercial y con el mayor número de características disponibles que implementen los protocolos SSL⁵ y TLS⁶ para el

⁵Secure Sockets Layer

⁶Transport Layer Security

establecimiento de comunicaciones seguras, así como una biblioteca criptográfica de propósito general.

El proyecto esta formado por tres elementos: una biblioteca que implementa los protocolos SSL y TLS, una biblioteca criptográfica y una aplicación de línea de comandos que permite utilizar los mecanismos implementados en las dos bibliotecas anteriores.

Con la aplicación de línea de comandos se puede construir una solución de PKI funcional aunque bastante rudimentaria y farragosa ya que es necesario crear invocaciones al comando *openssl* con un número enorme de parámetros. Por este motivo, no es muy recomendable construir una PKI directamente con esta capa que proporciona OpenSSL, aunque veremos que otras soluciones utilizan sus bibliotecas para desarrollar su software.

2.4.2. OpenCA PKI

El proyecto OpenCA [17] es un proyecto colaborativo cuyo objetivo es desarrollar una autoridad de certificación de código abierto utilizando los algoritmos más utilizados por la comunidad. El proyecto OpenCA está basado en otros proyectos de código abierto como OpenLDAP, OpenSSL y el proyecto Apache. El desarrollo del proyecto está dividido en dos tareas: estudiar y refinar el esquema de seguridad que garantice el mejor modelo a utilizar en una autoridad de certificación y desarrollar el software que permita administrar y configurar una autoridad de certificación.

El software proporciona una interfaz web, a través del servidor web Apache, que se puede utilizar mediante un navegador y que permite realizar las operaciones que proporciona la autoridad de certificación: solicitudes, revocación, búsqueda de certificados... La configuración de las autoridades de certificación también se realiza mediante una interfaz web. La aplicación también implementa protocolos de comprobación de estado de certificados tanto *online* como *offline* así como servicios de publicación de certificados.

2.4.3. EJBCA

EJBCA [9] es una autoridad de certificación completamente funcional. Está basada en la tecnología J2EE lo que la hace flexible e independiente de la plataforma. Esta solución se puede utilizar de manera independiente o integrada en otra aplicación basada en J2EE.

Este proyecto proporciona una solución PKI de nivel empresarial que permite construir una infraestructura completa para grandes empresas y organizaciones. Según su propia pagina web, si sólo se pretende emitir unos cuantos certificados para realizar pruebas o para pocos usuarios, existen otras soluciones más simples.

La interfaz de la aplicación con los administradores del sistema, así como con los usuarios finales, es vía web a través de un navegador.

Algunas características importantes de este software son:

- permite el manejo de varias autoridades de certificación simultáneamente
- tiene soporte para hardware criptográfico para acelerar ciertas operaciones y para el almacenamiento seguro de las identidades digitales de las autoridades de certificación.
- tiene soporte para protocolos de comprobación del estado de los certificados, así como servicios de publicación de los mismos.

Como opinión personal, mencionar que la interfaz de configuración es un tanto liosa y que se necesita una maquina potente para instalar el software, ya que al estar hecha en Java y utilizar JBoss [13] para servir la aplicación, también hecho en Java, el consumo de memoria es bastante alto.

2.4.4. Entrust

El software *Entrust Authority* [10] es el más popular dentro de las soluciones de PKI no libres. Está pensado sobre todo para proporcionar a grandes empresas certificados para su uso en el protocolo SSL. Según su página web sus características más importantes son:

- facilidad para la realización de solicitudes
- gestión de clientes: añadir, modificar o borrar
- servicios a medida bajo demanda
- herramientas de auditoría
- basado en web: no se necesita descargar una aplicación cliente

Al tratarse de un software de pago, en su página web se encuentra poca información técnica.

2.4.5. VeriSign

La empresa Verisign comercializa el software *VeriSign Public Key Infrastructure* [27] que permite construir infraestructuras de clave pública. Este software, al igual que el de *Entrust*, está orientado a grandes empresas para proporcionar servicios SSL, protección del correo electrónico así como seguridad en servicios de mensajería instantánea. La información disponible en su página web es de tipo comercial más que técnica por lo que no se pueden detallar las características técnicas de esta solución.

Uno de los servicios más importantes que proporciona Verisign es la emisión de certificados para su uso en el protocolo SSL sobre todo para sitios web. Disponer de un certificado emitido por Verisign permite que la mayoría de los navegadores lo acepten como confiable ya que su autoridad de certificación viene instalada por defecto en los navegadores web más utilizados.

Capítulo 3

TRABAJO PREVIO

Para llegar al objetivo de construir una solución software para la construcción de PKIs es necesario estudiar y asimilar los estándares que definen los elementos y servicios que deben implementar estas PKIs. En este apartado se describirán y estudiarán estos estándares para poder elegir los mas adecuados para la implementación de la solución.

3.1. Certificados digitales X.509

Un certificado digital es un documento digital firmado, mediante el cual una tercera parte confiable (una autoridad de certificación) garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública. En el mundo de las PKIs el formato utilizado para representar certificados digitales se rige por el estándar X.509 [HPF02].

La primera versión del estándar X.509 fue publicada en 1988 y su desarrollo comenzó conjuntamente con el estándar X.500. X.500 es un conjunto de estándares de redes de computadores sobre servicios de directorio. Este conjunto de estándares se desarrolló para usarlo como soporte del correo electrónico X.400. Los protocolos definidos en este conjunto de estándares incluyen el protocolo de acceso al directorio DAP, así como el sistema de directorio.

3.1.1. Definición

Cuando usamos una clave pública necesitamos tener la certeza de que la clave privada asociada pertenece a la persona correcta con la cual queremos utilizar un mecanismo de firma digital o de cifrado. Esta confianza se obtiene a través del uso de los certificados de clave pública, que son estructuras de datos que asocian claves públicas con entidades. La asociación se asegura teniendo una autoridad de certificación confiable que firme cada certificado. Las autoridades de certificación, y más concretamente las agencias de registro, deben asegurar esta asociación mediante algún método: por ejemplo mediante el

uso de pruebas de posesión de la clave privada o incluso con la presentación de la misma. Los certificados tienen un tiempo de vida limitado que está reflejado en los contenidos firmados. Puesto que la validez y la firma de un certificado se pueden comprobar de manera independiente por el usuario que desee utilizarlo, los certificados pueden ser distribuidos a través de canales de comunicaciones inseguros e incluso ser almacenados de manera no segura, es decir, sin necesidad de cifrarlos.

En los siguientes apartados veremos la estructura de los certificados X.509 y los campos que los componen. En el estándar X.509 se define el concepto de "conforming certificate authority", que son aquellas autoridades de certificación que se ciñen lo más posible a dicho estándar. Aquí estudiaremos los campos necesarios para construir este tipo de autoridades de certificación ya que son los que seguirá nuestra solución software.

3.1.2. Estructura básica

La estructura básica de un certificado X.509 está compuesta por los siguientes elementos:

tbsCertificate: Son los datos firmados del certificado. Este campo contiene el nombre del sujeto y del emisor del certificado, el periodo de validez y otra información asociada. Este campo también puede incluir extensiones cuyo formato veremos en el apartado 3.1.3.

signatureAlgorithm: Es el identificador del algoritmo criptográfico utilizado por la autoridad de certificación para firmar el certificado.

signatureValue: Contiene la firma digital realizada sobre la codificación ASN.1¹ DER² del campo *tbsCertificate*. Con la generación de esta firma la autoridad de certificación certifica la validez de la información del campo *tbsCertificate*.

El elemento *tbsCertificate* está formado por los siguientes elementos:

- Versión: Define la versión del certificado. Según la última revisión del estándar el valor de este elemento debería ser siempre 3.
- Número de serie: El número de serie debe ser un número entero positivo asignado por la autoridad de certificación a cada certificado. Este número debe ser único para cada certificado emitido por una misma autoridad de certificación.
- Algoritmo de firma: Este campo contiene el algoritmo de firma utilizado por la autoridad de certificación para firmar el certificado. El valor de este campo debe ser exactamente el mismo que el del campo *signatureAlgorithm*.

¹Abstract Syntax Notation One

²Distinguished Encoding Rules

- **Emisor:** Este campo identifica a la entidad que ha firmado y emitido el certificado. Este campo no puede estar vacío y debe seguir el formato del tipo *Name* definido en el estándar X.501. El tipo *Name* está definido como un conjunto de pares atributo-valor (el tipo de los atributos está definido en el estándar X.520). Las implementaciones del estándar X.509 deben soportar, al menos, los siguientes tipos de atributos: país, organización, unidad organizativa, estado o provincia y nombre común.
- **Sujeto:** Este campo identifica a la entidad asociada con la clave pública almacenada en el campo *clave pública del sujeto* (que veremos más adelante). El formato de este campo es el mismo que el del campo Emisor.
- **Validez:** El periodo de validez de un certificado es el intervalo de tiempo durante el cual la autoridad de certificación garantiza que mantendrá información sobre el estado del certificado. El formato de este campo es una secuencia de dos fechas: la fecha en la que el periodo de validez comienza y la fecha en la que finaliza.
- **Información de la clave pública del sujeto:** Este campo contiene la clave pública del sujeto así como el algoritmo con el cual se debe utilizar la clave, por ejemplo RSA o DSA.

3.1.3. Extensiones de los certificados

Las extensiones definidas en el estándar X.509 proporcionan métodos para asociar atributos adicionales con los usuarios o sus claves públicas y para administrar una jerarquía de certificación. Las extensiones sólo pueden aparecer en un certificado si se trata de la versión 3.

Cada extensión que aparezca en un certificado puede estar marcada como crítica o no crítica. Un sistema que utilice un certificado debe rechazarlo siempre que encuentre una extensión crítica que no sea capaz de interpretar, sin embargo las extensiones no críticas pueden ignorarse si no se reconocen.

Una extensión está formada por un OID³ que la identifica y una estructura ASN.1 que la define. Un certificado no puede incluir más de una instancia de la misma extensión, por ejemplo, un certificado sólo debería incluir una vez, si es necesario, la extensión que contiene información sobre la clave pública de la autoridad de certificación.

Las extensiones más comunes que pueden aparecer en un certificado X.509v3 son las siguientes:

³Object Identifier

3.1.3.1. Identificador de clave del emisor

Este campo proporciona un medio para identificar la clave pública correspondiente a la clave privada utilizada para firmar el certificado. Esta extensión es útil cuando un emisor tiene múltiples claves de firma, por ejemplo: al realizar una renovación de su propio certificado la autoridad de certificación mantiene dos pares de claves de manera simultánea. Dentro de esta extensión el campo *keyIdentifier* debe estar presente siempre para facilitar la construcción de cadenas de certificación, a menos que se trate de un certificado autofirmado, en cuyo caso puede omitirse. Esta extensión debe marcarse siempre como no crítica.

El valor del identificador de clave debe derivarse de la clave pública usada para verificar la firma del certificado o bien debe generarse a partir de un método que genere valores únicos. Los métodos más comunes para generar este valor son:

- Tomar los 160 bits de hash SHA-1 del valor de la clave pública del emisor.
- Concatenar un identificador de tipo, formado por 4 bits con el valor 0100, seguido de los 60 bits menos significativos del hash SHA-1 de la clave pública del emisor.

3.1.3.2. Identificador de clave del sujeto

Esta extensión proporciona un mecanismo para identificar certificados que contengan una clave pública en particular. Para facilitar la construcción de cadenas de certificación esta extensión debería aparecer en todos los certificados que pertenezcan a una autoridad de certificación. Cuando el certificado pertenezca a una autoridad de certificación, el valor de este campo coincide con el valor que aparece en el campo de identificador de clave del emisor de todos los certificados emitidos por esta autoridad de certificación. Los métodos utilizados para calcular el valor de este campo son los mismos que los explicados en el punto 3.1.3.1.

Cuando una entidad final ha obtenido varios certificados para el mismo par de claves, esta extensión proporciona un método para identificar rápidamente el conjunto de certificados que contienen una clave pública en particular.

3.1.3.3. Uso de la clave

Esta extensión define el propósito (por ejemplo, cifrado, firma digital, firma de certificados...) de la clave contenida en el certificado. Esta restricción de uso se debe aplicar cuando el uso de una clave, que puede usarse para distintos tipos de operaciones, debe estar restringido. Por ejemplo, cuando una clave RSA sólo se debe utilizar para verificar firmas, sólo el uso de la clave firma digital y no repudio deberían marcarse. Siempre que esta extensión aparezca debería marcarse como crítica. Los valores disponibles para esta extensión son:

- **Firma digital:** Este valor indica que la clave puede utilizarse en procedimientos de firma digital de objetos que no sean certificados digitales ni listas de revocación de certificados.
- **No repudio:** Este valor se debe marcar cuando la clave pública del sujeto se utiliza para verificar firmas digitales y proporcionar un servicio de no repudio que protege contra la negación de una acción realizada por la entidad firmante. En caso de un conflicto posterior, una tercera parte confiable debe determinar la autenticidad de la firma.
- **Cifrado de claves:** Este valor indica que la clave se puede utilizar para el transporte de claves, por ejemplo, cuando una clave RSA se utiliza para realizar intercambio de claves.
- **Cifrado de datos:** Este valor indica que la clave se puede utilizar para cifrar datos que no sean claves criptográficas.
- **Negociación de claves:** Este valor indica que la clave pública se puede utilizar en protocolos de negociación de claves.
- **Firma de certificados:** La clave pública se puede utilizar para verificar la firma de un certificado.
- **Firma de CRLs:** La clave pública se puede utilizar para verificar la firma de una lista de revocación de certificados.
- **Sólo cifrado:** Este valor solo tiene sentido cuando se ha marcado el uso de la clave para negociación de claves. Indica que en el proceso de negociación esta clave solo puede utilizarse para cifrar.
- **Sólo descifrado:** Al igual que el anterior valor, este campo sólo tiene sentido cuando se ha establecido que la clave pública es válida para su uso en protocolos de negociación de claves. El campo indica que la clave sólo puede utilizarse para descifrar datos durante el protocolo de negociación.

3.1.3.4. Políticas del certificado

Esta extensión contiene una secuencia de valores que proporcionan información sobre la política de certificación bajo la cual se ha emitido un certificado. Los valores típicos que contiene esta extensión son la política bajo la cual se emitió el certificado y los propósitos para los que puede ser usado. Las aplicaciones que definan requisitos acerca de las políticas de certificación que pueden aceptar, deben disponer de una lista de OIDs de las políticas que admiten para compararlas con las que aparecen en el certificado. Si

esta extensión se marca como crítica, las aplicaciones de verificación de cadenas de certificación deben ser capaces de tratar esta extensión o sino rechazar el certificado.

Para conseguir interoperabilidad, el estándar X.509v3 recomienda que esta extensión esté formada por un único OID que especifique la política de certificación. Si el OID no es suficiente para identificar la política se puede utilizar un atributo adicional que contenga una URL que apunte al documento de prácticas de certificación (CSP⁴) de la autoridad de certificación.

3.1.3.5. Restricciones básicas

Esta extensión indica si el sujeto del certificado es una autoridad de certificación o no y define la profundidad máxima que pueden tener las cadenas de certificación que incluyan el certificado. Los dos campos que puede contener esta extensión son:

isCA: Indica si la clave pública del certificado pertenece a una autoridad de certificación. Si el valor de este campo es *falso*, entonces el uso de la clave que indica la posibilidad de firmar certificados no puede estar marcado.

pathLenConstraint: Este campo es un valor entero que indica el número máximo de certificados no autofirmados que pueden seguir a este certificado en una cadena de validación. Si este valor es cero, sólo puede existir un certificado más en una cadena de validación. Si no se establece ningún valor para este campo, se supone un valor ilimitado.

3.1.3.6. Uso extendido de la clave

Esta extensión indica uno más propósitos para los que se puede utilizar la clave pública del certificado además de los establecidos en la extensión del uso de la clave. Como norma general esta extensión sólo aparecerá en certificados de entidades finales. Cada autoridad de certificación podrá definir los usos extendidos de la clave que considere oportunos, aunque el estándar tiene algunos establecidos por defecto, como por ejemplo: autenticación de servidor, autenticación de cliente, firma de código...

3.1.3.7. Puntos de distribución de CRLs

Esta extensión define como se realiza la publicación de CRLs por parte de una autoridad de certificación. Esta extensión está formada por una lista de puntos de distribución en los que se pueden encontrar las CRLs publicadas. La ruta debe ser una URI que apunte, por ejemplo, a un servidor web o a un nodo dentro de un directorio de publicación LDAP.

⁴Certification Practice Statement

3.2. Almacenamiento de identidades

Toda PKI debe proporcionar un método para almacenar de manera segura, tanto las claves generadas por los usuarios como las utilizadas por las autoridades de certificación para firmar los certificados que emiten. Así mismo debe ser posible almacenar los certificados digitales asociados a dichas claves. Existen diferentes soluciones para el almacenamiento de claves, tanto software como hardware, que detallaremos en las siguientes secciones. Los formatos más utilizados han sido elaborados y publicados, dentro del grupo de estándares PKCS, por la empresa RSA Security.

3.2.1. PKCS #12: Personal Information Exchange Syntax Standard

El estándar PKCS #12 [RSA99] describe una sintaxis para intercambiar información personal, incluyendo claves privadas, certificados y secretos de diversos tipos. Los objetos PKCS #12 permiten mover información personal de manera segura y con garantías de integridad entre diferentes sistemas. El estándar define diferentes modelos de uso a la hora de mantener la seguridad y la integridad.

Para mantener la privacidad existen dos modelos de uso:

- Modelo de privacidad basado en clave pública: En este modelo la información se cifra en la plataforma origen utilizando una clave pública confiable. El envoltorio resultado se podrá abrir en la plataforma destino con la clave privada correspondiente.
- Modelo de privacidad basado en una contraseña: La información se cifra con una clave simétrica derivada de un nombre de usuario y la contraseña proporcionada.

Para mantener la integridad de los datos también se dispone de dos modelos:

- Modelo de integridad basado en clave pública: En este modo de funcionamiento la integridad se garantiza mediante una firma digital de los datos, que se genera con la clave privada de la plataforma de origen. En la plataforma de destino se utiliza la correspondiente clave pública para verificar la firma.
- Modelo de integridad basado en contraseñas: La integridad se garantiza mediante un código de autenticación de mensajes (MAC ⁵) derivado de la contraseña secreta de integridad. Cuando se usa en conjunto con el modelo de privacidad basado en contraseñas, ambas contraseñas no tienen porque ser iguales.

Este estándar permite cualquier combinación de modelos de integridad y privacidad, aunque para mejorar la seguridad se deben evitar algunos de ellos, por ejemplo: si se

⁵Message Authentication Code

utiliza el modelo de privacidad basado en contraseñas, es deseable disponer de seguridad física a la hora de transportar el contenedor PKCS#12. Como norma general, los modelos basados en clave pública son más seguros que los basados en contraseñas desde el punto de vista de la seguridad. Sin embargo en muchas ocasiones no es posible utilizarlos ya que no sabemos cual es la plataforma de destino y por lo tanto no conocemos su clave pública.

