PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DE FIRMA DIGITAL EN UNA PLATAFORMA DE COMERCIO ELECTRÓNICO

Tesis para optar el Título de Ingeniero Informático

Presentado por:

Walter Augusto García Rojas

Lima – Perú

2008

TABLA DE CONTENIDO

TABLA DE GRÁFICOS

CAPITULO 1: INTRODUCCIÓN Y DEFINICIÓN DEL MARCO CONCEPTUAL

1. INTRODUCCIÓN

El comercio electrónico, desde sus inicios, se ha ido desarrollando permanentemente, de maneras muy diversas, bajo distintas modalidades y haciendo uso de las diferentes tecnologías vigentes en cada momento. En épocas recientes, desde la aparición de las tecnologías de información, los medios electrónicos han sido utilizados para realizar operaciones comerciales de toda índole, en atención a las necesidades de los agentes involucrados. Es en este contexto que se logra identificar que el comercio electrónico involucra transacciones comerciales en la cuales se procesan y se transfieren datos digitalizados.

El comercio electrónico exige mucho más que el solo hecho de transferir información, requiere que se proporcione a las partes interesadas una seguridad sobre la información transferida y sus efectos, no solo tecnológica, sino también jurídica. Para lograr atender esta necesidad nace la firma digital, que en términos

legales es el equivalente a una firma manuscrita y debe cumplir las mismas funciones principales, como son: la autenticación de la identidad del firmante, la integridad de la información, la confidencialidad de los datos y el no repudio de la información.

En el Perú y el mundo se han adoptado legislaciones orientadas a permitir, contribuir y fomentar el uso de la firma digital con la finalidad de promover el desarrollo del comercio electrónico en el sector empresarial, el cual comúnmente es conocido como comercio Business to Business (B2B).

Dentro de esta línea de pensamiento, en el caso específico de Perú, se encuentran vigentes desde el año 2000 tres leyes que tienen por finalidad incentivar y promover el comercio electrónico dentro y fuera del Perú. Estas leyes permiten otorgar validez y eficacia jurídica a los documentos electrónicos. Además existen leyes sobre firmas y certificados digitales, así como también leyes relativas a los delitos informáticos.

2. DEFINICION DEL PROBLEMA

La firma manuscrita es todavía la forma más utilizada y “confiable” para relacionar un documento con una persona en particular, de manera legal. Sin embargo, este método ha adolecido y sigue adoleciendo de diversas imperfecciones, entre ellas la posibilidad de falsificación y las dificultades en el proceso de verificación de la firma.

La firma en si, involucra dos acciones: la acción de firmar y la acción de verificación de la firma. La acción de firmar, en el caso de la firma manuscrita, consiste en que una persona deje su rúbrica; mientras que la acción de verificación es más complicada ya que se requiere en algunos casos la utilización de tecnología altamente sofisticada y siempre con probabilidad de error.

Otra limitación que se presenta en las transacciones comerciales (como la firma de contratos) es la necesidad de contar con la presencia física y simultánea de las

personas involucradas y la presencia de un notario que garantice la validez de ésta, lo cual hace lenta y costosa una transacción entre organizaciones ubicadas en diferentes partes del mundo.

Precisamente como solución a estos problemas nace una nueva tecnología que puede reemplazar a la firma manuscrita, y que se ha denominado firma digital.

Esta tecnología va llegando poco a poco a diferentes lugares del mundo, y los gobiernos, concientes de las claras ventajas de ésta, hacen los esfuerzos necesarios para implantarla en sus naciones, promulgando leyes y promoviendo su uso.

Asimismo, si queremos que se establezca evidencia de que los datos existieron en un instante de tiempo determinado y que fueron firmados en ese instante, a la firma digital de los documentos, se le deberá añadir también una indicación del momento en que se realizó, normalmente conocido como “sello de tiempo”. De esta manera, quedarían resueltos casi todos los problemas dentro de las transacciones electrónicas.

Si bien es cierto que ya se están desarrollando aplicaciones independientes que permiten firmar documentos digitalmente y otras que permiten verificar las firmas, queda pendiente todavía poder integrar estas dos acciones dentro de los diferentes procesos en las organizaciones para optimizar su uso.

2.1. Objetivo general

El objetivo de la presente tesis es desarrollar un esquema de Firma Digital para una plataforma Web de comercio electrónico haciendo uso de la infraestructura adecuada que permita firmar documentos y contratos con cien por ciento de valor legal y que sean cien por ciento confiables.

2.2. Objetivos específicos

Con el desarrollo del proyecto de fin de carrera se pretende lograr los siguientes objetivos específicos aplicados sobre una plataforma de comercio electrónico:

- ✓ Implementar un módulo que permita obtener información de una persona a partir de la firma digital en un documento con formato PDF.
- ✓ Validar firmas digitales en documentos con formato PDF.
- ✓ Implementar un esquema de firma de documentos PDF electrónicos generados automáticamente en la plataforma.
- ✓ Implementar un esquema de firma de contratos que permita registrar las firmas tanto del comprador como del vendedor.
- ✓ Incluir el sello de tiempo en la firma de documentos y contratos.

2.3. Resultados esperados

Los resultados esperados son los siguientes:

- ✓ Módulo de obtención de datos a partir de documentos firmados.
- ✓ Esquema de firma de documentos electrónicos en PDF.
- ✓ Esquema de firma de contratos en PDF.
- ✓ Validación de firmas digitales en documentos PDF.
- ✓ Inclusión del sello de tiempo en la firma de documentos y contratos.

3. MARCO CONCEPTUAL

En esta sección se mencionarán algunos conceptos relacionados con la firma digital que permitan entender más adelante los procesos en los cuales se utilice.

3.1. Criptografía

3.1.1. Definición

La palabra criptografía proviene del griego “kryptos” que significa ocultar y “grafos” que significa escribir, literalmente sería “escritura oculta”.

La criptografía es el arte o ciencia de cifrar y descifrar información utilizando técnicas matemáticas que hace posible que la transferencia de información sea segura y que solo pueda ser leída por las personas a quienes va dirigida.

3.1.2. Objetivo

La finalidad de la criptografía es, en primer lugar, garantizar el secreto en la comunicación entre dos entidades (personas, organizaciones, etc.) y, en segundo lugar, asegurar que la información que se envía es auténtica en un doble sentido: que el remitente sea realmente quien dice ser y que el contenido del mensaje enviado, habitualmente denominado criptograma, no haya sido modificado en su tránsito.

3.1.3. Tipos

Se puede distinguir principalmente tres tipos de criptografía: la simétrica, la asimétrica y la híbrida.

3.1.3.1. Criptografía simétrica

Este tipo de criptografía se usa para cifrar y descifrar mensajes, se basa en el uso de una única clave conocida de antemano por ambas partes.

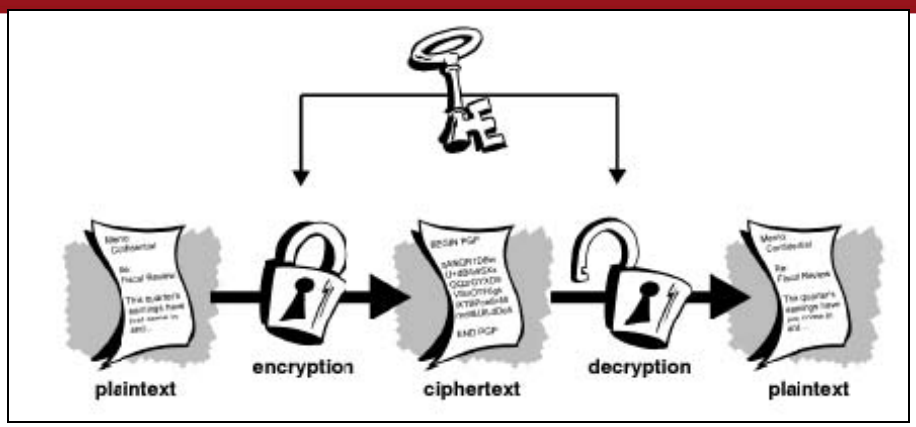
La seguridad de este tipo de criptografía radica principalmente en mantener en secreto la clave y no se preocupa necesariamente por el algoritmo de cifrado, es decir, que no es de mucha ayuda conocer el algoritmo que se utilizó.