Este estándar es el más utilizado para el almacenamiento local y el transporte de identidades digitales, aunque quizá no es el mejor. Algunas críticas que se le pueden hacer es que existe una gran confusión acerca del objetivo que tiene: la gran mayoría de los usuarios sólo necesitan los almacenes PKCS #12 para almacenar una, o en algunos casos varias, identidades digitales, pero el estándar permite almacenar otro tipo de objetos, que raramente desean guardar, como CRLs, contraseñas, otros certificados...

3.2.2. PKCS #11: Cryptographic Token Interface Standard

El estándar PKCS #11 [RSA04] define una API⁶ llamada "Cryptoki", independiente de la plataforma, para el acceso a *tokens* criptográficos y tarjetas inteligentes, que permitan almacenar información criptográfica y realizar operaciones sobre dicha información. Los objetivos más importantes del estándar son: obtener independencia de la tecnología utilizada (que sirva para cualquier tipo de dispositivo) y compartición de recursos (que varias aplicaciones puedan acceder a varios dispositivos). La finalidad es proporcionar a las aplicaciones una visión abstracta de cualquier dispositivo *token* criptográfico.

El estándar proporciona los tipos de datos y las operaciones necesarias para implementar una aplicación que requiera hacer uso de operaciones criptográficas sobre dispositivos criptográficos. *Cryptoki* permite aislar a las aplicaciones de los detalles hardware de los dispositivos. Esta API define los objetos criptográficos más utilizados, por ejemplo: claves RSA, certificados X.509, claves simétricas para el algoritmo DES/Triple DES...

Cryptoki proporciona una visión lógica de los dispositivos que implementan el estándar PKCS #11 que se puede ver en la figura 3.1 en la página siguiente. Según esta estructura lógica un dispositivo PKCS #11 es un dispositivo que almacena objetos y puede realizar operaciones criptográficas. El estándar define tres tipos de objetos: datos, certificados y claves. Los objetos del tipo dato los definen las aplicaciones, los objetos de tipo certificado almacenan certificados digitales y los objetos de tipo clave almacenan claves criptográficas, ya sean claves públicas, privadas o claves simétricas. Puesto que la implementación hardware de cada dispositivo es diferente, el objetivo de Cryptoki es traducir los elementos propios de la implementación de los dispositivos en la visión lógica de un dispositivo PKCS #11 estándar.

⁶Application Programming Interface

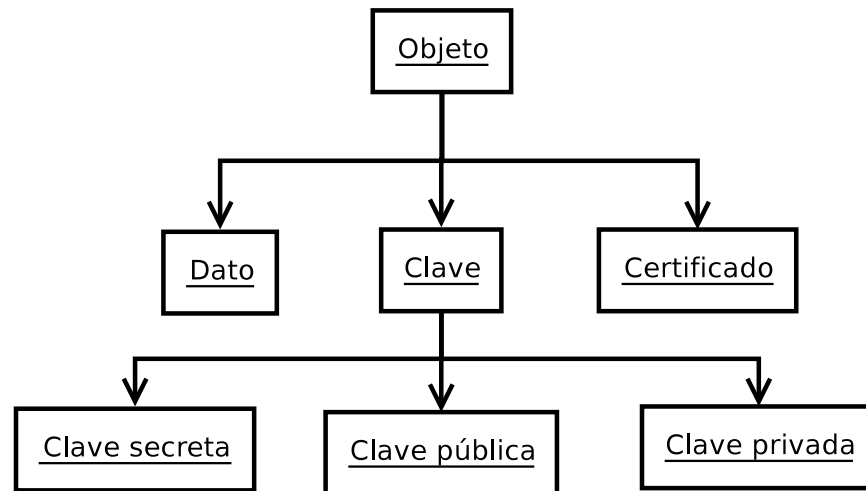


Figura 3.1: Estructura lógica de un dispositivo PKCS #11

Los objetos dentro de un token pueden ser de dos tipos: privados o públicos. Las aplicaciones pueden acceder a los objetos públicos sin necesidad de autenticarse, sin embargo, para acceder a los objetos privados es necesario autenticarse mediante un PIN ⁷ u otro elemento propio del token (algunos tokens disponen de elementos biométricos para realizar la autenticación).

La característica principal de estos dispositivos es que permiten utilizar las identidades digitales que están almacenadas en su interior sin extraerlas del dispositivo, ya que los algoritmos de cifrado están implementados dentro del mismo. Generalmente los pares de claves que se almacenan en este tipo de dispositivos pueden ser generados dentro del mismo o bien importarlos de otro almacén como por ejemplo un PKCS #12. Para mayor seguridad, algunos dispositivos como el DNIe, siguen el estándar FIPS-140 nivel 3 que no permite realizar una copia de seguridad ni exportar un par de claves fuera del mismo para garantizar que no existen más copias del par de claves.

3.2.3. PKCS #15: Cryptographic Token Information Format Standard

Como hemos visto en el punto 3.2.2 los tokens criptográficos proporcionan plataformas seguras para ofrecer una capa de seguridad y privacidad a las aplicaciones. Estos tokens son capaces de almacenar información como claves privadas, contraseñas, certificados... Además, muchos de estos dispositivos tienen la capacidad de realizar operaciones criptográficas sobre los datos que almacenan sin necesidad de extraerlos del propio dispositivo. Esta característica es muy importante a la hora de realizar operaciones

⁷Personal identification number

como firmas digitales.

Por desgracia, el uso de estos dispositivos se ha visto dificultado por la falta de interoperabilidad entre dispositivos de diferentes fabricantes. El mayor problema es que no existen estándares para definir un formato común para el almacenamiento de claves o certificados dentro de los dispositivos. Esto hace difícil crear aplicaciones que puedan funcionar con dispositivos de diferentes proveedores.

El estándar PKCS #15 [RSA00b] surgió para intentar solucionar varios problemas con los tokens criptográficos:

- Definir un formato común para el almacenamiento dentro de los dispositivos.
- Conseguir interoperabilidad entre diferentes dispositivos en diferentes plataformas (independencia de la plataforma).
- Permitir a las aplicaciones utilizar los dispositivos de diferentes fabricantes (independencia del fabricante).
- Permitir el uso de los avances en la tecnología sin necesidad de reescribir las aplicaciones (independencia de la aplicación).
- Mantener consistencia con los estándares existentes y extenderlos únicamente en los casos que sean necesarios.

El objetivo final de este estándar es que un usuario que disponga de un token criptográfico que contenga un certificado pueda utilizarlo en cualquier plataforma y con cualquier aplicación y pueda utilizarlo para realizar las operaciones criptográficas que desee.

Al igual que el estándar PKCS #11 este estándar puede ser implementado tanto en dispositivos hardware como software.

3.3. Protocolos de gestión de certificados

Toda infraestructura de clave pública necesita definir los mecanismos para que los elementos que la componen puedan interactuar entre ellos. A continuación estudiaremos los protocolos más utilizados para la creación y gestión de certificados.

3.3.1. Solicitudes PKCS #10

El estándar PKCS #10 [RSA00a] define una sintaxis para las solicitudes de certificados. Una solicitud de certificación está formada por el nombre de la entidad solicitante, su clave pública y, opcionalmente, un conjunto de atributos, todo ello firmado

digitalmente por la entidad solicitante del certificado. Las solicitudes de certificación se envían a una autoridad de certificación para que genere un certificado X.509.

La inclusión de atributos opcionales tiene dos objetivos: proporcionar información adicional de la entidad solicitante (por ejemplo una dirección postal o una contraseña para permitir la revocación posterior del certificado); y proporcionar atributos para su inclusión como extensiones en el certificado X.509 resultante. Este estándar no define como deben ser las interacciones entre las entidades finales y agencias de registro que autorizan o no las solicitudes de certificación.

El proceso para construir una solicitud de certificación es el siguiente:

1. Crear una estructura del tipo *CertificationRequestInfo*. Esta estructura contiene el nombre identificativo de la identidad solicitante, su clave pública y opcionalmente un conjunto de atributos cuya estructura está definida en el estándar PKCS #9 [RSA00c].
2. Firmar la estructura anterior con la clave privada de la entidad solicitante.
3. Crear una estructura del tipo *CertificationRequest* que contenga: el campo *CertificationRequestInfo*, el identificador del algoritmo utilizado para realizar la firma y la firma de la entidad final.

Una vez que la autoridad de certificación recibe la solicitud deberá autenticar a la entidad solicitante y verificará la firma de la solicitud; si la solicitud es válida emitirá el correspondiente certificado X.509.

Este estándar no define como se debe autenticar al solicitante ni especifica ningún método para recibir la solicitud del certificado ni devolver el certificado generado a la entidad solicitante. Así mismo tampoco se define ningún procedimiento para la realización de revocaciones, renovación de claves, ni recertificaciones.

3.3.2. Certificate Management Messages over CMS

El estándar *Certificate Management Messages over CMS*, definido en el RFC 2797 [MLS00], especifica un protocolo para la gestión de certificados utilizando el estándar CMS⁸ [Hou]. El estándar CMS define un formato para representar mensajes firmados, cifrados y/o autenticados.

Este estándar se diseñó intentando conseguir los siguientes objetivos:

- Reutilizar lo más posible los estándares existentes PKCS #10 y CMS.
- Realizar siempre la generación de claves en el software del cliente.

⁸Cryptographic Message Syntax

- Ofrecer la capacidad de generar pruebas de posesión de la clave por parte del cliente.

Las transacciones que define el protocolo están formadas, generalmente, por un par de mensajes, uno de solicitud y otro de respuesta. Los mensajes de solicitud de certificación siguen el estándar PKCS #10 visto en el punto 3.3.1 en la página 26 y los mensajes de respuesta deben seguir siempre el formato definido en el estándar CMS. Los mensajes de respuesta están firmados por la autoridad de certificación para proporcionar autenticación de las respuestas.

Este protocolo no define ningún mecanismo especial para realizar renovaciones ni recertificaciones.

3.3.3. CMP: Certificate Management Protocol

El estándar CMP [AFK05] define un conjunto de mensajes que permiten implementar todos los aspectos relevantes de la creación y gestión de certificados X.509v3, incluyendo, revocaciones, recertificaciones... En la figura 3.2 en la página siguiente, tomada del estándar, se pueden ver todas las entidades que deben formar parte de una PKI así como las interacciones entre las mismas. Analizando la figura podemos ver, a grandes rasgos, que las operaciones que se pueden realizar son:

1. Establecimiento de la autoridad de certificación: Cuando se inicializa una autoridad de certificación es necesario realizar ciertos pasos, por ejemplo, crear una CRL inicial o exportar y poner a disposición de las entidades finales su clave pública.
2. Inicializar la entidad final: Este proceso consiste en importar la clave pública de la autoridad de certificación y solicitar información sobre las opciones que soporta dicha autoridad.
3. Certificación: Existen diferentes tipos de operaciones que dan lugar a la creación de nuevos certificados.
 - a) Registro y solicitud inicial de certificación: Durante este proceso una entidad final se da a conocer a la agencia de registro antes de que la autoridad de certificación emita ningún certificado para dicha entidad. El objetivo final de este proceso, si se realiza correctamente, es que la autoridad de certificación genere un certificado para la clave pública de la entidad final y se lo entregue o lo publique en un repositorio público.
 - b) Actualización del par de claves: Cada par de claves de una entidad final puede ser actualizado regularmente, es decir, reemplazado con un nuevo par de claves y entonces será necesario generar un nuevo certificado.

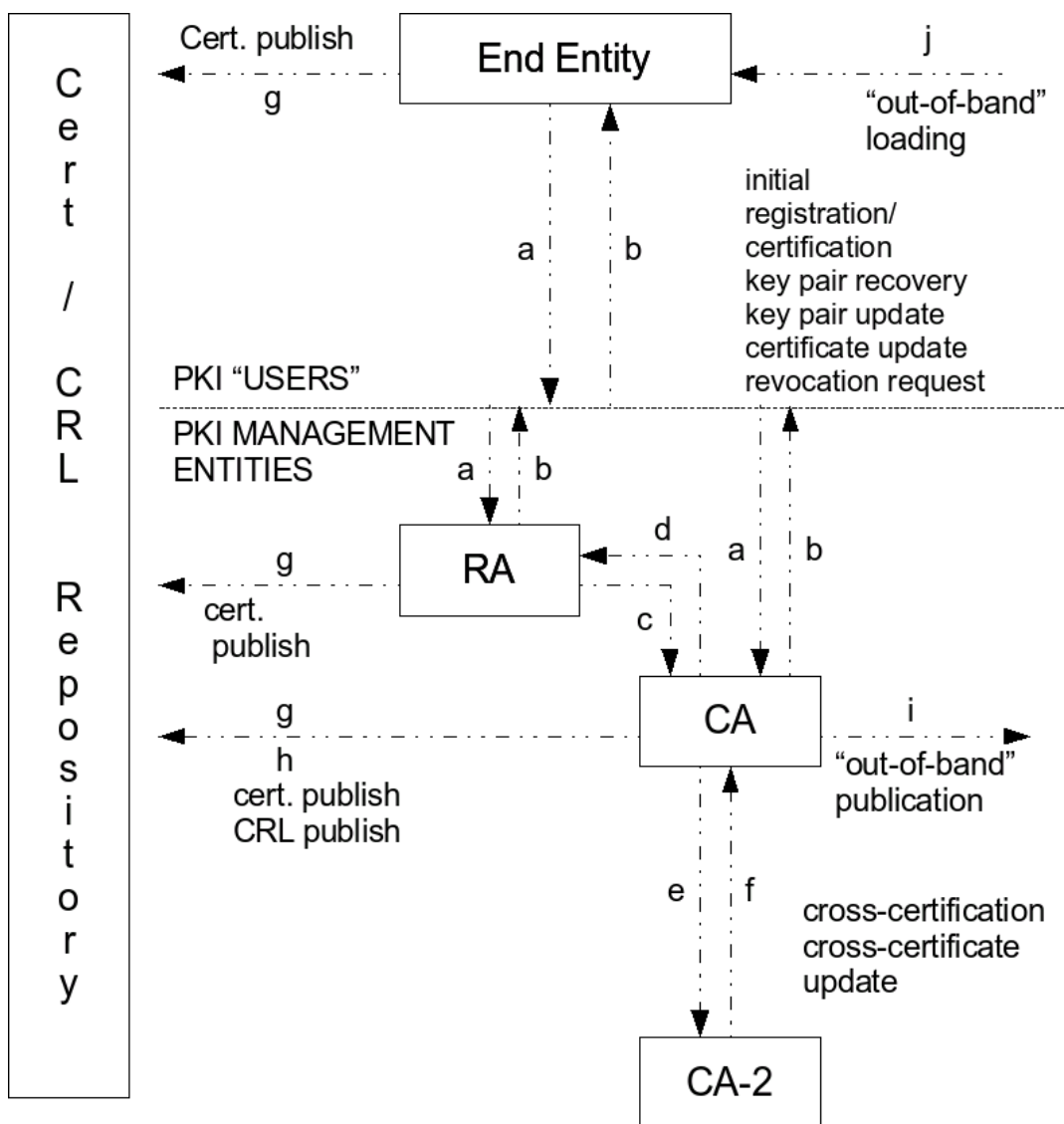


Figura 3.2: Modelo de una PKI

- c) Actualización de certificados: Cuando un certificado está a punto de expirar puede ser renovado siempre que los requisitos que se cumplían cuando se emitió sigan siendo válidos.

Uno de los aspectos más importantes que hay que analizar de este protocolo es como se realiza la solicitud inicial de certificación. El primer paso que debe realizar una entidad final es solicitar información acerca de las funciones que están implementadas en la PKI y obtener de forma segura una copia de la clave pública de la autoridad de certificación. Es importante tener en cuenta que en este punto la entidad final no ha tenido ningún contacto previo con la PKI, así que es necesario disponer de un mecanismo que permita autenticar al usuario ante la autoridad de certificación. Para ello la agencia de registro proporcionará, mediante algún mecanismo externo, un valor secreto (clave inicial de autenticación) y un valor de referencia (utilizado para identificar al usuario) que permitirá a la entidad final autenticar sus solicitudes frente a la autoridad de certificación.

La secuencia de pasos que debe realizar una entidad para solicitar el certificado inicial son:

1. Acudir a la agencia de registro para presentar todos los documentos necesarios que permitan demostrar su identidad.
2. La agencia de registro proporcionará a la entidad final una clave de autenticación inicial, un valor de referencia e información sobre como obtener el certificado de la autoridad de certificación para la cual está realizando la solicitud.
3. La entidad final generará su par de claves, creará la solicitud de certificación, la protegerá mediante la clave de autenticación inicial, realizando un HMAC, y la enviará a la autoridad de certificación
4. La autoridad de certificación verificará la solicitud y si es correcta emitirá el correspondiente certificado digital para entregárselo al usuario.

La secuencia de pasos para otro tipo de solicitudes, como revocaciones, recertificaciones y actualizaciones de clave, es similar. La única diferencia es que en el paso 3 las solicitudes deben estar firmadas, con un par de claves certificado anteriormente por la autoridad de certificación, en lugar de autenticadas con la clave de autenticación inicial. Estos dos mecanismos consiguen que todos los mensajes intercambiados durante el protocolo estén siempre protegidos criptográficamente.

Otra de las características más importantes de este protocolo es la utilización de mecanismos de pruebas de posesión de la clave privada de la entidad solicitante. Estos mecanismos proporcionan pruebas, mediante mecanismos de reto-respuesta, de que el solicitante realmente está en posesión de la clave privada cuya correspondiente clave pública desea que sea certificada.

Este protocolo es el más completo de todos los que existen ya que permite realizar todas las operaciones necesarias que debe proporcionar una PKI. Muchas soluciones PKI actuales, como por ejemplo EJBCA, están comenzando a incluir implementaciones de este protocolo.

3.4. Protocolos de comprobación del estado de certificados

Como hemos visto en apartados anteriores, cuando deseamos verificar un certificado debemos seguir los siguientes pasos:

1. Comprobar la fecha de validez del certificado para comprobar que no haya caducado.
2. Verificar que la firma del certificado es válida.
3. Verificar que el certificado ha sido firmado por una autoridad de certificación confiable.
4. Comprobar que el certificado ha sido emitido para el propósito para el cual lo queremos utilizar.
5. Comprobar si el certificado ha sido revocado.

La revocación de un certificado es un proceso por el cuál la validez de un certificado ha terminado antes de su fecha de caducidad. Esta revocación puede deberse a diferentes causas, por ejemplo: el sujeto del certificado abandona la organización a la que pertenece, el nombre del sujeto ha cambiado, existe la sospecha de que la clave privada de la entidad final ha sido comprometida... Existen diferentes métodos para llevar a cabo la comprobación acerca del estado de un certificado que veremos en los siguientes apartados.

3.4.1. CRL: Certificate Revocation List

Una CRL [HPF02] es una lista, con un sello de tiempo, firmada por una autoridad de certificación, que contiene los números de serie de los certificados que han sido revocados y que puede estar en un repositorio público a disposición de los usuarios. Cuando una aplicación desea utilizar un certificado, además de comprobar la firma y la validez del mismo, debería comprobar que el número de serie del certificado no esté presente en la última CRL emitida. Las listas de revocación de certificados se deben generar cada cierto tiempo, por ejemplo, cada hora o cada día. Cuando se revoca un

certificado, éste deberá aparecer siempre en la siguiente CRL que se emita. Además, un certificado revocado deberá permanecer siempre en todas las CRLs siguientes hasta que se emita una CRL planificada para un momento posterior a la finalización de la validez del certificado. Esto evita el problema de que se revoque un certificado y que la próxima CRL se vaya a emitir después de la finalización de la validez del certificado; en este caso existiría un certificado revocado que nunca habría aparecido en una CRL.

La mayor ventaja de este método de revocación es que las CRLs se pueden distribuir mediante los mismos medios que los certificados digitales, es decir, utilizando métodos de comunicación no confiables.

La mayor limitación en el uso de CRLs es que la granularidad de la revocación en el tiempo está limitada por el periodo de publicación de las CRLs. Por ejemplo, si se realiza una revocación ahora, la información sobre esa revocación no estará disponible hasta que se publique la siguiente CRL, que puede ser dentro de una hora, un día o una semana dependiendo de la frecuencia de actualización de las CRLs.

La estructura general de una CRL es similar a la de un certificado X.509. Los campos más importantes de una CRL son:

***tbsCertList*:** Este campo es una secuencia que contiene el nombre del emisor, la fecha de emisión, la fecha de publicación de la siguiente CRL, la lista de certificados revocados (si hay alguno) y opcionalmente un conjunto de extensiones. Si no hay ningún certificado revocado actualmente la lista de certificados no aparecerá en la estructura. En el caso de que sí existan certificados revocados la lista es una secuencia de elementos compuestos por el número de serie del certificado, la fecha de revocación y opcionalmente extensiones para indicar, por ejemplo, el motivo de la revocación.

***signatureAlgorithm*:** Este campo indica el algoritmo de firma que se debe utilizar para verificar la firma de la CRL.

***signatureValue*:** Este campo contiene la firma digital de la codificación ASN.1 del campo *tbsCertList*.

3.4.2. OCSP: Online Certificate Status Protocol

El protocolo OCSP [MAM99] proporciona un método para determinar el estado de revocación de un certificado X.509 sin necesidad de utilizar CRLs. A diferencia de las CRLs, el protocolo OCSP proporciona información sobre el estado de revocación de un certificado en el momento actual. Los clientes OCSP realizan consultas a los servidores OCSP y aplazan el uso del certificado hasta que reciben una respuesta sobre el estado del mismo.

3.4. PROTOCOLOS DE COMPROBACIÓN DEL ESTADO DE CERTIFICADOS

Una consulta OCSP está formada por los siguientes campos: identificador de versión, tipo de solicitud, el identificador del certificado a consultar y opcionalmente un conjunto de extensiones.

Cuando un servidor OCSP recibe una solicitud deberá comprobar que está bien construida, que sea capaz de realizar el tipo de solicitud que le indica el cliente y que la información que necesita para procesar la solicitud es correcta. Todas las respuestas OCSP tienen que estar obligatoriamente firmadas digitalmente por la autoridad de certificación que emitió el certificado en cuestión o por una entidad confiable para el usuario para firmar respuestas OCSP.

Una respuesta OCSP puede tener tres posibles valores a cerca del estado del certificado:

good: Indica una respuesta positiva a la pregunta. Esta respuesta indica que el certificado no ha sido revocado, lo cual no quiere decir que el certificado sea válido: puede ser que el certificado esté caducado o incluso que nunca haya sido emitido.

revoked: Indica que el certificado ha sido revocado.

unknown: Esta respuesta indica que el servidor no tiene ninguna información acerca del certificado recibido en la consulta.

El protocolo OCSP es uno de los más utilizados en la actualidad para realizar consultas sobre el estado de los certificados. El diseño de este protocolo se realizó basándose en la información disponible en las CRLs y con el objetivo de que ambos mecanismos fueran totalmente compatibles, de hecho, algunos prestadores de servicios OCSP sólo disponen de las CRLs generadas por las autoridades de certificación para obtener información sobre el estado de los certificados. Este tipo de soluciones sólo funcionan correctamente si las autoridades de certificación generan una CRL cada vez que se revoque un certificado en lugar de generarlas periódicamente. Otro problema que presenta este modo de funcionamiento es que a la única pregunta que puede responder el prestador de servicios de validación es ¿está el certificado X en la última CRL de que dispones?; está pregunta sólo tiene dos respuestas válidas: Sí o No, mientras que el protocolo especifica tres tipos de respuestas como hemos visto anteriormente.

Cuando se realiza una consulta OCSP se debe especificar el certificado que se desea consultar, para ello el estándar define la estructura *CertID* que consta de los siguientes campos:

serialNumber: Número de serie del certificado a consultar.

issuerNameHash: Hash del DN del emisor.

issuerKeyHash: Hash de la clave pública del emisor.

hashAlgorithm: Algoritmo hash utilizado para calcular los dos valores anteriores.

Este método de identificación presenta varios problemas:

- En ocasiones es imposible consultar el estado de un certificado por sí solo ya que es necesario conocer la clave pública del emisor del certificado para calcular el valor del campo *issuerNameHash* o bien leerlo de la extensión Identificador de clave del emisor que como vimos en el punto 3.1.3.1 puede no estar presente.
- Los prestadores de servicios OSCP deben disponer de una lista de valores hash de las claves públicas y el DN de cada autoridad de certificación para la que prestan servicios ya que es el único dato que viene en la consulta y no es posible obtener el valor original a partir del hash.

3.4.3. RTCS: Real-time Certificate Status Facility for OSCP

El estándar RTCS [Gut02b] proporciona un conjunto de propuestas para modificar el protocolo OSCP y proporcionar servicios de validación de certificados en tiempo real. A diferencia del OSCP, este protocolo está pensado para proporcionar información sobre el estado de un certificado utilizando un almacén de certificados "vivo" (aunque puede ofrecer información basándose en CRLs igual que OSCP). El objetivo final de esta especificación es poder responder, de manera inequívoca, a la pregunta ¿es este certificado válido ahora mismo?. Para conseguir este objetivo se propone utilizar las extensiones definidas en el estándar OSCP y que en la actualidad no se utilizan.

Las mayores diferencias con el protocolo OSCP, que vienen a resolver los problemas que hemos visto, son las siguientes:

- Utilizar un almacén de certificados en el que podamos tener tanto los certificados revocados como los no revocados para poder dar una respuesta fiable al usuario.
- En lugar de utilizar el mecanismo de identificación de certificados visto en el protocolo OSCP, esta especificación utiliza el concepto más extendido de huella digital del certificado para identificar certificados X.509. La huella digital o *fingerprint* del certificado es un hash del mismo que permite identificarlo y distinguirlo de otro. Con este identificador se puede comprobar el estado de un certificado sin necesidad de disponer de ninguna información no contenida en el propio certificado.

3.5. Protocolos de publicación de certificados

Una PKI puede y/o debe ofrecer servicios de publicación de certificados que permitan a los usuarios descargar certificados de otras personas para poder comunicarse con ellas o

descargar CRLs para comprobar el estado de dichos certificados. La publicación de certificados es un servicio opcional en algunos entornos ya que publicar una lista de personas que están afiliadas a una PKI puede verse como una intromisión de su privacidad. En la actualidad se suelen utilizar dos mecanismos diferentes para la publicación de certificados que veremos en los siguientes apartados.

3.5.1. Publicación Web

Una forma simple de poner a disposición de los usuarios de una PKI los certificados de los demás miembros es utilizar un servidor Web de donde se puedan descargar mediante el protocolo *http*. El acceso a este servidor puede ser público (para cualquier persona) o privado, introduciendo algún mecanismo de control de acceso, por ejemplo algún método simple como autenticación mediante usuario/contraseña, o algo más complejo mediante previa identificación con el certificado del usuario.

Una vez que se tiene acceso a este repositorio se puede mostrar al usuario una lista con los certificados disponibles para que descargue el que necesite o bien disponer de una interfaz Web que permita realizar búsquedas.

Algunas soluciones existentes, como EJBCA, disponen de este mecanismo para realizar la publicación de certificados. EJBCA proporciona dos páginas web, una con enlaces directos a los certificados de las autoridades de certificación en diferentes formatos y otra con una caja de texto en la que se pueden realizar búsquedas de certificados por el número de serie o el DN.

Esta solución no es muy buena ya que no está estandarizada, sino que cada aplicación la implementa de acuerdo a sus necesidades y por tanto no se integra directamente con las aplicaciones en las que se suelen usar los certificados.

3.5.2. LDAP: Lightweight Directory Access Protocol

LDAP [Ope06] es un protocolo de nivel de aplicación, basado en el estándar X.500, que permite realizar consultas y modificaciones sobre un servicio de directorio. Un directorio es un conjunto de objetos con atributos organizados en una jerarquía. Los servidores LDAP suelen utilizarse, sobre todo en entornos empresariales, para almacenar información de acceso de usuarios (nombre de usuario y contraseña), pero también permiten almacenar otro tipo de información como datos de contacto del usuario, certificados... En resumen, el protocolo LDAP es un protocolo de acceso a un conjunto de información sobre una red.

Como hemos vistos los directorios LDAP son una estructura jerárquica de objetos con sus atributos. En la raíz de esta jerarquía se encuentra el nodo raíz que suele ser de una de las siguientes formas:

- `o="Nombre de la organización", c="País"`: Esta era la forma en la que se generaba el nodo raíz en los directorios X.500, sin embargo hoy en día no se suele utilizar

ya que debido a que las grandes empresas están presentes en Internet el campo país deja de tener tanto sentido.

- `o="empresa.es"`: Esta forma está derivada de la anterior en la que se ha eliminado el campo país.
- `dc="empresa", dc="es"`: En este caso se utilizan los componentes del dominio de la empresa en Internet para generar el nodo raíz del directorio. Esta forma es la más actualizada en la actualidad.

Debajo del nodo raíz se crean contenedores lógicos para representar los datos. Por razones históricas y herencia de los protocolos X.500, estas separaciones lógicas se realizan mediante el uso de entradas OU⁹. Estas entradas se utilizaban para separar organizaciones funcionales dentro de las empresas, como por ejemplo ventas, gestión o finanzas, pero hoy en día separa los datos en otro tipo de categorías como usuarios, grupos o dispositivos, aunque se sigue utilizando el nombre OU. Una vez definida la estructura lógica del directorio se puede comenzar a insertar objetos finales, como por ejemplo usuarios, con los atributos que se desee almacenar: nombres, dirección, teléfono...

Para realizar una consulta y obtener datos de un directorio LDAP existen dos opciones:

- Indicarle al servidor LDAP la ruta completa que debe utilizar para navegar por el árbol y llegar al objeto que deseamos buscar.
- Dar al servidor LDAP unos criterios de búsqueda para que encuentre por nosotros el nodo que deseamos y devuelva la información solicitada de ese nodo.

Una vez visto el funcionamiento general de un directorio LDAP analizaremos como se pueden utilizar para almacenar certificados digitales X.509. Para guardar certificados digitales en un directorio LDAP estándar se utiliza el objeto *inetOrgPerson* que sirve para almacenar información asociada con personas. Este objeto tiene un atributo denominado *userCertificate* que permite almacenar, como un fichero binario, un certificado X.509.

El almacenamiento de certificados en directorios LDAP mediante este tipo de objeto presenta diversas limitaciones:

- Cada objeto *inetOrgPerson* sólo puede tener un atributo *userCertificate* por lo que no es posible almacenar varios certificados para un mismo usuario dentro de un nodo.
- Es imposible realizar búsquedas dentro del LDAP sobre los campos de un certificado ya que estos se almacenan como un objeto binario y el servidor LDAP ni siquiera

⁹Organizational Unit

sabe que el contenido es un certificado X.509 (de hecho el contenido del atributo puede ser cualquier objeto binario ya que no se realizan comprobaciones sobre el contenido cuando se inserta).

Para intentar solventar este último problema el *PKIX Working Group* encargado de desarrollar estándares para construir PKI basadas en X.509 creó un estándar llamado *LDAP: Schema Definitions for X.509 Certificates [K. 06]* para permitir almacenar tanto certificados como listas de revocación de una manera mucho más eficiente. Este estándar tiene como objetivo facilitar el almacenamiento de diferentes tipos de objetos que se pueden utilizar en una infraestructura de clave pública: certificados, CRLs o información de usuarios de la PKI y no solamente el almacenamiento de certificados X.509. A pesar de estas definiciones, la mayoría de soluciones de PKI existentes se limitan a almacenar los certificados de la manera especificada anteriormente ya que es lo que espera encontrar una aplicación de propósito general que necesite obtener certificados de un servicio de publicación LDAP.

El uso de servicios de directorio LDAP para la publicación de certificados es una actividad que algunos autores desaconsejan debido a su complejidad y al estar definida en estándares para muchos anticuados y que no se adaptan demasiado a las tecnologías actuales.

En el capítulo de desarrollo de la solución veremos las decisiones de diseño que se han tomado para implementar el soporte de publicación de certificados utilizando LDAP.

Capítulo 4

DESARROLLO SOFTWARE

El objetivo principal de este trabajo es diseñar y programar un conjunto de aplicaciones que permitan la construcción de PKIs jerárquicas siguiendo el estándar X.509. El estudio realizado en el capítulo anterior constituye la base necesaria para elegir los mejores mecanismos y estándares para realizar la implementación de una solución software para PKI. En este capítulo se detallaran las características del sistema implementado, así como las decisiones de diseño tomadas y los detalles de la implementación de cada uno de los elementos.

4.1. Características de la solución

Una vez analizados los fundamentos de las PKIs así como algunas soluciones existentes, los aspectos más importantes que se han tenido en cuenta para diseñar y desarrollar este proyecto son:

Modularidad: En lugar de construir una aplicación enorme que disponga de una gran cantidad de opciones, la aplicación se diseñará como un conjunto de pequeñas aplicaciones que, actuando en conjunto, proporcionen toda la funcionalidad necesaria para construir una PKI.

Estándar: El desarrollo se ha realizado teniendo siempre presentes los estándares existentes para conseguir una aplicación robusta y, en la medida de lo posible interoperable con otras soluciones existentes.

Ligera: La aplicación será lo mas pequeña posible, en cuanto a recursos necesarios para ejecutarse, así como en cuanto a las dependencias que necesite para su funcionamiento.

Portable: Otro objetivo fundamental de la aplicación es que pueda ejecutarse en sistemas heterogéneos, es decir, que sea multiplataforma o independiente de la misma.

Simple: Muchas aplicaciones para la construcción de PKIs existentes presentan bastantes dificultades para instalarse, configurarse e incluso para ser utilizada por usuarios inexpertos. Puesto que la complejidad de las PKIs es, en muchas ocasiones, una de las trabas más importantes en su implantación, la aplicación deberá ser fácil de instalar, configurar y utilizar incluso para usuarios inexpertos.

4.2. Tecnologías utilizadas

4.2.1. Lenguajes de programación

Para la implementación de las aplicaciones desarrolladas en este proyecto se han utilizado dos lenguajes de programación diferentes:

C: Lenguaje de programación creado en 1972 en los laboratorios Bell como evolución del lenguaje B. Es un lenguaje orientado a la implementación de sistemas operativos. Es el lenguaje de programación más popular para crear *Software* de sistemas aunque también se utiliza para construir aplicaciones.

Java: Es un lenguaje de programación orientado a objetos desarrollado por *Sun Microsystems* a principios de los años 90. Gran parte de su sintaxis es similar a C y C++ pero elimina muchas construcciones de bajo nivel y proporciona un modelo de objetos más simple de utilizar que C++.

El lenguaje C se ha elegido para implementar la autoridad de certificación así como los servicios de publicación y validación ya que estos elementos son los que más carga de proceso tienen (deben permitir atender varias solicitudes simultaneas de varios clientes) y es necesario que sean lo más eficientes posible.

El lenguaje Java se ha utilizado para construir las aplicaciones cliente. Este lenguaje se ha elegido para que las aplicaciones puedan ser portables ya que a priori no sabemos cual será la plataforma de ejecución del cliente. Puesto que el lenguaje Java es portable a cualquier plataforma donde se pueda ejecutar la maquina virtual de Java es un buen lenguaje para conseguir el objetivo de la portabilidad.

4.2.2. Bibliotecas

Para implementar el sistema se han utilizado varias bibliotecas que proporcionan las funcionalidades necesarias:

cryptlib[7]: cryptlib es un conjunto de herramientas de seguridad que permiten a los programadores añadir de manera sencilla servicios de cifrado y autenticación a sus aplicaciones. Dispone de una interfaz de alto nivel que permite utilizar mecanismos

de seguridad sin necesidad de conocer los detalles de los algoritmos que utiliza. Esta biblioteca incluye la implementación de muchos de los estándares estudiados en el capítulo 3 como por ejemplo: CMP, OCSP, X.509, CRL... La biblioteca proporciona interfaces para diferentes lenguajes de programación, entre ellos Java y C.

unixODBC [26]: Es una biblioteca que proporciona una API para utilizar ODBC¹ en plataformas que no sean Windows. ODBC es un estándar de acceso a bases de datos desarrollado por Microsoft Corporation. El objetivo de ODBC es hacer posible el acceso a cualquier dato desde cualquier aplicación sin importar el gestor de base de datos que almacene la información, por ejemplo MySQL, SQL Server, PostgreSQL... Esta biblioteca se usará para acceder a la base de datos y almacenar la información sobre certificados, CRLs y usuarios de la PKI. Existe otra biblioteca, llamada iodb [12], que proporciona funcionalidad similar pero se ha elegido unixODBC ya que su uso está más extendido.

confuse [6]: Es una biblioteca para analizar sintácticamente ficheros de configuración del tipo "opción=valor", aunque soporta configuraciones más complejas. Esta biblioteca se ha utilizado para analizar los ficheros de configuración de la autoridad de certificación y los servidores de validación y publicación.

openLDAP [18]: openLDAP es una implementación de código abierto del protocolo LDAP. Dispone de una API para interactuar con servidores LDAP así como un conjunto de aplicaciones para crear y administrar un servidor LDAP.

pcsc-lite [21]: Esta biblioteca proporciona una API que implementa la especificación PC/SC². Esta especificación permite la integración de tarjetas inteligentes en ordenadores personales. El objetivo de esta biblioteca es proporcionar una API que permita a los desarrolladores trabajar de forma uniforme con lectores de tarjetas de distintos fabricantes siempre que cumplan con la especificación. Los sistemas Windows ya disponen de una implementación instalada por defecto, pero para sistemas GNU/Linux es necesario utilizar esta biblioteca.

4.2.3. Herramientas

En este apartado comentaremos brevemente algunas de las herramientas que se han utilizado para realizar el desarrollo de este proyecto:

Netbeans [16]: Entorno de desarrollo, de código abierto, para la creación de aplicaciones. Tiene soporte para varios lenguajes de programación (C, C++,

¹Open Database Connectivity

²Personal Computer/Smart Card

Java...). Esta herramienta se ha utilizado para desarrollar todas las partes implementadas en Java y realizar el diseño de las interfaces gráficas.

Ant [2]: Herramienta basada en java para la construcción de proyectos: compilación, comprobación de dependencias, generación de objetos ejecutables... Es una herramienta con funcionalidad similar a Make pero sin alguno de sus problemas y con la portabilidad que proporciona el estar hecha en Java. Esta herramienta se ha utilizado para compilar y generar las distribuciones de las versiones finales de los programas realizados en Java.