Dado que toda la seguridad se centra en la clave, esta tiene que ser difícil de adivinar. Esto quiere decir que el abanico de claves posibles, es decir, el espacio de posibilidades de claves, debe ser amplio. Algunos ejemplos de algoritmos simétricos son 3DES, AES, Blowfish e IDEA.

Uno de los principales inconvenientes con este tipo de sistema no está ligado a su seguridad, sino al intercambio de claves. El canal utilizado para el intercambio debe ser lo suficientemente seguro. Una vez que el remitente y el destinatario hayan intercambiado las claves pueden usarlas para comunicarse con seguridad.

Otro problema es el número de claves que se necesitan. Si tenemos un número ***n*** de personas que necesitan comunicarse entre sí, se necesitan ***n/2*** claves para cada pareja de personas que tengan que comunicarse de modo privado. Esto puede funcionar con un grupo reducido de personas, pero sería imposible llevarlo a cabo con grupos más grandes.

Para solucionar estos problemas existen la criptografía asimétrica y la criptografía híbrida.



Fuente: http://www.tdec.com.br/Parceiros/PGP/PGP_Intro.htm

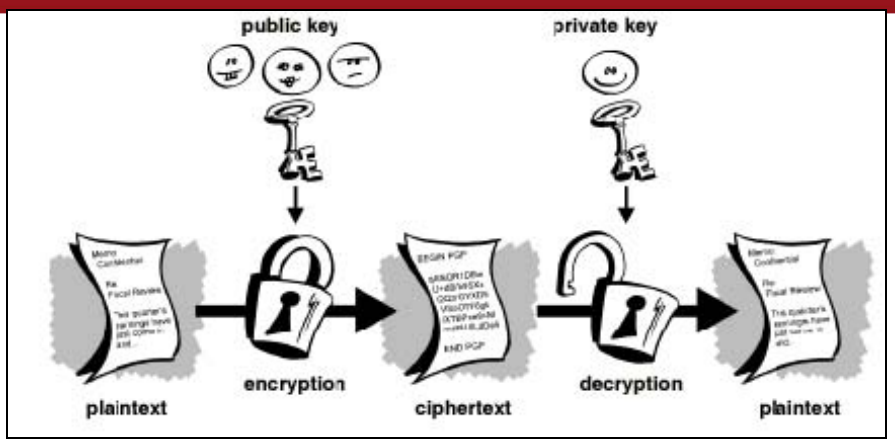
Ilustración 1: Proceso de criptografía simétrica.

3.1.3.2. Criptografía asimétrica

Este tipo de criptografía utiliza un par de claves (clave pública y clave privada) para el envío del mensaje, una para cifrar y otra para descifrar el mensaje; lo que se cifra en emisión con una clave, se descifra en recepción con la clave inversa.

Las claves públicas y privadas son otorgadas por la autoridad de certificación. Aunque ambas claves son propias de cada persona, la clave privada no se transmite nunca y se mantiene secreta. La clave pública, por el contrario, se puede y se debe poner a disposición de cualquiera, pues con esa finalidad fue creada. Esto no implica ningún problema de seguridad dado que es imposible deducir la clave privada a partir de la pública.