GCC [11]: Colección de compiladores del proyecto GNU. Las partes del proyecto implementadas en C han sido compiladas utilizando el compilador de C incluido dentro de GCC.

Autotools [3, 4]: Conjunto de herramientas, también conocidas como *GNU build system*, diseñadas para ayudar a crear paquetes de código fuente portables a diferentes sistemas Unix. Estas herramientas se han utilizado para construir los paquetes de las partes del proyecto implementadas en C.

Como gestores de bases de datos se han realizado pruebas con dos alternativas diferentes:

PostgreSQL [23]: Gestor de bases de datos de código abierto. Tiene más de quince años de desarrollo y una gran reputación en cuanto a su fiabilidad, integridad de los datos que almacena y corrección.

MySQL [15]: Uno de los gestores de bases de datos más popular. Destaca por su rapidez, fiabilidad y facilidad de uso.

Puesto que el código de la aplicación de la autoridad de certificación utiliza la interfaz ODBC para el acceso a base de datos se podría utilizar cualquier gestor de base de datos como respaldo, por ello se han realizado pruebas con dos de los gestores de código abierto más conocidos y utilizados.

Aladdin Etoken RTE [1]: Entorno de ejecución para dispositivos PKCS#11 de Aladdin. Este software proporciona la capa PKCS#11 necesaria para comunicarse y realizar operaciones con dispositivos criptográficos Aladdin, en concreto con el *eToken PRO* que ha sido el utilizado para el desarrollo de estas aplicaciones.

LyX [14]: El primer procesador de textos WYSIWYM³. Es un procesador de textos que sigue la filosofía de redactar documentos basándose en la estructura del escrito en lugar de en su apariencia. Lyx utiliza el motor tipográfico T_EX para la generación del documento resultado. Dispone de opciones para exportar documentos en otros formatos como OpenDocument, L^AT_EX, HTML... Este procesador de textos se ha utilizado para redactar la memoria del proyecto.

³What You See Is What You Mean

4.3. Análisis del protocolo CMP

Para construir la solución software, objetivo de este proyecto, se ha decidido utilizar el protocolo CMP ya que es el más completo de los protocolos de gestión de certificados existentes. Para el desarrollo se ha estudiado el estándar CMP y se ha analizado y utilizado la implementación del protocolo CMP incluida dentro de la biblioteca criptográfica *cryptlib*.

4.3.1. Estructura general de los mensajes

Todos los mensajes definidos en el protocolo CMP tienen la siguiente estructura:

```
PKIMessage ::= SEQUENCE {  
    header          PKIHeader ,  
    body            PKIBody ,  
    protection      [0] PKIProtection OPTIONAL,  
    extraCerts  
[1] SEQUENCE SIZE (1..MAX) OF CMPCertificate  
                                OPTIONAL  
}
```

El campo *PKIHeader* contiene información común a todos los mensajes.

El campo *PKIBody* contiene información específica de cada uno de los tipos del mensaje.

El campo *PKIProtection*, si está presente, contiene bits que se utilizan para proteger el mensaje.

El campo *extraCerts* puede contener certificados que pueden ser útiles para el destinatario. Este campo puede contener, por ejemplo, certificados que sean necesarios para que un usuario final verifique su nuevo certificado en el caso de que la entidad emisora no sea una autoridad de certificación raíz.

En los siguientes apartados veremos en detalle cada uno de estos elementos.

4.3.2. Cabecera de los mensajes

Todos los mensajes definidos en este protocolo necesitan una cabecera para identificar cada transacción. La estructura de este elemento está definida en ASN.1 de la siguiente forma:

```
PKIHeader ::= SEQUENCE {  
    pvno                INTEGER  
{ cmp1999(1), cmp2000(2) },  
    sender              GeneralName ,  
    recipient           GeneralName ,  
    messageTime        [0] GeneralizedTime OPTIONAL,
```

protectionAlg	[1]	AlgorithmIdentifier	OPTIONAL,
senderKID	[2]	KeyIdentifier	OPTIONAL,
recipKID	[3]	KeyIdentifier	OPTIONAL,
transactionID	[4]	OCTET STRING	OPTIONAL,
senderNonce	[5]	OCTET STRING	OPTIONAL,
recipNonce	[6]	OCTET STRING	OPTIONAL,
freeText	[7]	PKIFreeText	OPTIONAL,
generalInfo	[8]	SEQUENCE SIZE (1..MAX) OF InfoTypeAndValue	OPTIONAL

}

PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String

El campo *pvno* debe tener el valor 2 que se corresponde con la última versión de la especificación de este protocolo.

El campo *sender* contiene el nombre del emisor del mensaje. Este campo, junto con el campo *senderKID*, debe ser suficiente para identificar la clave que se debe utilizar para verificar la protección del mensaje. Un caso particular es la solicitud inicial de certificación en la que el emisor del mensaje no sabe nada de sí mismo dentro del ámbito de la PKI; en este caso el valor del campo *sender* debe ser siempre *NULL*. En este caso el campo *senderKID* debe contener obligatoriamente un valor (número de referencia) que indique al receptor el secreto compartido que se debe utilizar para verificar el mensaje. En apartados sucesivos veremos como la agencia de registro se encargará de proporcionar este valor a los usuarios finales para identificar sus solicitudes iniciales.

El campo *recipient*, junto con el valor almacenado en el campo *recipKID*, si está presente, debería poder usarse para verificar la protección del mensaje.

El campo *protectionAlg* define el algoritmo utilizado para proteger el mensaje. Este campo debería omitirse siempre que no esté presente el campo *protection* para evitar confusiones.

El campo *transactionID* se utiliza para que el remitente de un mensaje relacione dicho mensaje con otros mensajes pertenecientes a una misma transacción que ya esté en curso. Este campo es necesario para las transacciones del protocolo que están formadas por más de un único par de mensajes de solicitud/respuesta. Este campo se utiliza de la siguiente forma:

- Para operaciones que están formadas solo por una solicitud y una respuesta el cliente puede establecer un valor para este campo. Si el servidor recibe un mensaje con este campo relleno, entonces copiará el mismo valor en su respuesta. Sin embargo si el servidor recibe un mensaje sin este campo relleno, será él el que podrá establecer un valor para este campo si lo desea.
- Para operaciones que impliquen más de un mensaje de solicitud y respuesta el procedimiento es el siguiente: El cliente debe generar un identificador para la transacción para el primer mensaje. Cuando el servidor reciba la respuesta debe

copiar el valor establecido por el cliente obligatoriamente en su respuesta. En caso de que el cliente no establezca un valor para este campo, el servidor deberá generar un valor y establecerlo en la respuesta para utilizarlo en todos los mensajes de la transacción.

En todos los casos en los que se asigne un valor para el campo *transactionID*, un cliente no puede tener mas de una transacción en curso con el mismo identificador a la vez con el mismo servidor. Si un servidor recibe el primer mensaje de una transacción con un identificador que no es único para él en ese momento, es decir, que se está tratando una solicitud con el mismo identificador, el servidor debe devolver un mensaje de error al cliente con el valor *transactionIdInUse*. Para evitar este problema, el estándar recomienda que todos los clientes rellenen este campo con 128 bits pseudoaleatorios para reducir la probabilidad de que exista una transacción en uso en el servidor.

Los campos *senderNonce* y *recipNonce* protegen los mensajes contra ataques de *REPLAY*. El campo *senderNonce* contendrá, normalmente 128 bits de datos pseudoaleatorios generados por el emisor. El valor del campo *recipNonce* se copiará del campo *senderNonce* del mensaje anterior en la transacción.

El campo *messageTime* contiene la fecha en la que el emisor envió el mensaje.

El campo *freeText* se puede utilizar para enviar al destinatario un mensaje legible por un humano.

4.3.3. Estructuras de datos

Antes de especificar el formato que define el contenido de los mensajes CMP definiremos un conjunto de estructuras de datos que son comunes a los diferentes mensajes del protocolo.

4.3.3.1. Valores solicitados para el certificado

Algunos mensajes del protocolo permiten que el solicitante del mensaje indique algunos valores que desee que estén presentes en el certificado resultante. La estructura *CertTemplate* permite que la entidad final especifique el valor de todos los campos del certificado que desee. La estructura de este campo es idéntica a la de un certificado X.509 pero todos los campos son opcionales.

Aunque el solicitante especifique cualquiera de estos valores, la autoridad de certificación puede modificar y decidir los valores que considere oportunos sin necesidad de mantener los valores propuestos por la entidad final.

4.3.3.2. Valores cifrados

Cuando es necesario enviar valores cifrados dentro de un mensaje del protocolo se utilizará la estructura de datos *EncryptedValue*. Para utilizar esta estructura de datos es

necesario que tanto el emisor como el receptor del mensaje sean capaces de cifrar y descifrar respectivamente, por tanto, es necesario que tanto el emisor como el receptor sean capaces de generar y compartir un secreto compartido. Si el destinatario ya dispone de una clave privada capaz de descifrar, entonces el campo *encSymmKey* contendrá la clave de sesión cifrada con la clave pública del destinatario y evitar así disponer de un secreto compartido de antemano.

4.3.3.3. Códigos de error

Todos los mensajes del protocolo pueden incluir un código con información adicional con los siguientes valores:

```
PKIStatus ::= INTEGER {  
    accepted                (0),  
    grantedWithMods         (1),  
    rejection                (2),  
    waiting                  (3),  
    revocationWarning       (4),  
    revocationNotification  (5),  
    keyUpdateWarning        (6)  
}
```

Los mensajes de respuesta también pueden incluir un código para proporcionar más información acerca de las causas que pueden provocar un error. El conjunto de valores definido en el estándar es el siguiente:

```
PKIFailureInfo ::= BIT STRING {  
    badAlg                (0),  
    badMessageCheck       (1),  
    badRequest            (2),  
    badTime               (3),  
    badCertId             (4),  
    badDataFormat         (5),  
    wrongAuthority        (6),  
    incorrectData         (7),  
    missingTimeStamp      (8),  
    badPOP                (9),  
    certRevoked           (10),  
    certConfirmed         (11),  
    wrongIntegrity        (12),  
    badRecipientNonce     (13),  
    timeNotAvailable      (14),  
    unacceptedPolicy      (15),  
    unacceptedExtension   (16),  
    addInfoNotAvailable   (17),  
}
```



```

        badSenderNonce      (18),
        badCertTemplate     (19),
        signerNotTrusted    (20),
        transactionIdInUse  (21),
        unsupportedVersion   (22),
        notAuthorized        (23),
        systemUnavail        (24),
        systemFailure        (25),
        duplicateCertReq     (26)
    }

    PKIStatusInfo ::= SEQUENCE {
        status          PKIStatus ,
        statusString    PKIFreeText  OPTIONAL,
        failInfo        PKIFailureInfo OPTIONAL
    }

```

4.3.3.4. Estructuras de representación de las pruebas de posesión

En todas las solicitudes de certificación es necesario que el usuario proporcione pruebas de posesión de la clave privada para la cual está solicitando la certificación.

Cuando el uso de la clave que se desea certificar sea realizar firmas digitales, las pruebas de posesión se realizarán mediante el uso de la estructura *POPOSigningKey* cuyo formato es el siguiente:

```

    POPOSigningKey ::= SEQUENCE {
        poposkInput          [0] POPOSigningKeyInput OPTIONAL,
        algorithmIdentifier   AlgorithmIdentifier ,
        signature             BIT STRING
    }

```

En esta estructura el campo *poposkInput* contiene los datos a ser firmados, el campo *algorithmIdentifier* contiene el OID del algoritmo de firma a ser utilizado y el campo *signature* contiene la firma digital de la codificación DER del campo *poposkInput*. Mediante el uso de esta estructura se puede comprobar, verificando la firma digital, que el usuario está en posesión de la clave que desea certificar.

En caso de que el uso de clave sea el cifrado de datos, entonces las pruebas de posesión de la clave privada pueden realizarse de alguna de estas tres formas:

- **Inclusión de la clave privada:** Mediante la inclusión de la clave privada (cifrada) en el mensaje, la autoridad de certificación puede asegurar que la entidad final está en posesión de la clave privada que desea certificar. Este mecanismo solo es válido cuando la CA pueda, o deba, disponer de copias de la clave privada de los usuarios de la PKI.

- Método indirecto: Este mecanismo está basado en que la autoridad de certificación devuelva el nuevo certificado al usuario cifrado para que éste solo sea capaz de recuperarlo y usarlo si está en posesión de la clave privada que certifica. El certificado estará cifrado con una clave simétrica elegida al azar por la autoridad de certificación y está estará cifrada a su vez con la clave pública para la cual el sujeto está solicitando el nuevo certificado.

4.3.4. Protección de los mensajes

Algunos de los mensajes involucrados en el proceso de gestión de certificados utilizando CMP deberán tener protección de integridad. Además, en el caso de que la protección se realice mediante el uso de algoritmos de clave asimétrica y el componente público de la clave haya sido certificado previamente los mensajes estarán, además, autenticados. En caso de que la clave no esté certificada, no se podrá realizar la autenticación automáticamente aunque si se podrá realizar por medios externos al protocolo.

La estructura que representa la protección de un mensaje del protocolo es la siguiente:

```
PKIProtection ::= BIT STRING
```

y la entrada para calcular este valor es la codificación DER de la siguiente estructura de datos:

```
ProtectedPart ::= SEQUENCE {  
    header    PKIHeader ,  
    body      PKIBody  
}
```

La protección de los mensajes del protocolo CMP puede realizarse mediante dos mecanismos: firma digital o códigos de autenticación de mensajes.

4.3.4.1. Protección basada en secretos compartidos

En este modelo de autenticación, el emisor y el receptor comparten de antemano un clave secreta (posteriormente veremos como la agencia de registro provee al usuario final de esta información). En este caso, el campo *PKIProtection* contiene un valor MAC y el campo *protectionAlg* está definido por la siguiente estructura:

```
id-PasswordBasedMac OBJECT IDENTIFIER ::= {1 2 840 113533 7 66 13}  
PBMPParameter ::= SEQUENCE {  
    salt          OCTET STRING,  
    owf           AlgorithmIdentifier ,  
    iterationCount INTEGER,  
    mac           AlgorithmIdentifier  
}
```

En esta estructura el significado de cada uno de los campos es el siguiente:

salt: Este valor se concatenará con el secreto compartido entre las dos entidades y formará la entrada de la función *hash*.

owf: Este valor indica la función hash que se debe utilizar para realizar el HMAC.

iterationCount: Número entero que indica el número de iteraciones que se deben realizar utilizando la función *hash* definida en *owf*. La primera entrada de la función será la concatenación del valor *salt* con el secreto compartido y las llamadas sucesivas utilizarán como entrada la salida de la invocación anterior a la función *hash*.

mac: Identificador del algoritmo utilizado para realizar el HMAC.

El estándar recomienda que los valores de todos estos campos permanezcan constantes de un mensaje a otro dentro de la misma transacción.

Este mecanismo de protección será el que se utilizará para enviar el mensaje de solicitud inicial de certificación a la autoridad de certificación ya que en ese momento no dispone de ninguna clave certificada que pueda utilizar para autenticar su solicitud.

4.3.4.2. Protección basada en firmas digitales

Este tipo de protección se puede utilizar cuando el emisor del mensaje dispone de una clave pública certificada y reconocida por el destinatario que le permite firmar los mensajes del protocolo. Este mecanismo de protección puede utilizarse en todos los mensajes, excepto en la solicitud inicial de certificación; de hecho, el estándar obliga a que algunos de los mensajes estén protegidos usando firma digital, como por ejemplo la solicitudes de revocación, y no permite que estén autenticados usando secretos compartidos.

En este caso el campo *PKIProtection* contendrá el valor de la firma digital de los datos y el campo *protectionAlg* será el OID de un algoritmo de firma digital como por ejemplo *md5WithRSAEncryption*.

4.4. Autoridad de certificación

La autoridad de certificación es la parte más importante dentro de una PKI ya que es la encargada de la generación de los certificados digitales que es el fin último de una PKI. La implementación de la autoridad de certificación se ha realizado en el lenguaje C y consta dos elementos: las aplicaciones de inicialización y el servidor CMP. La aplicación se ha desarrollado para que funcione en el sistema operativo GNU/Linux aunque al estar hecha en C estándar es portable a otros sistemas.

4.4.1. Inicialización del sistema

La autoridad de certificación necesita dos elementos para su funcionamiento: una identidad digital que será la que utilice para firmar los certificados que emita y un sistema de almacenamiento para guardar los certificados emitidos, las CRLs, la información de los usuarios, las solicitudes pendientes...

4.4.1.1. Configuración previa

Todas las aplicaciones desarrolladas en el proyecto en las que se pueden generar y/o utilizar claves asimétricas disponen de la opción de utilizar un dispositivo hardware PKCS #11 para su gestión. Como vimos en el capítulo 3.2.2 estos dispositivos necesitan una biblioteca, proporcionada por el fabricante, que implemente el acceso físico al dispositivo y sobre el que se apoya la interfaz genérica PKCS #11. Para que las aplicaciones de la autoridad de certificación puedan hacer uso de estos dispositivos es necesario realizar una configuración previa. Para ello se ha escrito la aplicación *register_pkcs11* que recibe como parámetro la ruta del fichero que contiene la biblioteca proporcionada por el fabricante y configura la biblioteca *cryptlib* para que pueda acceder a los dispositivos que utilicen esa biblioteca. Una vez realizado este proceso, el token PKCS #11 se podrá utilizar en el resto de aplicaciones de la autoridad de certificación.

Para el almacenamiento de los certificados y CRLs generados por la autoridad de certificación es necesario disponer de una base de datos que, como ya hemos visto, puede estar en cualquier gestor de base de datos al cual se pueda acceder mediante la interfaz ODBC. Configurar una base de datos para que sea accesible mediante ODBC utilizando unixODBC es un proceso sencillo: Primero se debe crear la base de datos utilizando las herramientas propias del gestor de base de datos, por ejemplo *MySQL Administrator*. Una vez creada la base de datos, unixODBC dispone de una aplicación gráfica llamada *ODBCConfig* que permite añadir un nuevo recurso ODBC simplemente asignándole un nombre identificativo, para referirnos al recurso posteriormente, e indicando la ruta y el puerto del servidor de base de datos y el nombre de la base de datos que se ha creado previamente. En el apartado 4.4.1.3 veremos como se crea la estructura de tablas que necesita nuestra autoridad de certificación y como se accede después a ella utilizando este recurso ODBC.

4.4.1.2. Generación de la identidad digital

Para crear la identidad de la autoridad de certificación se ha desarrollado una aplicación denominada *generate_root_ca* que permite crear un par de claves RSA, con un certificado X.509 asociado a la clave pública y almacenar todo ello en un fichero PKCS #15 o bien en un dispositivo hardware PKCS #11. La interfaz de la aplicación es a través de la línea de comandos e interactivamente solicitará los datos necesarios para construir la identidad. En la figura 4.1 en la página 52 se puede ver la secuencia de generación de una identidad

para una autoridad de certificación raíz y su almacenamiento en un fichero PKCS #15. El procedimiento para utilizar un dispositivo PKCS #11 es exactamente el mismo pero hay que invocar al comando con la opción `-pkcs11`.

Este paso no es necesario realizarlo cuando queramos construir una autoridad de certificación subordinada ya que su certificado X.509 no será autogenerado sino que lo generará otra autoridad (en el punto 4.5 veremos como se genera la identidad digital de una autoridad de certificación subordinada) .

4.4.1.3. Sistema de almacenamiento

En el apartado 4.4.1.1 hemos visto como configurar un recurso ODBC para utilizarlo como almacenamiento para la autoridad de certificación. En este apartado veremos como se crea la estructura inicial de la base de datos así como las tablas que la componen. Para crear las tablas en el sistema gestor de bases de datos se ha implementado la aplicación *initialize_ca_database* que, de manera interactiva, solicita los parámetros necesarios para acceder a la base de datos y crear las tablas necesarias. En la figura 4.2 en la página 53 se puede ver el resultado de crear la base de datos sobre un recurso ODBC llamado *myodbc-test*. Si la base de datos a la que hace referencia el recurso ODBC no está vacía, el programa informará acerca de esta situación y no generará ninguna tabla en la base de datos.

La estructura de la base de datos está definida por la biblioteca *cryptlib* y contiene un conjunto de tablas que permiten automatizar el proceso de generación y manejo de certificados utilizando el protocolo CMP. La base de datos está formada por cinco tablas cuyo contenido es el siguiente:

pkiUsers: Esta tabla contiene información acerca de los usuarios que han sido dados de alta en el sistema por la agencia de registro y que están autorizados a realizar solicitudes de certificación con la autoridad de certificación.

certRequests: En esta tabla se almacenan las solicitudes de certificación que están pendientes de ser procesadas por la autoridad de certificación.

certificates: Esta tabla contiene todos los certificados que están activos en el momento actual.

CRLs: En esta tabla se almacenan las CRLs que han sido emitidas por la autoridad de certificación. También se almacenan las CRLs que se van construyendo de manera temporal: el sistema genera una CRL cada vez que se revoca un certificado que solo necesitará ser firmada cuando vaya a ser emitida.

certLog: Esta tabla almacena un registro de las operaciones que va realizando la autoridad de certificación: revocaciones, generación de nuevos certificados... y permite mantener un log de las mismas así como recuperar operaciones que hayan podido quedarse a medias por problemas de conexión con la base de datos.

```
$ ./generate_root_ca
This program generate a key pair and a selfsigned
  certificate for a root CA in a PKCS#15 file or PKCS
  #11 hardware token

INFO: Use --pkcs11 to generate the key in a PCKS#11
  compatible device

Enter keysize in bits: 2048
Enter label to identify keypair: root-label
Generating keypair, please wait... Done
Enter filename of new PKCS#15 keyset to save keypair:
  root-ca.p15
Enter password to protect keypair:

Enter data for CA certificate:
Common name (CN) (Mandatory): Root CA
Country (C) (Mandatory): ES
Organization (O): CriptoLab Organization
Unit (OU): Personal
State or Province (SP):
Locality name (L):
HASH signature algoritm (SHA, SHA2, RIPEMD-160) (
  Mandatory): SHA
Certificate validity in days (Mandatory): 365
Max lenght constraint: 0
Certificate policy OID:
CSP uri:
Uri for OCSP service:
Uri of CRL: http://criptolab.org/crl.der
Writing certificate to keystore, please wait...Done
Filename to write DER certificate: root-ca.der
Program finished OK
```

Figura 4.1: Generación de la identidad de una autoridad de certificación raíz

```
./initialize_ca_database
This program initialize database for a new CA
Database must exist and must be empty
Enter ODBC data source name: myodbc-test
Enter username: dani
Enter password:

Database generated OK
```

Figura 4.2: Creación de la estructura de la base de datos

4.4.2. Servidor CMP

En el capítulo 2.3.2 se detallaron los protocolos de gestión de certificados existentes y como se puede observar el protocolo CMP es el más avanzado y permite realizar todas las operaciones que debería proporcionar una autoridad de certificación. Por tanto, este desarrollo utilizará este protocolo para proporcionar a los usuarios los mecanismos necesarios para gestionar sus certificados digitales. El estándar del protocolo CMP no especifica el protocolo de comunicaciones que se debe utilizar para el envío y recepción de mensajes, pero si que recomienda el protocolo de transporte HTTP para comunicar el servidor con los clientes. Para ello se ha desarrollado una aplicación llamada *ca_server* que implementa un servidor CMP multihilo. Esta aplicación es capaz de recibir solicitudes de generación de un certificado inicial, de revocación, de recertificación y de actualización del par de claves. Para ejecutar la aplicación se le debe pasar como parámetro un fichero con la configuración, como se puede ver en la figura 4.3 en la página siguiente. La aplicación esperará entonces a recibir una solicitud CMP que procesará y devolverá el resultado correspondiente. En los siguientes apartados veremos como se procesa cada una de las solicitudes CMP definidas en el estándar.

4.4.2.1. Solicitud inicial de certificación

Esta solicitud es la primera que realiza un usuario de una PKI. En este momento el usuario final no ha tenido contacto previo con la autoridad de certificación por lo tanto no dispone de ningún certificado digital de esta autoridad. Por tanto, esta solicitud no estará firmada sino protegida por un HMAC como está definido en el estándar.

Cuando el servidor CMP recibe una solicitud de certificación inicial realiza los siguientes pasos para procesarla:

1. Obtener el identificador de usuario de la solicitud y comprobar que existe en la tabla *pkiUsers*.

```
# Port to listen
port = 5000
# Private keyset options
#use-pkcs11 = true
keyset_file = root-ca.p15
keyset_pass = root
keyset_label = root
# Number of threads
max_threads = 10
# Time to expire certs
expire_timeout = 600
# Database options
# Format: [user:pass@]odbc-source
keystore = dani:password@myodbc-test
# Certificate validity in days
certificate_validity = 1000
```

Figura 4.3: Fichero de configuración de la autoridad de certificación

2. Obtener la clave de autenticación inicial asociada a ese identificador de usuario y verificar el HMAC. Si no se verifica se devolverá un mensaje de error.
3. Si el HMAC es correcto la autoridad de certificación generará y firmará un certificado digital X.509 asociado a la clave pública del usuario (que estará indicada en la solicitud). El valor de algunos de los campos presentes en el certificado resultante habrá sido establecido por la agencia de registro cuando se produjo la verificación de la identidad real del usuario.
4. Almacenar el certificado en la base de datos.
5. Devolver el certificado digital al usuario como parte de la respuesta del protocolo CMP.

4.4.2.2. Revocación de certificados

El estándar CMP solo permite la solicitud de revocación de certificados por parte del usuario si las solicitudes están firmadas con la clave privada que se desea revocar. En el estándar existe un apéndice que propone, aunque no lo recomienda, proporcionar al usuario una contraseña para poder revocar un certificado en caso de que la revocación se deba a una pérdida de la clave privada y sea incapaz de firmar la solicitud de revocación. Esta contraseña se utiliza para autenticar la solicitud mediante un HMAC de la misma forma que en las solicitudes iniciales de certificación.

Para el desarrollo de este proyecto se ha optado por permitir solo revocaciones firmadas pero se ha desarrollado una agencia de revocación, que veremos mas adelante, en la que un usuario pueda solicitar la revocación de un certificado cuya clave privada asociada ya no esté disponible. El procedimiento a realizar cuando se recibe una solicitud de revocación CMP es el siguiente:

1. Verificar la firma de la solicitud de revocación. Puesto que la solicitud debe estar firmada con la misma clave que se desea revocar, la firma se verifica con la clave pública que aparece en la solicitud. Este mecanismo proporciona pruebas de posesión de la clave privada y garantiza que solo el usuario que está en posesión de la clave privada puede revocarla.
2. Comprobar que el certificado que se desea revocar está en la base de datos.
3. Eliminar el certificado de la base de datos
4. Generar la estructura de la próxima CRL añadiendo este certificado para que esté preparada para la siguiente publicación y solo sea necesaria añadir la fecha de publicación y realizar la firma.
5. Enviar al usuario la respuesta CMP indicando que la revocación ha sido correcta.

4.4.2.3. Solicitud de recertificación

Las solicitudes de recertificación se producen cuando un usuario desea generar un nuevo certificado para el mismo par de claves generalmente cuando el certificado de que dispone ahora esta próximo a caducar. Como normal general, el estándar CMP recomienda que cuando se recertifique una clave se revoque el certificado actual para evitar tener dos certificados que solo se diferencien en el número de serie y en la fecha de validez. El procedimiento para procesar una solicitud de recertificación es el siguiente:

1. Verificar la firma de la solicitud de recertificación. Puesto que se está solicitando un certificado para una clave ya existente, la firma de la solicitud deberá verificarse con la clave que aparece en la propia solicitud. Este mecanismo proporciona pruebas de posesión de la clave privada y garantiza que solo el usuario que está en posesión de la clave privada puede solicitar la recertificación de la clave.
2. Comprobar que el certificado que se desea recertificar está en la base de datos.
3. Generar un nuevo certificado, que en este caso será para la misma clave pública, pero con distinta validez y número de serie.
4. Añadir el certificado a la base de datos.

5. Realizar la revocación del certificado anterior. El procedimiento para realizar la revocación será idéntico al definido en el apartado 4.4.2.2.
6. Enviarle al usuario el nuevo certificado como parte de la respuesta CMP.

4.4.2.4. Solicitud de renovación de clave

Una solicitud de renovación de claves se produce cuando un usuario desea actualizar su par de claves y obtener un nuevo certificado para ese par de claves pero con los mismos valores para los campos que los del certificado que posee actualmente. El procedimiento definido en el protocolo CMP para renovar una clave es el siguiente:

1. Verificar la firma de la solicitud de renovación. La firma de la solicitud debe ser verificada con la clave pública del nuevo par de claves para el que se está solicitando el certificado. Este mecanismo proporciona pruebas de posesión de la clave privada y garantiza que solo el usuario que está en posesión de la clave privada cuya clave pública se desea certificar.
2. La solicitud de renovación contiene también una firma interna de la solicitud realizada con la clave privada antigua que permite asegurar que solamente el usuario está en posesión de la clave privada que se desea actualizar. Este mecanismo se utiliza para comprobar que el usuario está autorizado a realizar la solicitud de renovación de la clave.
3. Comprobar que el certificado cuya clave se desea renovar está en la base de datos.
4. Generar un nuevo certificado para la nueva clave pública incluida en la solicitud.
5. Almacenar el certificado en la base de datos.
6. Enviar el nuevo certificado al usuario como parte de la respuesta del protocolo CMP.

4.4.2.5. Mantenimiento de la base de datos

En los apartados previos hemos visto como se realizan las tareas de gestión de certificados: inicializaciones, revocaciones... Estas operaciones generan inserciones y borrados de campos en la base de datos. Es importante realizar operaciones para mantener consistentes los valores almacenados en la base de datos. El servidor CMP realiza dos tipos de operaciones para conseguir este objetivo:

- Eliminación de certificados caducados: El servidor CMP comprobará cada cierto tiempo la existencia de certificados caducados. El servidor eliminará todos los certificados que estén caducados ya que su almacenamiento no tiene ninguna utilidad excepto si se desea guardar un registro con propósitos históricos.

- Eliminación de solicitudes fallidas: Si se produce algún error al procesar alguna solicitud de certificación puede ocurrir que se quede almacenada en la base de datos sin que se hay procesado. Para eliminar estas solicitudes, cuando se arranca el servidor CMP, se analizará la tabla que guarda los logs de las operaciones y se eliminarán estas solicitudes fallidas.

4.5. Agencia de registro y revocación

La agencia de registro es un elemento fundamental dentro de una infraestructura de clave pública ya que es la que permite verificar la identidad real de los usuarios y proporcionarles la información necesaria para realizar las gestiones oportunas con la autoridad de certificación. Otro elemento importante es la agencia de revocación que permite a un usuario realizar la revocación de alguno de sus certificados cuando no dispone de la clave privada para firmar la solicitud.

Para implementar estas dos agencias se ha desarrollado una única aplicación que permite manejar los dos servicios. La decisión de realizar una única implementación se ha tomado no por la similitud de los dos servicios ya que las dos agencias pueden, o incluso deben, ser dos elementos independientes, sino por la similitud en el código para implementarlas y reducir el número de herramientas necesarias para construir una infraestructura de clave pública.

Esta aplicación ha sido desarrollada en el lenguaje Java para conseguir la mayor portabilidad. La portabilidad de esta aplicación es importante ya que, a priori, no sabemos el entorno en el que se puede ejecutar. La agencia de registro se puede ver como una aplicación que se ejecuta en una maquina fija y dedicada en una oficina a la que acuden los usuarios a solicitar el registro en una infraestructura de clave pública, o como una aplicación móvil, instalada en un portátil, que permite que sea el operador el que acuda a realizar las inscripciones a otros lugares. La aplicación está implementada utilizando la biblioteca gráfica Swing [25] para dibujar las ventanas, que también es portable, y permite mostrar ventanas en todas las plataformas que pueden ejecutar la máquina virtual de Java.

4.5.1. La agencia de registro

La agencia de registro es responsable de verificar la identidad real de los usuarios que desean pasar a formar de la infraestructura de clave pública y dotar a los mismos de la información necesaria para utilizar los servicios disponibles por dicha PKI.

La aplicación de la agencia de registro permite a un operador de la agencia de registro añadir y eliminar usuarios de una PKI. Para ello la aplicación debe tener permisos de escritura sobre la tabla *pkiUsers* de la base de datos de la autoridad de certificación. Una vez que el operador presenta las credenciales que le permiten ponerse en contacto con la

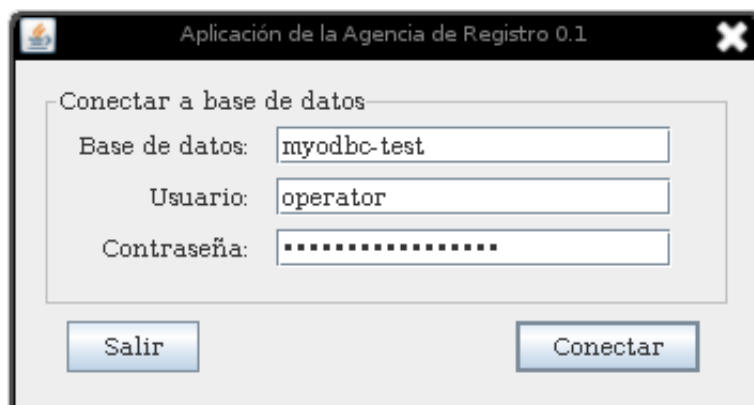


Figura 4.4: Ventana de conexión con la base de datos

base de datos podrá realizar dos tipos de operaciones: dar de alta a usuarios y eliminar usuarios. En la figura 4.4 se puede ver la ventana que permite al operador introducir los datos necesarios para autenticarse frente al servidor de base de datos.

4.5.1.1. Alta de usuarios

Una vez verificada la identidad del solicitante, el operador de la agencia de registro deberá cumplimentar un formulario con información acerca de usuario que servirá como plantilla para los certificados que sean emitidos posteriormente para ese usuario. Los datos que se deben rellenar obligatoriamente, como se puede ver en la figura 4.5 en la página siguiente, son: el nombre común del usuario y el país. Entre los datos opcionales están: la organización, la unidad organizativa, la provincia, la localidad y la dirección de correo electrónico. Una vez establecido el valor de estos campos se debe seleccionar si el certificado es para un usuario final o para una autoridad subordinada. En función del tipo de usuario que se desee añadir se deberán cumplimentar unos campos u otros.

- **Usuarios finales:** Los usuarios finales son aquellos cuyo certificado no tiene presente el valor isCA dentro de la extensión restricciones básicas. Para dar de alta un usuario final la agencia de registro debe rellenar, además, los valores indicados anteriormente, los valores deseados para el uso de la clave y el uso extendido de la clave. En la figura 4.6 en la página siguiente se puede ver una captura de pantalla del proceso.
- **Autoridades de certificación subordinadas:** Una autoridad de certificación subordinada es aquella que dispone de un certificado que tiene el valor isCA dentro de la extensión restricciones básicas y que no está autofirmado, sino firmado por otra autoridad de certificación. Esta última autoridad puede ser otra autoridad de

Añadir usuario

Datos del usuario

Nombre Común: **País:**

Organización:

Unidad Organizativa:

Provincia o Estado:

Localidad:

e-mail:

Tipo de certificado

☒ Entidad final ☐ Sub CA

* Los campos marcados en rojo son obligatorios

Figura 4.5: Ventana de inicialización del usuario

Añadir usuario final

Uso de la clave

☒ Firma digital ☐ No repudio

☒ Cifrado de claves ☐ Cifrado de datos

Uso extendido de la clave

☐ Firma de código

☐ Protección de correo electrónico

☐ Firma de OCSP

☐ Autenticación de servidor SSL

☐ Autenticación de cliente SSL

☐ Sellado de tiempos

☐ IPSEC - Sistema final

☐ IPSEC - Tunel

☐ IPSEC - Usuario

☐ Servicio de directorio

Figura 4.6: Ventana de datos de inicialización de un usuario final

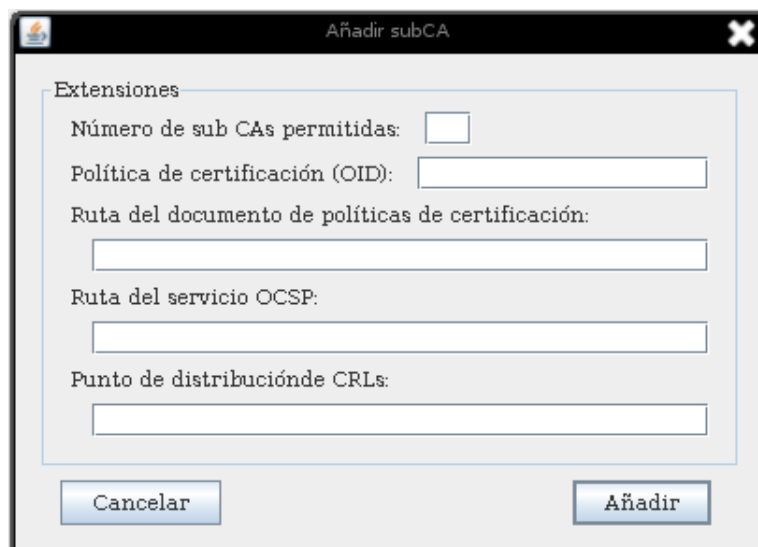


Figura 4.7: Ventana de datos de inicialización de una autoridad subordinada

certificación subordinada o una autoridad de certificación raíz. Cuando se crea una autoridad de certificación subordinada se pueden rellenar, como se muestra en la figura 4.7, los siguientes datos:

- Número de autoridades de certificación subordinadas que pueden aparecer a continuación de ésta en una cadena de validación.
- OID de la política de certificación para esta autoridad.
- Ruta al documento de políticas de certificación.
- Ruta en la que se publicará la CRL de la nueva autoridad.

Una vez dado de alta el nuevo usuario, la agencia de registro deberá proporcionar al usuario los datos que necesita para ponerse en contacto con la autoridad de certificación y realizar solicitudes utilizando el protocolo CMP. Para ello, la agencia de registro proporcionará al usuario un fichero con los siguientes datos:

- Ruta del servidor CMP: Este campo contiene una URL en la cual el servidor CMP de la autoridad de certificación atiende las solicitudes de certificación.
- Puerto de servidor CMP: Número de puerto del servidor CMP de la autoridad de certificación.
- Certificado de la autoridad: URL en la que se puede obtener el certificado de la autoridad de certificación. Como se vio en el punto 3.3.3 para realizar la solicitud inicial de certificación es necesario obtener de manera segura una copia del certificado de la autoridad de certificación.