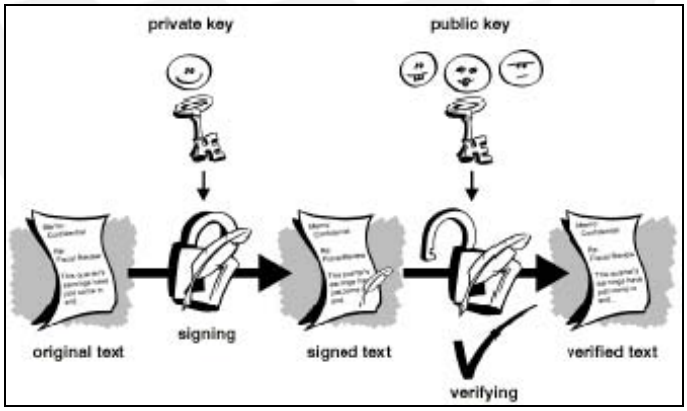
Se puede cifrar un mensaje con la clave pública y descifrar con la privada, dando confidencialidad al mensaje, ya que sólo podrá ser visto por el usuario con la correspondiente clave privada (Ver Ilustración 2)



Fuente: http://www.tdec.com.br/Parceiros/PGP/PGP_Intro.htm

Ilustración 2: Proceso de criptografía asimétrica (Encriptación con clave pública)

También, se puede cifrar con la clave privada y descifrar con la clave pública, lo cual no proporciona confidencialidad ya que cualquiera puede descifrar un mensaje cifrado con una clave secreta al poder obtener siempre la componente pública de su interlocutor, sin embargo, el hecho de cifrar un mensaje con la clave secreta de un usuario implica una identificación del usuario al igual que lo hace una firma, por lo que este proceso se conoce con el nombre de firma digital (Ver Ilustración 3)



Fuente: http://www.tdec.com.br/Parceiros/PGP/PGP_Intro.htm

Ilustración 3: Proceso de criptografía asimétrica (Encriptación con clave privada)

Las principales ventajas de este tipo de criptografía es que la clave secreta ya no tiene que transmitirse entre los interlocutores y tampoco es necesario tener claves diferentes para cada pareja de interlocutores, es suficiente con que cada usuario tenga su clave pública y su clave privada.

Algunos ejemplos de algoritmos asimétricos son: Diffie-Hellman, RSA, DSA, ElGamal, Criptografía de curva elíptica.

El más extendido de los sistemas de clave pública es el RSA, que fue desarrollado por Rivest, Shamir y Adleman, y se conoce como criptosistema RSA. Este algoritmo es reversible, es decir, además de permitir cifrar con la clave pública y descifrar con la privada, permite cifrar con la clave privada y descifrar con la clave pública.

3.1.3.3. Criptografía híbrida

Este tipo de criptografía utiliza tanto el cifrado simétrico como el asimétrico. Emplea el cifrado de clave pública para compartir una clave para el cifrado simétrico. El mensaje que se envía en el momento, se cifra usando la clave única (cifrado asimétrico) y se envía al destinatario.

Tanto PGP como GnuPG usan sistemas de cifrado híbridos.

3.2. Certificados

3.2.1. Certificado digital

Es un documento electrónico emitido por una empresa denominada “Autoridad de certificación” que garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública.

3.2.2. Autoridad de certificación

Una autoridad de certificación (AC o CA por sus siglas en inglés Certification Authority) o entidad de certificación, es una persona jurídica que presta servicios de emisión, gestión, cancelación u otros servicios inherentes a la certificación digital. Asimismo, puede asumir las funciones de registro o verificación. *[LEY01]* [1]

Las CAs disponen de sus propios certificados públicos, cuyas claves privadas asociadas son empleadas por las CAs para firmar los certificados que emiten. Un certificado de CA puede estar auto-firmado cuando no hay ninguna CA de rango superior que lo firme. Este es el caso de los certificados de CA raíz (autoridad de certificación raíz), el elemento inicial de cualquier jerarquía de certificación. Una jerarquía de certificación consiste en una estructura jerárquica de CAs en la que se parte de una CA auto-firmada, y en cada nivel, existe una o más CAs que pueden firmar certificados de entidades finales (personas, aplicación de software, etc) o bien certificados de otras CAs subordinadas, plenamente identificadas y cuya Política de Certificación sea compatible con la CA de rango superior.

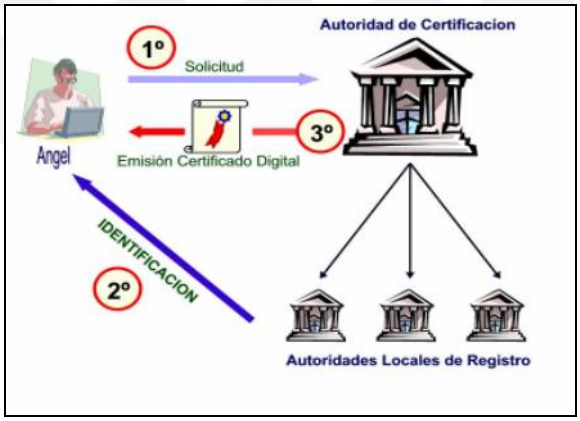
En el Perú la Entidad de Certificación Nacional para el Estado Peruano, que cumple funciones de Entidad de Registro y Verificación es la RENIEC (Registro Nacional de Identificación y Estado Civil).

[1] *Definición de Entidad de Certificación. Reglamento de la Ley de Firmas y Certificados Digitales Ley Nº 27269*

3.2.3. Proceso para la obtención de un certificado digital

El proceso para obtener un certificado digital es el siguiente (Ver Ilustración 4):

1. El solicitante se dirige a una empresa o entidad que tenga el carácter de Prestador de Servicios de Certificación y solicita de ellos las claves y el certificado digital correspondiente a las mismas. Este trámite generalmente se puede realizar presencialmente, acudiendo a dicha entidad o virtualmente, por medio de Internet, utilizando la página Web del Prestador de Servicios de Certificación.
2. El prestador de Servicios de Certificación comprobará la identidad del solicitante, bien sea directamente o por medio de entidades colaboradoras (Autoridades Locales de Registro), para lo cual se deberá mostrar el D.N.I. y si se trata de un representante de una sociedad (administrador, apoderado, etc.) o de cualquier otra persona jurídica, deberá acreditar documentalmente el cargo y las facultades del mismo (vigencia de poderes)
3. El prestador de Servicios de Certificación mediante los dispositivos técnicos adecuados crea las claves pública y privada que le corresponde al solicitante, y genera el certificado digital correspondiente a dichas claves.



Fuente: http://www.tuguialegal.com/firmadigital3.htm

Ilustración 4: Proceso de obtención de un certificado digital

3.3. Firma digital

Se puede citar dos definiciones para la firma digital que ayudarán a entender este concepto:

"La firma digital es una modalidad de firma electrónica que utiliza una técnica de criptografía asimétrica y que tiene la finalidad de asegurar la integridad del mensaje de datos a través de un código de verificación, así como la vinculación entre el titular de la firma digital y el mensaje de datos remitido." [IND01] [2]

De esta definición se puede concluir que la firma digital es un método criptográfico que permite garantizar la identidad del firmante así como la integridad de la información, la confidencialidad de los datos y el no repudio de la información.

3.4. Infraestructura de clave pública

Es una combinación de hardware y software, políticas y procedimientos de seguridad que permiten la ejecución segura de operaciones criptográficas (como el cifrado, la firma digital, el no repudio de transacciones electrónicas).

En el Perú la Autoridad Administrativa Competente de la Infraestructura Oficial de Firma Electrónica es INDECOPI (Instituto Nacional de Defensa de la Competencia y de la Propiedad Intelectual)

3.5. Formatos de firma

Las normas TS 101 733 y TS 101 903 definidas por la ETSI (European Telecommunications Standards Institute) definen los formatos técnicos de la firma electrónica. La primera se basa en el formato clásico PKCS#7 de RSA y la segunda en XMLDsig, firma XML especificada por el consorcio W3C.