- Nombre de usuario: Este campo se incluirá en la solicitud inicial de certificación y servirá para que la autoridad de certificación pueda identificar la solicitud.
- Contraseña: La solicitud inicial de certificación deberá estar autenticada mediante un *HMAC* utilizando esta contraseña. La autoridad de certificación verificará la solicitud obteniendo la contraseña asociada al usuario indicado en el campo anterior y comprobando el *HMAC*.

En el capítulo 4.6 veremos como debe utilizar el usuario este fichero en la aplicación cliente para generar sus solicitudes de gestión de certificados.

4.5.1.2. Baja de usuarios

La segunda operación que puede realizar un operador de la agencia de registro es la baja de usuarios. Esta operación puede ser solicitada por un usuario cuando ya no desee formar parte de la agencia de registro o bien puede iniciarse directamente por la agencia de registro cuando el usuario ya no esté autorizado a realizar más operaciones con la autoridad de certificación. En cualquiera de los dos casos, una vez realizada la operación, el usuario no podrá realizar ninguna otra operación con la autoridad de certificación, ni solicitudes, revocaciones o recertificaciones. Por otro lado, los certificados que estén dentro del periodo de validez seguirán siendo validos y no serán revocados automáticamente. La revocación de un certificado de un usuario que haya sido dado de baja de la infraestructura de clave pública ya no podrá ser realizada por dicho usuario sino que deberá presentarse en la agencia de revocación para tramitar la revocación del certificado.

Para eliminar un usuario de la infraestructura, el operador de la agencia de registro deberá acceder a la base de datos de la PKI (de igual manera que para realizar el alta de un usuario). Una vez conectado con la base de datos el operador recibirá una lista con los usuarios que están dados de alta en la PKI. En la lista el operador deberá seleccionar el usuario que desee eliminar y éste será borrado del sistema. En la figura 4.8 en la página siguiente se puede ver una captura de pantalla de la aplicación en la que se muestra este procedimiento.

4.5.2. La agencia de revocación

La agencia de revocación es la aplicación encargada de gestionar las revocaciones de certificados que no están iniciadas directamente por la aplicación de usuario. Los tipos de revocaciones que realiza la agencia son:

- Revocaciones de certificados solicitadas por el usuario debido a que la clave privada asociada con el certificado ya no está disponible.



Figura 4.8: Ventana de baja de usuarios

- Revocaciones iniciadas directamente por la propia infraestructura de clave pública debidas, por ejemplo, a un cambio de afiliación del usuario, o un cese del mismo en la organización indicada en el certificado.

Cuando se genera una revocación de este tipo se generará, de manera automática, la información necesaria para que los servicios de validación proporcionen de manera adecuada el estado del certificado cuando sea consultado por otros usuarios. Cualquier certificado que haya sido revocado ya no se podrá utilizar para realizar renovaciones, revocaciones, ni recertificaciones.

Como ya hemos visto, la agencia de revocación se ha implementado como parte de la agencia de registro por su similitud en el código más que por su parecido en funcionalidad. Para realizar una revocación el operador deberá conectarse con la base de datos de igual forma que para dar de alta o de baja a un usuario. Una vez conectado se presentará al operador una ventana, como la que se puede ver en la figura 4.9 en la página siguiente, en la que deberá rellenar los campos que desee para realizar una búsqueda de los certificados validos que han sido emitidos por la autoridad de certificación y que cumplan las condiciones especificadas en dichos campos.

Una vez establecidos los criterios de búsqueda se mostrará al operador una lista con los certificados que cumplen dichos criterios o un mensaje indicando que no se encuentra ningún certificado que cumpla los criterios. Cuando se encuentran certificados que cumplan las condiciones, el operador podrá examinar los detalles de cualquiera de los certificados, así como revocar alguno de ellos como se puede ver en la figura 4.10 en la página siguiente.

Buscar certificados

Criterios de búsqueda

Nombre Común: País:

Organización:

Unidad Organizativa:

Provincia o Estado:

Localidad:

e-mail:

Cancelar Buscar

Figura 4.9: Ventana de búsqueda de certificados para revocar

Nombre	Organización	Unidad Organizativa	Provincia	Localidad	País
Usuariol	CriptoLab				ES

Cancelar Ver Certificado Revocar

Figura 4.10: Ventana de selección de certificados a revocar

4.6. Aplicación de usuario

La aplicación de usuario es el software que ejecutan los usuarios de la infraestructura de clave pública una vez que han sido dados de alta por la agencia de registro. Puesto que no sabemos a priori en que tipo de sistemas se ejecutará la aplicación es importante realizar una implementación portable. Para conseguir este objetivo la aplicación está implementada en Java y utiliza la biblioteca Swing que permite que la aplicación se ejecute en cualquier plataforma en la que se pueda ejecutar la máquina virtual de Java.

Un gran número de soluciones de PKI existentes están diseñadas para que la aplicación cliente sea el navegador web del usuario. Este modelo, basado en un navegador web, tiene la ventaja de que utiliza el almacén de identidades de que disponen la mayoría de los navegadores web para almacenar identidades. Esto permite al usuario utilizar directamente sus identidades digitales, por ejemplo, para autenticarse frente a algún servidor web. Un claro ejemplo de este tipo de diseños es el proyecto Ceres [5] de la FNMT: esta autoridad de certificación utiliza el navegador web del cliente para almacenar su identidad una vez generada para que pueda utilizarla directamente para autenticarse en algunos servicios como la Seguridad Social o la Agencia Tributaria. Por otra parte, este tipo de infraestructuras, en algunos casos, provocan confusión en los usuarios ya que no ven claramente la identidad digital que han generado ya que se almacena en su navegador de forma demasiado transparente. Infraestructuras como ésta, orientadas a usuarios con conocimientos básicos de informática y prácticamente nulos en seguridad, son necesarias, pero para otro tipo de escenarios hay que valorar la simplicidad frente a la seguridad.

Una característica importante a señalar de la aplicación es cómo realiza el almacenamiento de identidades. Al tratarse de una aplicación independiente no utiliza el almacén de identidades del navegador ni ningún otro almacén global disponible en el sistema operativo (como los que están disponibles en Windows y Mac OS X). La aplicación permite realizar el almacenamiento de identidades en ficheros PKCS #12, PKCS #15 y dispositivos *hardware* criptográficos PKCS #11. Una vez generada una identidad digital y almacenada en uno de estos tipos de contenedores, el usuario podrá utilizar estos ficheros para importar sus claves en la aplicación en la que desee utilizarlas, por ejemplo, un cliente de correo o en su navegador web.

La aplicación de usuario permitirá a los usuarios realizar, como se puede ver en la figura 4.11 en la página siguiente, todas las operaciones definidas en el estándar CMP (solicitud inicial, revocación, renovación y actualización de la clave), así como comprobar el estado de un certificado utilizando en protocolo RTCS.

La figura 4.11 en la página siguiente muestra la ventana principal de la aplicación cliente. En esta ventana se puede ver que se debe indicar la ruta al fichero proporcionado por la agencia de registro, cuyo contenido se vio en el apartado 4.5.1.1, que contiene los datos necesarios para realizar las operaciones con la autoridad de certificación. Para las operaciones del protocolo CMP es necesario proporcionar siempre este fichero ya que

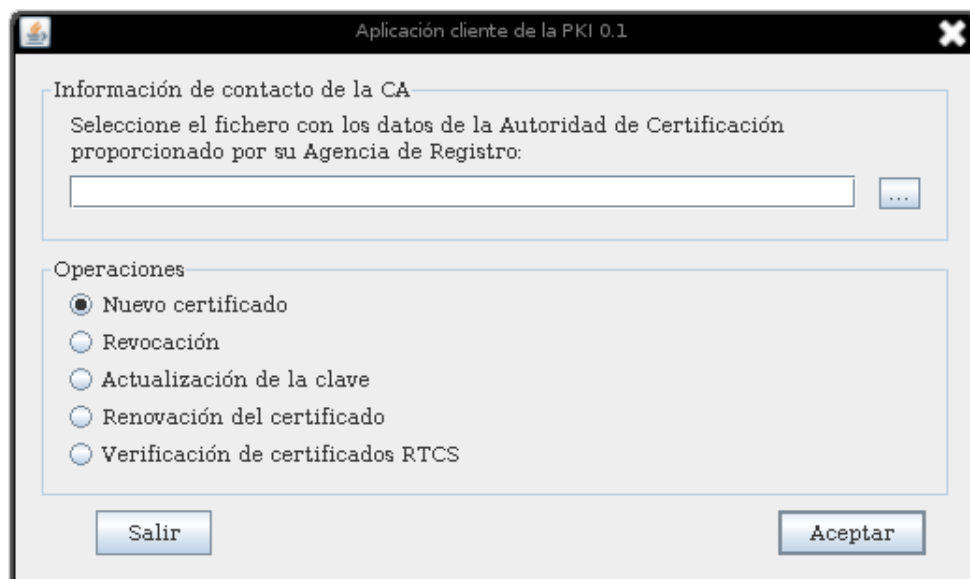


Figura 4.11: Ventana de operaciones de la aplicación de usuario

contiene, entre otras cosas, la ruta en la que se puede obtener el certificado de la autoridad de certificación.

En los siguientes apartados veremos como se realiza cada uno de los distintos tipos de operaciones.

4.6.1. Solicitud inicial

Una vez indicada a la aplicación la ruta del fichero con los datos proporcionados por la autoridad de certificación y seleccionada la opción de generar nuevo certificado se mostrará una ventana, como se puede ver en la figura 4.12 en la página siguiente, en la cual se deben seleccionar los parámetros deseados para la nueva identidad digital. Los parámetros que se pueden elegir son:

- **Tamaño de la clave:** Tamaño en bits deseado de la nueva clave.
- **Nombre de la clave:** En este campo se debe asignar un nombre identificativo para la nueva clave que servirá para identificarla dentro del dispositivo de almacenamiento de claves.
- **Usar dispositivo PKCS #11:** Si se activa esta opción se utilizará un dispositivo PKCS #11 para generar y almacenar la nueva identidad digital en lugar de hacerlo en un fichero. Cuando se marca esta opción es necesario introducir el PIN que permite abrir y utilizar el dispositivo hardware.

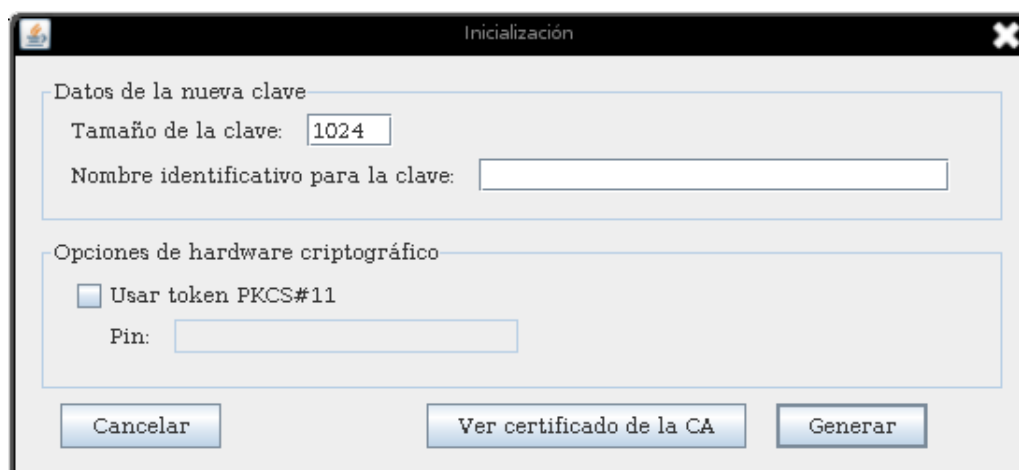


Figura 4.12: Ventana de generación de identidades

Cuando se han establecido los parámetros, la aplicación generará la nueva clave para el usuario, construirá la solicitud de certificación y la enviará al servidor CMP. Cuando el servidor reciba la solicitud verificará que sea correcta, como se vio en el apartado 4.4.2.1, y si lo es devolverá al usuario su nuevo certificado. El último paso que deberá realizar el usuario es seleccionar el fichero en el que desea almacenar la nueva identidad junto con el nuevo certificado generado.

4.6.2. Revocación

Para realizar una revocación mediante el protocolo CMP es necesario que la solicitud de revocación esté firmada por lo que el primer dato que solicita la aplicación es el fichero o el dispositivo PKCS #11 en el que se encuentra la identidad a revocar, así como la contraseña o el PIN respectivamente para poder acceder a la misma. Una vez selecciona la identidad que se desea revocar la aplicación generará la solicitud CMP de revocación y se la enviará al servidor. El servidor analizará la solicitud, como se vio en el apartado 4.4.2.2, y si la solicitud es correcta, realizará todas las tareas que implica realizar una revocación y enviará al usuario una respuesta indicando que la revocación se ha hecho efectiva.

4.6.3. Recertificación

Para realizar una recertificación se le indicará a la aplicación, de forma similar al caso de la revocación, el fichero o dispositivo que contiene la identidad a recertificar, así como la contraseña o el PIN de acceso. La aplicación generará la solicitud CMP de recertificación y la enviará al servidor. Éste verificará, como se vio en el apartado 4.4.2.3, que la solicitud sea correcta y si lo es enviará al usuario su nuevo certificado. A continuación la aplicación

cliente solicitará al usuario la ruta en la que desea almacenar el nuevo par de claves, que contendrá la clave privada antigua junto con el nuevo certificado.

4.6.4. Actualización de la clave

El proceso de actualización de clave es similar al de recertificación. Como ya hemos visto esta operación consiste en solicitar un certificado con los mismos campos que un certificado ya existente pero para un nuevo par de claves. La secuencia de pasos a seguir para realizar esta solicitud son los siguientes:

- Abrir el contenedor que contiene el certificado cuya clave se desea actualizar.
- Generar un nuevo par de claves para el que se desea solicitar el certificado.
- Enviar la solicitud de actualización de clave al servidor CMP de la infraestructura de clave pública.
- Recoger la respuesta del servidor con el nuevo certificado.
- Almacenar la nueva identidad junto con el nuevo certificado en un almacén de claves.

4.6.5. Verificación de certificados utilizando RTCS

La aplicación cliente dispone de una funcionalidad adicional aparte de poder realizar operaciones CMP de gestión de certificados. Esta aplicación dispone de una opción para verificar certificados utilizando el protocolo RTCS. Como se vio en el capítulo 3.4.3 el protocolo RTCS es una extensión del protocolo OCSP que permite comprobar en tiempo real el estado en el que se encuentra un certificado. Si la infraestructura de clave pública dispone de un servidor RTCS cualquier cliente de la PKI podrá realizar consultas al mismo para comprobar el estado de los certificados de dicha PKI antes de utilizarlos.

Para realizar una consulta RTCS el usuario deberá indicar la dirección y el puerto en el que atiende las peticiones el servidor RTCS y la ruta (dentro del sistema de ficheros o una URL) desde donde se pueda descargar el certificado que se desea verificar. Con estos datos la aplicación construirá la solicitud RTCS que enviará al servidor y recibirá la respuesta mostrando al usuario el resultado de la consulta con el estado del certificado.

4.7. Servicio de publicación LDAP

Un servicio bastante importante dentro de las aplicaciones de construcción de PKI es la publicación de certificados. Una PKI puede decidir publicar los certificados en algún

```
# LDAP server options
# Format "ldap://<url>/<credentials>?<password>"
ldap-repository = "ldap://fafnir/cn=Manager,dc=fafnir?pass"

# Database options
# Format: "DNS=<odbc-source>;UID=<user>;PWD=<pass>"
keystore = "DSN=myodbc-test;UID=dani;PWD=danipw"

# Intervalo en segundos entre publicaciones
publication-interval = 300
```

Figura 4.13: Fichero de configuración del publicador

tipo de repositorio al que puedan acceder los usuarios de dicha PKI, o incluso cualquier persona, para facilitar la obtención de certificados digitales por parte de los usuarios.

Los repositorios de publicación de certificados no son un elemento obligatorio que deba poseer cualquier PKI ya que en algunos casos no es necesario o incluso su uso puede ser desaconsejable. En entornos en los que los certificados digitales se utilizan, por ejemplo, para cifrar o firmar correos es interesante disponer de este tipo de servicios para permitir que unos usuarios puedan obtener de manera sencilla los certificados de otra persona a la que desean enviar un correo electrónico de manera confidencial. Otro ejemplo donde estos servicios son importantes es cuando los certificados se utilizan para autenticar usuarios frente a un sistema; en este caso el servidor de autenticación debe disponer de acceso a un repositorio de certificados para comprobar su validez o pertenencia a un grupo. Sin embargo, cuando la PKI pertenece a una organización privada puede ser necesario que los certificados no estén disponibles para personas ajenas a la PKI, o incluso para usuarios legítimos de la PKI si no se desea que unos usuarios conozcan qué otros usuarios pertenecen a la organización.

En el capítulo 3.5 estudiamos los diferentes protocolos que se pueden utilizar para realizar la publicación de certificados. Para la implementación llevada a cabo en este proyecto se ha decidido utilizar el protocolo LDAP que, aunque vimos que presenta algunos problemas, es el más utilizado en la actualidad para llevar a cabo este tipo de servicios.

Para implementar este servicio se ha diseñado una aplicación denominada *ldap_publisher* que se encargará de realizar periódicamente la tarea de publicar los certificados emitidos por nuestra autoridad de certificación en un directorio LDAP. Esta aplicación recibe como parámetro un fichero con su configuración como se puede ver en la figura 4.13 y, opcionalmente, un parámetro (*-republish*) que sirve para indicar al publicador que debe publicar todos los certificados de nuevo.

El servidor de publicación necesita tener acceso de lectura a la tabla *certificates* de la base de datos de la autoridad de certificación y permisos de escritura en el directorio

LDAP en el que se desea realizar la publicación de los certificados.

El funcionamiento de la aplicación es el siguiente:

1. Obtener todos los certificados de la base de datos de la autoridad de certificación que tengan el valor *false* en el campo *published*.
2. Conectar con el servidor LDAP.
3. Para cada certificado obtenido en el punto 1 crear el objeto a insertar en el LDAP tomando la información de los campos del certificado.
4. Enviar cada entrada al servidor LDAP para que sea almacenada.
5. Actualizar el campo *published* de la base de datos para indicar que el certificado ha sido publicado.
6. Esperar el tiempo especificado en el fichero de configuración hasta el siguiente plazo de publicación.
7. Volver al paso 1.

Para realizar el desarrollo y las pruebas de la aplicación se ha utilizado el servidor *OpenLDAP* aunque la aplicación puede funcionar con cualquier otro servidor que implemente el estándar de directorio LDAP. Además el servicio puede funcionar tanto con directorios locales dentro de la misma máquina como con servidores disponibles en otra máquina accesibles por red.

4.7.1. Modificaciones sobre la biblioteca *Cryptlib*

La biblioteca criptográfica *Cryptlib* tiene implementado parcialmente el soporte para realizar lecturas y escrituras de certificados digitales sobre un directorio LDAP. Para el desarrollo de este proyecto ha sido necesario modificar el código de la biblioteca para completar y adaptar el comportamiento a nuestras necesidades. Para realizar estas tareas se ha escrito un conjunto de parches, incluidos en la distribución de código fuente de la aplicación, que realizan las siguientes tareas:

- Añadir soporte para autenticarse mediante usuario y contraseña frente al directorio. El código fuente original de la biblioteca no tiene soporte para autenticación mediante usuario y contraseña sobre el directorio LDAP.
- Modificar la estructura de la tabla *certificates* que crea la biblioteca para añadir un nuevo campo que indique si un certificado ha sido publicado en el servidor LDAP o no. Así mismo se han modificado las funciones de acceso para lectura y escritura sobre esta tabla para que tengan en cuenta este nuevo campo.

- El código fuente de la biblioteca *Cryptlib* publica los certificados en el nodo raíz del directorio como un objeto no estándar denominado *certPerson*. Para que nuestra implementación se pueda utilizar de manera más general se ha modificado el código para que los certificados se publiquen en el atributo *userCertificate;binary* de un objeto del tipo *inetOrgPerson*. Esta decisión se ha tomado teniendo en cuenta que es en este lugar donde la mayoría de clientes buscarán los certificados de un usuario. Además, en lugar de publicar los certificados en el nodo raíz del directorio, se ha modificado el código para que la publicación se realice en la estructura arborescente que se forma con los atributos O y OU de los certificados.

4.8. Servicios de validación

El último elemento que puede formar parte de una infraestructura de clave pública son los servicios de validación de certificados que permiten comprobar el estado en el que se encuentra un certificado digital antes de utilizarlo para llevar a cabo operaciones criptográficas con él. Como se estudió en el apartado 3.4 existen diferentes protocolos que permiten realizar la verificación de certificados digitales. En este proyecto se han decidido implementar dos métodos diferentes, uno *offline* y otro *online*, para realizar la verificación de certificados. El primer método es *offline* y consiste en utilizar listas de revocación de certificados y el segundo, *online*, utiliza el protocolo RTCS para consultar el estado de un certificado en tiempo real. A continuación se detallará el desarrollo de cada una de las aplicaciones que implementan estos mecanismos.

4.8.1. Generación de CRLs

Para generar las listas de revocación de certificados periódicamente se ha creado la aplicación *crl_server* que actúa como un demonio que generará cada cierto tiempo (el especificado en la CRL en el campo *nextUpdate*) una nueva CRL con los certificados revocados en ese instante. La aplicación recibe como parámetro la ruta del fichero de configuración donde se almacenan todos los parámetros necesarios para su funcionamiento.

Como se puede ver en la figura 4.14 en la página siguiente la aplicación tiene los siguientes parámetros:

- Ruta del fichero con la identidad de la autoridad de certificación, así como la contraseña y la etiqueta para acceder a la misma, para firmar las CRLs. Las listas de revocación de certificados siempre tienen que estar firmadas con la misma identidad que la utilizada para firmar los certificados que se revocan. Si en lugar de almacenar la identidad de la autoridad de certificación en un almacén PKCS #15 se utiliza un dispositivo hardware PKCS#11 se debe asignar el valor *true* a la variable


```
# Private keyset options
keyset_file = root-ca.p15
keyset_pass = root
keyset_label = root

use-pkcs11 = false

# Database options
# Format: [ user:pass@ ]odbc-source
keystore = user:password@ca-database

crl-file = crl.der
crl-update-days = 1
```

Figura 4.14: Fichero de configuración del generador de CRLs

use-pkcs11. Si se indica este valor, el programa utilizará la identidad almacenada en el token criptográfico para realizar la firma de las CRLs.

- Parámetros de acceso al almacén de certificados de la autoridad de certificación. Como ya vimos en el apartado 4.4.1.3 el servidor CMP generará la estructura de una CRL cada vez que se revoca un nuevo certificado. La aplicación utilizará estos parámetros para acceder a la base de datos y extraer esta estructura y firmarla para generar la nueva CRL.
- Nombre del fichero para almacenar la CRL: El campo *crl-file* permite especificar la ruta del fichero en el que se desea guardar la nueva CRL emitida.
- Periodo de publicación: Este campo permite especificar la frecuencia con la que se deben publicar nuevas CRLs. Esta medida de tiempo se incluirá como valor del campo *nextUpdate* de la CRL que se está generando para informar a los usuarios de cuando se publicará la próxima CRL.

4.8.2. Servidor de validación RTCS

Para proporcionar un servicio de validación de certificados *online* se ha decidido utilizar el protocolo RTCS. Este protocolo utiliza las extensiones del protocolo OCSP para subsanar las carencias que presenta este último. Para implementar este servicio de ha desarrollado la aplicación *rtcs_server* que proporciona mecanismos verificación de certificados online utilizando el protocolo RTCS. Esta aplicación, que actúa como un demonio, recibe como parámetro un fichero de configuración como se puede ver en la figura 4.15 en la página siguiente.

```
# Port to listen
port = 5001

# Private keyset options
use-pkcs11 = false
keyset_file = ocsp.p15
keyset_pass = root
keyset_label = root

# Number of threads
max_threads = 10

# Database options
# Format: [ user:pass@ ]odbc-source
keystore = user:userpw@ca-database
```

Figura 4.15: Fichero de configuración del servidor RTCS

Los parámetros que necesita la aplicación para funcionar correctamente son:

- Puerto en el que el servidor atenderá las solicitudes de los clientes.
- Ruta del fichero con la identidad que se utilizará para firmar las respuestas, así como la contraseña y la etiqueta para acceder a la misma. La identidad con la que se van a firmar las respuestas del protocolo RTCS debe estar generada por la autoridad de certificación que emitió el certificado por el que se está consultando su estado y además, debe tener activo el bit que indica que la clave se puede utilizar para firmar respuestas a solicitudes sobre la consulta del estado de certificados. Si en lugar de almacenar la identidad de la autoridad de certificación en un almacén PKCS #15 se utiliza un dispositivo hardware PKCS#11 se debe asignar el valor *true* a la variable *use-pkcs11*. Si se indica este valor, el programa utilizará la identidad almacenada en el token criptográfico para realizar la firma de las CRLs.
- Puesto que la aplicación es un servidor multihilo, en el campo *max_threads* se debe especificar el número máximo de procesos ligeros que se deben crear para atender concurrentemente varias solicitudes por parte de uno o varios usuarios.
- Información para acceder a la base de datos de la autoridad de certificación. Puesto que esta aplicación proporciona el estado en tiempo de real de un certificado, la aplicación debe disponer de permisos para realizar lecturas en la tabla *certificates* de la base de datos de la autoridad y así conocer el estado en el que se encuentra cada certificado en un determinado instante.

El funcionamiento de esta aplicación es el siguiente:

1. Acceder al fichero o dispositivo que almacena la identidad para firmar las solicitudes.
2. Abrir la conexión con la base de datos de la autoridad de certificación para poder consultar el estado de los certificados.
3. Abrir el puerto definido en el fichero de configuración y esperar la llegada de solicitudes.
4. Cuando se reciba una solicitud se consultará en la base de datos el estado del mismo, que puede ser, por ejemplo, valido, revocado, inexistente... A partir de esta información se construirá la respuesta a la solicitud y se firmará con la clave privada del servicio.
5. Esperar la llegada de nuevas solicitudes y volver al paso 4 cuando se reciba una nueva solicitud.

4.9. Esquema general de la solución

En los apartados anteriores hemos visto uno por uno todos los elementos que componen la solución software construida en este proyecto. En la figura 4.16 en la página siguiente se puede ver un esquema con todos los elementos así como las interacciones que se producen entre los mismos. Las conexiones continuas indican lectura y escritura de datos y las representadas con una línea punteada indican comunicaciones entre los distintos componentes.

Para establecer una infraestructura de clave pública con esta solución solo es necesario disponer del servidor CMP, la agencia de registro y la aplicación de usuario. El resto de elementos que forman parte de la autoridad de certificación son opcionales y pueden ejecutarse tanto en la misma máquina como en otra totalmente independiente de la que ejecuta el servidor CMP. Con este diseño, basado en diferentes módulos independientes, se ha conseguido construir una solución totalmente modular que era uno de los principales objetivos que se perseguía con el desarrollo de la aplicación.

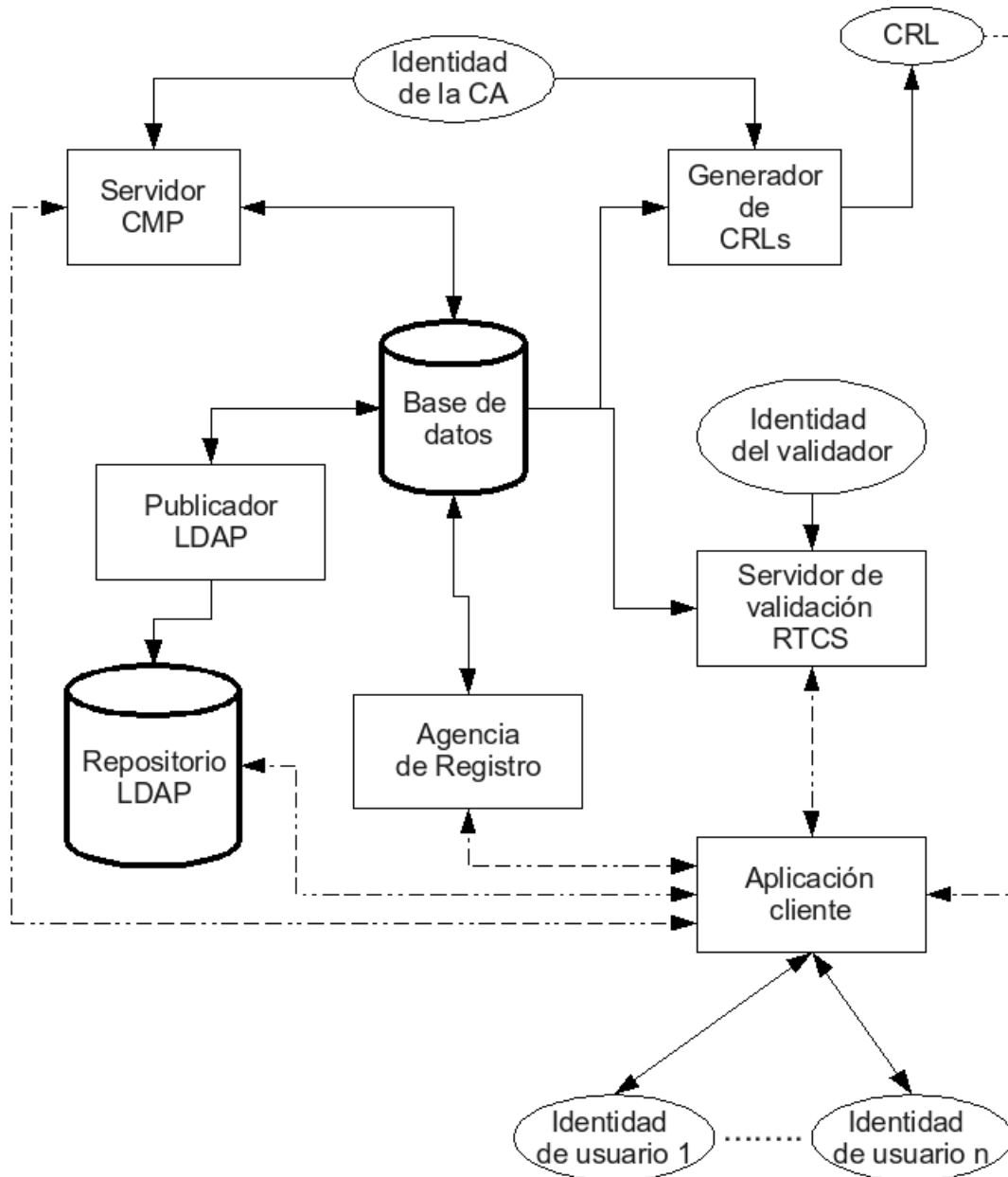


Figura 4.16: Esquema general de la solución software

Capítulo 5

CONCLUSIONES

En la actualidad, las infraestructuras de clave pública están teniendo un importante auge debido en parte al importante crecimiento de la informática de consumo en los hogares y al acceso a la misma por una gran parte de la sociedad. Actualmente se pueden realizar un gran número de tramites con las administraciones públicas a través de la red, como por ejemplo, realizar gestiones con la agencia tributaria o con la seguridad social.

La implantación de este tipo de servicios hace necesario disponer de una infraestructura de clave pública que permita proveer a los ciudadanos de identidades digitales que les permitan realizar con facilidad y seguridad este tipo de gestiones. Las infraestructuras de clave pública son la solución más extendida en la actualidad para gestionar el acceso a este tipo de servicios; claros ejemplos son la PKI del DNI electrónico [8] o el proyecto Ceres [24].

El desarrollo de los estándares relacionados con las PKI sigue creciendo actualmente gracias al *PKIX Working Group* [22] que continúa creando, actualizando y corrigiendo los estándares relacionados con las identidades digitales X.509. Gracias a este grupo de expertos, la tecnología X.509 continua avanzando y corrigiendo algunos de los defectos o carencias que presentan los estándares, como hemos visto en el estudio realizado en este proyecto.

Algunos expertos dudan del futuro de estas soluciones, argumentando que están basados en estándares obsoletos de hace décadas y que deberían adaptarse a las tecnologías actuales. Algunas propuestas son, por ejemplo, usar XML como lenguaje de representación para los certificados X.509 ya que los estándares de firma digital y cifrado en XML se están desarrollando a una gran velocidad y no parecen encajar muy bien con las representaciones, para muchos arcaicas, en ASN.1.

El futuro de las infraestructuras de clave pública deberá adaptarse a las necesidades actuales y mejorar en dos aspectos fundamentales:

Usabilidad: Como hemos visto, las PKIs están formadas por un gran número de elementos, en algunos casos, bastante complejos. Muchas de las soluciones PKI existentes presentan un alto grado de complejidad tanto para su gestión por los

administradores de la misma, como para su uso por parte de los usuarios. Puesto que cada vez más servicios se ofrecen a personas con pocos conocimientos en informática, y menos aún en seguridad, es necesario construir soluciones simples y fáciles de utilizar para evitar que se produzca una relación entre alta seguridad y alta complejidad de uso.

Interoperabilidad: Las PKIs se apoyan en un gran número de estándares que, en ocasiones, presentan pequeñas deficiencias que permiten que puedan ser interpretadas de diferente forma por diferentes personas. Esto provoca que muchos elementos de soluciones de PKI no sean interoperables con otros elementos de diferentes soluciones. Es necesario que estos estándares se revisen y se llegue a consensos acerca de estos pequeños aspectos y se consiga realizar implementaciones modulares e interoperables que permitan una mayor facilidad de mantenimiento y desarrollo.

Bibliografía

- [AFK05] C. Adams, S. Farrell, and T. Kause. Internet X.509 Public Key Infrastructure Certificate Management Protocols. *RFC 4210*, September 2005.
- [AL02] C. Adams and S. Lloyd. *Understanding PKI: Concepts, Standards and Deployment Considerations*. Addison Wesley, second edition, 6 November 2002.
- [Cho02] Suranjan Choudhury. *Public Key Infrastructure: Implementation and Design*. March 2002.
- [DH76] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, vol. IT-22:644–654, November 1976.
- [ElG85] Taher ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.
- [Ell00] B. Ellison, C.; Schneier. Ten Risks of PKI: What You’re not Being Told about Public Key Infrastructure. *Computer Security Journal*, XVI(1), 2000.
- [Gut02a] P. Gutmann. PKI: It’s Not Dead, Just Resting. *IEEE Computer magazine*, 35(8):41–49, August 2002.
- [Gut02b] P. Gutmann. Real-Time Certificate Status Facility for OCSP - RTCS. December 2002.
- [Gut03] P. Gutmann. Plug-and-Play PKI: A PKI Your Mother Can Use. *12th USENIX Security Symposium*, pages 45–58 of the Proceedings, 2003.
- [Hou] R. Housley. Cryptographic Message Syntax. *RFC 2630*.
- [HPF02] R. Housley, W. Polk, and W. Ford. Internet X.509 Public Key Infrastructure: Certificate and CRL Profile. *RFC 3280*, April 2002.
- [K. 06] K. Zeilenga. Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates. *RFC 4523*, June 2006.

- [Kob87] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, (48):203–209, 1987.
- [MAM99] M. Myers, R. Ankney, and A. Malpani. Online Certificate Status Protocol - OCSP. *RFC 2560*, June 1999.
- [Mil85] V. Miller. Use of elliptic curves in cryptography. *CRYPTO*, (85), 1985.
- [MLS00] M. Myers, X. Liu, and J. Schaad. Certificate Management Messages over CMS. *RFC 2797*, April 2000.
- [NDJ01] A. Nash, W. Duane, and C. Joseph. *PKI: Implementing and Managing E-Security*. McGraw-Hill, Inc, 2001.
- [Ope06] OpenLDAP Foundation. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. *RFC 4510*, June 2006.
- [RSA78] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21 (2):120–126, 1978.
- [RSA99] RSA Laboratories. PKCS #12: Personal Information Exchange Syntax Standard. 22 June 1999.
- [RSA00a] RSA Laboratories. PKCS #10: Certification Request Syntax Standard. 26 May 2000.
- [RSA00b] RSA Laboratories. PKCS #15: Cryptographic Token Information Format Standard. 6 June 2000.
- [RSA00c] RSA Laboratories. PKCS #9: Selected Attribute Types. 25 February 2000.
- [RSA04] RSA Laboratories. PKCS #11: Cryptographic Token Interface Standard v2.20. 28 June 2004.
- [Smi00] M. Smith. Definition of the inetOrgPerson LDAP Object Class. *RFC 2798*, April 2000.
- [U.S94] U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology. DIGITAL SIGNATURE STANDARD (DSS). *Federal Information Processing Standards Publication 186*, 19 May 1994.

Referencias

- [1] Aladdin Etoken RTE, <http://www.aladdin.com/etoken/>.
- [2] Ant, <http://ant.apache.org/>.
- [3] Autoconf, <http://www.gnu.org/software/autoconf/>.
- [4] Automake, <http://www.gnu.org/software/automake/>.
- [5] Ceres, <http://www.cert.fnmt.es/>.
- [6] Confuse, <http://www.nongnu.org/confuse/>.
- [7] Criplib, <http://www.cs.auckland.ac.nz/~pgut001/cryptlib/>.
- [8] DNI electronico, <http://www.dnielectronico.es/>.
- [9] EJBCA, <http://ejbca.sourceforge.net/index.html>.
- [10] Entrust Authority, <http://www.entrust.com/>.
- [11] GCC, <http://gcc.gnu.org/>.
- [12] iodbc, <http://www.iodbc.org/>.
- [13] JBoss, <http://labs.jboss.com/>.
- [14] LyX, <http://www.lyx.org/>.
- [15] MySQL, <http://mysql.com/>.
- [16] NetBeans, <http://www.netbeans.org>.
- [17] OpenCA, <https://www.openca.org/projects/openca/>.
- [18] OpenLDAP, <http://www.openldap.org/>.
- [19] OpenPGP, <http://www.openpgp.org/>.