[2] *Definición de INDECOPI para firma digital.*

El PKCS#7 es una estándar sobre la sintaxis de los mensajes criptográficos. Otros estándares de criptografía de clave pública que son de interés son:

- ✓ PKCS#11: Interfaz de dispositivo criptográfico. Define un API genérico de acceso a dispositivos criptográficos.
- ✓ PKCS#12: Estándar de sintaxis de intercambio de información personal. Define un formato de fichero usado comúnmente para almacenar claves privadas con su certificado de clave pública protegido mediante clave simétrica.

El XMLDsig es un estándar donde se define la sintaxis y las reglas de proceso para la creación de firmas digitales XML.

3.6. Sellado de tiempo

El sellado de tiempo (Timestamping) es un mecanismo en línea que permite demostrar que una serie de datos han existido, y siguen existiendo, y que no han sido alterados desde un instante específico en el tiempo, que es precisamente el que indica el sello.

Una Autoridad de Sellado de Tiempo actúa como tercera parte de confianza testificando la existencia de dichos datos electrónicos desde una fecha y hora concretos.

3.7. Función hash

Una función o algoritmo hash, en términos informáticos, se refiere a una función o método para generar claves o llaves que identifican de manera casi unívoca a un documento, archivo, registro, etc. Como resultado de la aplicación de esta función o algoritmo a un documento, archivo o registro dado se obtiene, lo que se denomina, un ***hash*** (llamado también “resumen”)

Una función hash o resumen, es una función para resumir o identificar probabilísticamente un gran conjunto de información, dando como resultado un conjunto imagen finito, generalmente menor (un subconjunto de los números naturales, por ejemplo).

Una propiedad fundamental del hashing (aplicación de la función hash) es que si los resultados de aplicar la función a dos documentos diferentes, entonces los dos documentos que generaron dichos resultados también lo son.

Sin embargo, es posible que existan claves resultantes iguales para objetos diferentes, ya que el rango de posibles claves es mucho menor que el de posibles objetos (las claves suelen tener alrededor de un centenar de bits, mientras que los objetos no tienen tamaño límite).

Cuando dos claves resultantes son iguales para objetos de entrada diferentes, se produce una colisión. Una buena función de hash es una que experimenta pocas colisiones en el conjunto esperado de entrada; es decir que se podrán identificar unívocamente las entradas (ver función inyectiva, anexo 1: Glosario de términos)

Las funciones o algoritmos hash son usadas en múltiples aplicaciones, como los arrays asociativos, criptografía, procesamiento de datos y firmas digitales, entre otros. Los algoritmos hash MD5 y SHA-1 son dos de los más populares.

4. ESTADO DEL ARTE

Actualmente existe un gran número de aplicaciones que permiten firmar digitalmente documentos de manera independiente, así como applets que se integran dentro de aplicaciones Web para firmar documentos. Por otro lado existen también aplicaciones independientes que permiten verificar firmas digitales.

Lo que se pretende ahora es integrar los procesos de una empresa de comercio con aplicaciones de firma digital propias.

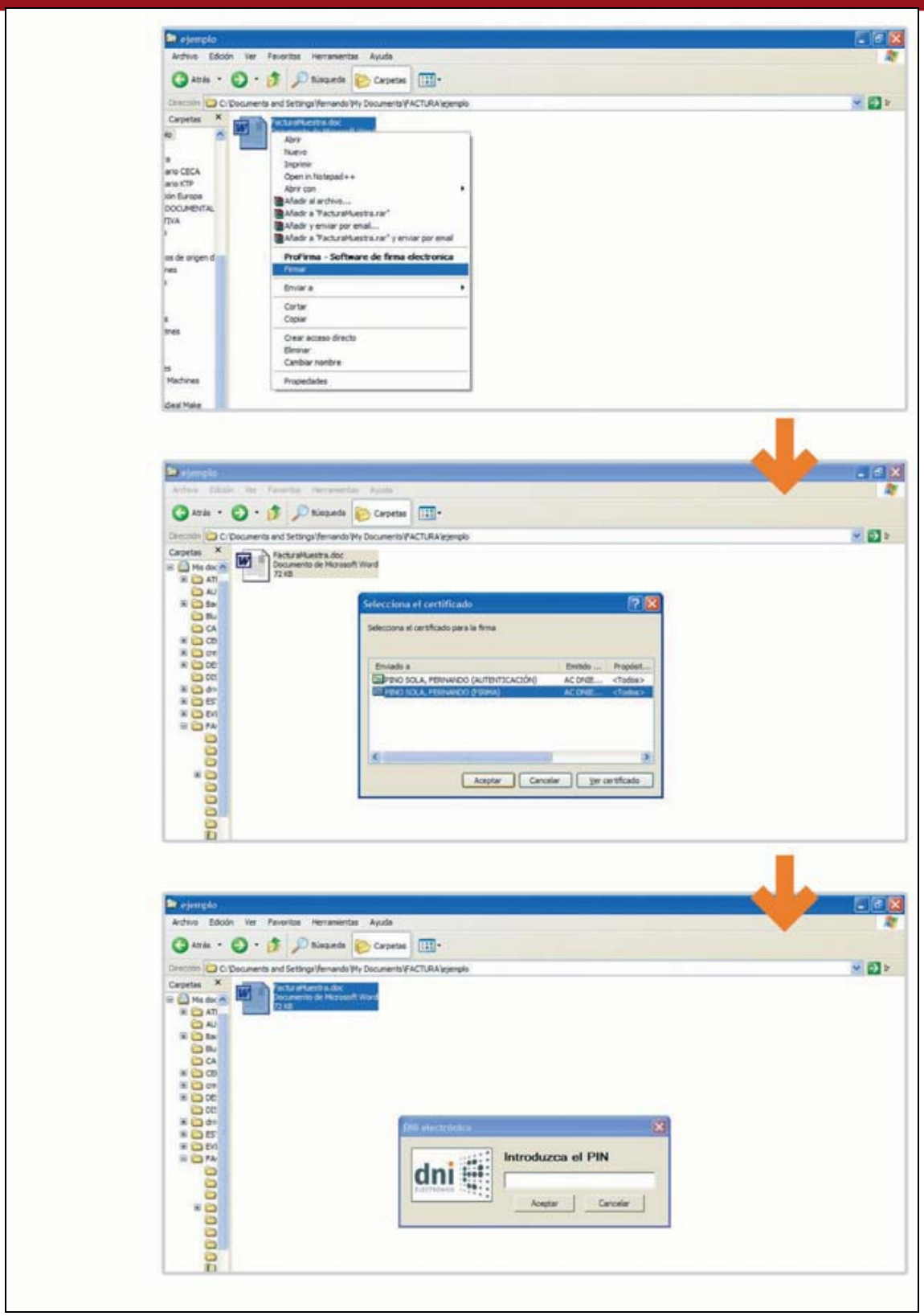
A continuación se mencionan algunas aplicaciones que permiten la firma digital de documentos.

Profirma es una aplicación para Windows que permite la firma de documentos independientemente de su formato. Esta aplicación genera archivos en formato PKCS#7 con la extensión ".fir", en cuyo interior se almacenan las firmas electrónicas, los certificados utilizados y el documento firmado, que se puede extraer para visualizarlo con el programa adecuado.

Con esta aplicación instalada en el ordenador, se asocia al menú contextual (el que aparece al pulsar el botón derecho del ratón sobre el archivo) una serie de opciones como "Firmar" o "Comprobar firma" (Ver Ilustración 5) *[PRO01]*[3].

Por otro lado **Adobe** tiene al Acrobat, software que permite crear documentos PDF y aplicar codificaciones, permisos y firmas electrónicas a los archivos. Los documentos PDF pueden visualizarse con el software gratuito Adobe Reader y con esta herramienta se permite además validar firmas electrónicas y comprobar la certificación de documentos. Además, mediante el Adobe LiveCycle Reader Extensions, los usuarios Adobe Reader podrán firmar electrónicamente los archivos PDF.