- [20] OpenSSL, <http://www.openssl.org/>.
- [21] PC/SC lite, <http://www.linuxnet.com/middle.html>.
- [22] PKIX Working Group, <http://www.ietf.org/html.charters/pkix-charter.html>.
- [23] PostgreSQL, <http://www.postgresql.org/>.
- [24] Proyecto CERES, <http://www.cert.fnmt.es/>.
- [25] Swing, <http://java.sun.com/products/jfc/tsc/articles/architecture/>.
- [26] UnixODBC, <http://www.unixodbc.org/>.
- [27] VeriSign Public Key Infrastructure, <http://www.verisign.com/products-services/security-services/pki/pki-application/index.html>.

Apéndice A

MANUALES DE INSTALACIÓN

A.1. Manual de instalación de la autoridad de certificación

A.1.1. Requisitos de la aplicación

Para construir el software que compone la autoridad de certificación es necesario disponer de las siguientes bibliotecas: confuse, unixODBC, openLDAP y Cryptlib. Las dos primeras están disponibles en los repositorios de distribución de software de la gran mayoría de distribuciones de GNU/Linux por lo que no se detallará aquí el proceso manual de compilación e instalación. Sin embargo la biblioteca Cryptlib no está disponible en muchas distribuciones de GNU/Linux por lo que se describirá el proceso de compilación e instalación.

Para compilar la biblioteca Cryptlib se deben seguir los pasos que se pueden ver en la figura A.1 en la página siguiente. Es necesario tener instalado previamente las bibliotecas unixODBC y openLDAP para que durante la compilación de Cryptlib se detecte que están instaladas y se añada el soporte correspondiente. Además, como vimos en el capítulo de desarrollo es necesario aplicar varios parches a Cryptlib para añadir, entre otras cosas, soporte para la escritura en directorios LDAP. Los parches, que forman parte de la distribución de código fuente de la autoridad de certificación son los ficheros *cryptlib-3.3.2_rc2-ldap.patch* y *cryptlib-3.3.2_rc2-database-published-field.patch* y deben aplicarse antes de compilar la biblioteca.

A.1.2. Instalación de la aplicación

Una vez instaladas las bibliotecas correspondientes se puede proceder a compilar el código fuente de las aplicaciones que componen la autoridad de certificación. Para ello se deben seguir los pasos que aparecen en la figura A.2 en la página siguiente. Una vez

```
$ wget http://www.cypherpunks.to/~peter/snapshot.zip
$ unzip -qoa snapshot.zip -d cryptlib
$ cd cryptlib
Copiar los 2 ficheros con los parches aqui
$ patch -p0 < cryptlib-3.3.2_rc2-ldap.patch
$ patch -p0 < cryptlib-3.3.2_rc2-database-published-field.patch
$ make shared
$ su
Contraseña :
# cp libcl.so.3.3.0 /usr/local/lib
# cp cryptlib.h /usr/local/include
# ln -s /usr/local/lib/libcl.so.3.3.0 /usr/local/lib/libcl.so
```

Figura A.1: Compilación e instalación de Cryptlib

```
$ tar zxvf ca-0.1.tar.gz
$ cd ca-0.1
$ ./configure
$ make
$ su -
Contraseña :
# make install
```

Figura A.2: Compilación de las aplicaciones de la autoridad de certificación

realizado el proceso todas las aplicaciones se habrán instalado en el directorio */usr/local/bin* del sistema y estarán accesibles para poder ser ejecutadas por todos los usuarios de la maquina; además en el directorio */usr/local/share/ca* estarán disponibles las plantillas para los ficheros de configuración de cada una de las aplicaciones, cuyo formato se especificó en el capítulo de desarrollo de la aplicación.

A.1.3. Configuración previa

En este apartado veremos que pasos se deben seguir para configurar los elementos necesarios para que las aplicaciones que forman la agencia de registro puedan funcionar correctamente.

A.1.3.1. Acceso a dispositivos PKCS #11

Todas las aplicaciones que componen la autoridad de certificación que utilizan identidades digitales pueden almacenar las mismas en dispositivos PKCS #11 hardware o

```
[ODBC Drivers]  
myodbc = Installed  
  
[myodbc]  
Driver = /usr/lib64/libmyodbc3 - 3.51.12. so
```

Figura A.3: Ejemplo de fichero de configuración .odbcinst.ini

en ficheros PKCS #15. Si se desea utilizar un dispositivo hardware para guardar y utilizar la identidad de la autoridad de certificación es necesario indicar a las aplicaciones donde se encuentra la biblioteca, proporcionada por el fabricante, que permite el acceso a dicho dispositivo. Para realizar esta tarea se debe utilizar la aplicación *register_pkcs11*. Esta aplicación recibe como parámetro el nombre de la biblioteca dinámica necesaria para acceder al dispositivo y configurará Cryptlib correctamente para que pueda tener acceso al mismo.

A.1.3.2. Acceso a la base de datos de la autoridad de certificación

Las aplicaciones necesitan disponer de acceso a una base de datos en la que la autoridad de certificación pueda almacenar los certificados generados, las CRLs, etc. Para acceder a esta base de datos se ha decidido utilizar ODBC ya que permite el acceso a cualquier base de datos que cumpla con el estándar ODBC (que son la gran mayoría). Puesto que las aplicaciones desarrolladas en este proyecto son independientes del gestor de base de datos utilizado no veremos aquí como se configura ningún gestor en concreto sino que supondremos que el usuario tiene acceso a una base de datos llamada, por ejemplo, *ca-database* en un gestor de base de datos con interfaz ODBC (MySQL o PostgreSQL por ejemplo). Para configurar el acceso a la base de datos utilizando unixODBC se deben crear dos ficheros:

- *.odbcinst.ini*: Este fichero contiene la configuración genérica del driver para acceder a un tipo concreto de base de datos. En la figura A.3 se puede ver un ejemplo de configuración para acceder a la una base de datos MySQL.
- *.odbc.ini*: Este fichero contiene la configuración concreta para acceder a una base de datos utilizando el driver definido en el fichero anterior. En la figura A.4 en la página siguiente se puede ver un ejemplo de como se debería definir el fichero para acceder a la base de datos *ca-database* en un servidor MySQL situado en una máquina remota.

Como se puede ver en ambas figuras, el recurso ODBC tiene por nombre *myodbc-test*; el nombre elegido aquí será el que se utilice en todas las aplicaciones de la PKI para acceder a la base de datos.

```
[ODBC Data Sources]
myodbc-test          = MySQL ODBC myodbc-3.51.11 Driver

[myodbc-test]
Description          = MySQL ODBC myodbc-3.51.11 Driver
Driver               = myodbc
Socket               = /var/run/mysqld/mysqld.sock
Server               = database-server.ls.fi.upm.es
Database             = ca-database
Option               = 3
```

Figura A.4: Ejemplo de fichero de configuración .odbc.ini

Una vez configurado el acceso a la base de datos es necesario crear la estructura inicial de la misma, es decir, es necesario crear la estructura de tablas para el correcto funcionamiento de todas las aplicaciones. Para construir esta estructura inicial se debe lanzar el ejecutable *initialize_ca_database*. Cuando se llama a este ejecutable nos pedirá el nombre del recurso ODBC (*myodbc-test* en nuestros ejemplos) y el usuario y la contraseña para acceder a la base de datos. Si no hay ningún error con los parámetros, el programa creará las tablas correspondientes dentro de la base de datos que ya estará lista para ser utilizada por el resto de aplicaciones.

A.1.3.3. Generación de la identidad de la autoridad de certificación

Antes de utilizar las aplicaciones de la autoridad de certificación es necesario crear la identidad digital autofirmada de la autoridad. Para ello se utilizará la aplicación *generate_root_ca*. Esta aplicación preguntará interactivamente al usuario algunos datos acerca de la clave privada y el certificado que se va a generar. Una vez generado el par de claves, éste y el certificado autofirmado serán almacenados en un fichero, cuyo nombre también se le solicitará al usuario. Este fichero lo utilizarán el resto de aplicaciones para firmar las solicitudes y los certificados necesarios. Si se desea generar la identidad de la autoridad dentro de un dispositivo PCKS #11, se deberá utilizar el parámetro *-pkcs11* que provoca que todas las operaciones se realicen sobre el dispositivo hardware en lugar de sobre un contenedor PKCS #15.

A.1.4. Ejecución de los servicios

En este apartado definiremos como se ejecutan y que parámetros son necesarios para arrancar cada una de las aplicaciones que pueden formar parte de nuestra infraestructura de clave pública. Como ya hemos visto, el único que debe arrancarse obligatoriamente es el servidor CMP que es el que permite tramitar las solicitudes de los usuarios. El resto de

```
# Port to listen
port = 5000
# Private keyset options
#use-pkcs11 = true
keyset_file = root-ca.p15
keyset_pass = root
keyset_label = root
# Number of threads
max_threads = 10
# Time to expire certs
expire_timeout = 600
# Database options
# Format: [user:pass@]odbc-source
keystore = dani:password@myodbc-test
```

Figura A.5: Ejemplo de fichero de configuración del servidor CMP

los servicios son opcionales y pueden utilizarse o no en función de la funcionalidad que se desee proporcionar a la PKI. Todos los elementos especificados en este apartado son independientes por lo que pueden ejecutarse en la misma máquina o en otra independiente. El único requisito que deben tener en común es tener acceso al gestor de base de datos en el que se almacena la base de datos de la autoridad de certificación.

A.1.4.1. Servidor CMP

La aplicación que implementa el servidor CMP se llama *ca_server* y para ejecutarla simplemente es necesario especificar el fichero de configuración de la misma. En la figura A.5 se puede ver el ejemplo de fichero de configuración que se instala junto con la distribución en el fichero */usr/local/share/ca/server.conf*.

El significado de cada una de las variables del fichero es el siguiente:

port: Puerto en el que el servidor atenderá las solicitudes de los clientes.

use-pkcs11: Variable *booleana* que indica si se debe utilizar un dispositivo hardware PKCS #11 para leer la identidad de la autoridad de certificación. Si el valor de esta variable es *false* o la variable no aparece (como en el ejemplo, que aparece comentada), entonces se utilizará un fichero PCKS #15 que se habrá generado previamente como vimos en el apartado A.1.3.3.

keyset_file: Ruta al fichero que contiene la identidad de la autoridad de certificación.

keyset_pass: Contraseña para acceder al fichero de la identidad.

keyset_label: Nombre identificativo de la identidad dentro del fichero anterior.

max_threads: Número de threads que se deben utilizar. Esta variable indica el número máximo de hilos que creará la aplicación y por tanto el número máximo de peticiones simultaneas que puede atender el servidor.

expire_timeout: Intervalo de tiempo que indica cada cuanto tiempo se revisará la base de datos para eliminar certificados que hayan caducado.

keystore: Información de acceso al recurso ODBC que permite el acceso a la base de datos de la autoridad de certificación. En la figura A.5 en la página anterior se puede ver que se debe indicar el nombre de usuario, la contraseña y el nombre del recurso. El nombre del recurso será el que se haya definido al configurar unixODBC como se vio en el capítulo A.1.3.2.

Una vez arrancado el servidor, pasándole como parámetro el fichero de configuración, éste permanecerá a la espera de solicitudes CMP indefinidamente. Para parar la aplicación bastará con enviarle la señal de terminación mediante la combinación Ctrl+C y el servidor terminará su ejecución de manera controlada.

A.1.4.2. Generación de CRLs

Si se desea que nuestra PKI genere listas de revocación de certificados periódicamente se debe utilizar la aplicación *crl_server*. Esta aplicación recibe como parámetro un fichero de configuración con los parámetros necesarios para su funcionamiento. Un ejemplo de este fichero de configuración, como se puede ver en la figura A.6 en la página siguiente, se instalará junto con la aplicación en la ruta */usr/local/share/ca/crl-server.conf*.

El significado de cada uno de los parámetros de este fichero es el siguiente:

keyset_file: Ruta del fichero con la identidad de la autoridad de certificación para firmar las CRLs.

keyset_pass: Contraseña para acceder al fichero del campo anterior.

keyset_label: Etiqueta de la identidad que se debe utilizar para firmar las CRLs.

use-pkcs11: Si en lugar de almacenar la identidad de la autoridad de certificación en un almacén PKCS #15 se utiliza un dispositivo hardware PKCS#11 se debe asignar el valor *true* a la variable *use-pkcs11*. Si se indica este valor, el programa utilizará la identidad almacenada en el token criptográfico para realizar la firma de las CRLs.

keystore: Parámetros de acceso al almacén de certificados de la autoridad de certificación.

crl-file: Nombre del fichero para almacenar la nueva CRL emitida.


```
# Private keyset options
keyset_file = root-ca.p15
keyset_pass = root
keyset_label = root

use-pkcs11 = false

# Database options
# Format: [ user:pass@ ]odbc-source
keystore = user:password@ca-database

crl-file = crl.der
crl-update-days = 1
```

Figura A.6: Ejemplo de fichero de configuración del generador de CRLs

crl_update-days: Este campo permite especificar la frecuencia con la que se deben publicar nuevas CRLs. Esta medida de tiempo se incluirá como valor del campo *nextUpdate* de la CRL que se está generando para informar a los usuarios de cuando se publicará la próxima CRL.

Una vez arrancada la aplicación, pasándole como parámetro el fichero de configuración, ésta generará CRLs cada vez que se cumpla el periodo de publicación. Para parar la aplicación bastará con enviarle la señal de terminación mediante la combinación Ctrl+C y el servidor terminará su ejecución de manera controlada.

A.1.4.3. Servicio de validación RTCS

De igual manera que las aplicaciones vistas hasta ahora, el servidor de validación RTCS actúa como un demonio que atenderá de manera indefinida las solicitudes de verificación de estado de certificados que reciba de los clientes y proporcionará el resultado a dichas consultas. La aplicación que implementa este servicio se denomina *rtcs_server* y recibe como parámetro un fichero con los parámetros de configuración necesarios para su correcto funcionamiento. Junto con la aplicación se instala un fichero de configuración de ejemplo llamado */usr/local/share/ca/rtcs.conf* cuyo contenido se puede ver en la figura A.7 en la página siguiente.

El significado de cada uno de los elementos de este fichero es el mismo que el visto en los apartados anteriores.

```
# Port to listen
port = 5001

# Private keyset options
#use-pkcs11 = true
keyset_file = ocsp.p15
keyset_pass = root
keyset_label = root

# Number of threads
max_threads = 10

# Database options
# Format: [ user:pass@ ]odbc-source
keystore = dani:danipw@myodbc-test
```

Figura A.7: Ejemplo de fichero de configuración del validador RTCS

A.1.4.4. Servicio de publicación LDAP

La última aplicación, llamada *ldap-publisher*, que forma parte de la infraestructura de clave pública es el servicio de publicación LDAP. Esta aplicación, al igual que las anteriores, recibe como parámetro un fichero con la configuración de los elementos necesarios para su correcto funcionamiento. Opcionalmente puede recibir otro parámetro, llamado *-republish*, que provoca que todos los certificados vuelvan a ser publicados en el directorio LDAP. Esta opción es útil cuando se cambia el servidor LDAP en el que se realizan las publicaciones para forzar que el directorio se rellene con todos los certificados aunque en su estado en la base de datos aparezcan como publicados. En la figura A.8 en la página siguiente se puede ver el contenido del fichero de configuración de ejemplo */usr/local/share/ca/ldap-publisher.conf*.

A.2. Manual de instalación de la agencia de registro

A.2.1. Configuración previa

Para utilizar la agencia es necesario tener configurado un recurso ODBC que permita el acceso a la base de datos de la autoridad de certificación para poder añadir y borrar usuarios de la PKI. El procedimiento para realizar esta configuración es el mismo que el descrito en el capítulo A.1.3.2 en la página 83.

```
# LDAP server options
# Format "ldap://<url>/<credentials>?<password>"
ldap-repository = "ldap://fafnir/cn=Manager,dc=fafnir?pass"

# Database options
# Format: "DNS=<odbc-source>;UID=<user>;PWD=<pass>"
keystore = "DSN=myodbc-test;UID=dani;PWD=danipw"

# Intervalo en segundos entre publicaciones
publication-interval = 300
```

Figura A.8: Ejemplo de fichero de configuración del publicador LDAP

```
direccion=cmp-server.es
puerto=5000
url-certificado=http://ca-server.es/root-ca.der
```

Figura A.9: Ejemplo de fichero de configuración de la agencia de registro

A.2.2. Instalación de la aplicación

La aplicación de la agencia de registro se distribuye mediante un fichero comprimido en zip. Existe un fichero para cada uno de los tres sistemas operativos soportados por la aplicación: GNU/Linux, Windows y Mac OS X. Aunque la aplicación está realizada en Java y es portable, se han creado tres distribuciones diferentes ya que junto con la aplicación se distribuyen las bibliotecas apropiadas para ejecutar la aplicación en la plataforma correspondiente.

La instalación de la aplicación en cada una de las plataformas es bastante similar. Simplemente se debe descomprimir el fichero zip que creará un directorio denominado *registry-agency* que contendrá la estructura de directorios necesaria para ejecutar la aplicación. Antes de ejecutar la aplicación se debe personalizar el fichero *config.conf* de acuerdo con la autoridad de certificación para la que se vaya a trabajar. La aplicación contiene un fichero de configuración de ejemplo cuyo contenido se puede ver en la figura A.9.

El significado de cada uno de los campos de este fichero es el siguiente:

dirección: Dirección IP o nombre de la maquina en la que el servidor CMP atiende las peticiones.

puerto: Número de puerto en el que el servidor CMP atiende las peticiones CMP.

url-certificado: Ruta en la que se puede encontrar el certificado de la autoridad de certificación.

Estos campos los utilizará la agencia de registro para construir el fichero de datos que proporcionará a los usuarios de la PKI y que les permitirá ponerse en contacto con la autoridad de certificación una vez dados de alta.

Para iniciar la aplicación una vez instalada y configurada solamente es necesario ejecutar el script *registry-agency.bat*, si se está ejecutando en un sistema Windows, o el script *registry-agency.sh* si se está utilizando el sistema operativo GNU/Linux o Mac OS X.

A.3. Manual de instalación de la aplicación de usuario

El procedimiento para instalar la aplicación de usuario es similar al descrito en el apartado anterior. Se han desarrollado tres distribuciones diferentes, una para cada sistema operativo en el que funciona la aplicación. Para instalar la aplicación solo es necesario descomprimir el fichero zip que se distribuye para cada plataforma, ya que la distribución está autocontenida, y solo es necesario disponer de una máquina virtual de Java instalada en el sistema. Para ejecutar la aplicación, una vez descomprimida, solo es necesario ejecutar el script correspondiente: *pki-client.sh* si se está ejecutando sobre los sistemas operativos GNU/Linux o Mac OS X, o *pki-client.bat* en el sistema operativo Windows.

Como se vio en el capítulo de desarrollo, esta aplicación tiene soporte para utilizar dispositivos hardware PKCS #11 para el almacenamiento y gestión de las identidades digitales que genera. Puesto que la aplicación solo se ha podido probar con un dispositivo hardware en concreto (Aladdin Etoken), el soporte para este hardware está activo por defecto siempre que el driver del mismo, proporcionado por el fabricante, se haya instalado previamente en el sistema.