[3] *Software Profirma. http://www.efactura.org.es/index.php/documentos/*



Fuente: Manual Plan Avanza – La factura electrónica

Ilustración 5: Firma de documentos con Profirma.

5. PLAN DE PROYECTO

El cronograma del proyecto contempla todo el proceso de desarrollo del mismo, que va desde la investigación sobre el tema hasta la puesta en marcha en una plataforma de comercio electrónico. Este cronograma se ajusta al tiempo que la empresa consideró pertinente para el desarrollo de las aplicaciones.

A continuación se detalla el cronograma del proyecto (Ver Ilustración 6). El día laborable aquí considerado es de 6 horas, que es el tiempo diario que se dedicó para el desarrollo del proyecto:

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	Inicio del proyecto de implementación de Firma Digital	0 días	lun 02/04/07	lun 02/04/07	
2	**Investigación de la tecnología**	**18 días**	**lun 02/04/07**	**jue 19/04/07**	**1**
3	Revisar documentos de librerías para la generación de formularios PDF	6 días	lun 02/04/07	vie 06/04/07	
4	Revisar estándar de firma PKCS12	6 días	vie 06/04/07	vie 13/04/07	3
5	Revisar estándar de firma PKCS11	6 días	vie 13/04/07	jue 19/04/07	4
6	Implementación de Firma Digital en cualquier documento (fuera de la plataforma)	10 días	jue 19/04/07	mar 01/05/07	2
7	Análisis de la generación de documentos de la plataforma	3 días	mar 01/05/07	jue 03/05/07	6
8	Definición del esquema de generación de PDFs en la plataforma	6 días	jue 03/05/07	mié 09/05/07	7
9	Obtener datos de la Firma Digital	10 días	mié 09/05/07	lun 21/05/07	8
10	Modificar el proceso de registro de la plataforma	10 días	lun 21/05/07	mié 30/05/07	9
11	**Firmar documentos generados por la plataforma del seguimiento pedido**	**22 días**	**mié 30/05/07**	**vie 22/06/07**	**10**
12	Firmar el primer documento (Factura comercial)	4 días	mié 30/05/07	lun 04/06/07	
13	Docuemntos restantes (6 documentos)	18 días	lun 04/06/07	vie 22/06/07	12
14	**Proceso de firma de contratos**	**30 días**	**vie 22/06/07**	**mar 24/07/07**	**11**
15	Firma del vendedor	15 días	vie 22/06/07	lun 09/07/07	
16	Firma del comprador	15 días	lun 09/07/07	mar 24/07/07	15
17	Implementación de módulo con información de firma digital	3 días	mar 24/07/07	vie 27/07/07	14
18	Configuración de Servidor de sello de tiempo	30 días	lun 10/09/07	mié 10/10/07	17
19	Investigación sobre incrustación de sello de tiempo en PDF	10 días	mié 10/10/07	lun 22/10/07	18
20	Incrustación de sello de tiempo en los documentos PDF	10 días	lun 22/10/07	mié 31/10/07	19
21	Integración completa de firma digital en plataforma	0 días	mié 31/10/07	mié 31/10/07	20

Ilustración 6: Cronograma del proyecto

El proyecto incluye etapas claramente definidas que serán desarrolladas de manera secuencial por existir mucha dependencia entre ellas. Las fases consideradas son las siguientes:

- ✓ Investigación de la tecnología: en esta fase se buscó información sobre la tecnología de firma digital y se determinó la factibilidad de su implementación dentro de la plataforma.

- ✓ Implementación de un primer desarrollo fuera de la plataforma: en esta fase se consideró oportuno desarrollar un programa piloto independiente para tener una idea previa de la funcionalidad de las aplicaciones.
- ✓ Definición del esquema de firma aplicable sobre la plataforma, tomando en cuenta el entorno en que ella se ha desarrollado.
- ✓ Desarrollo del proceso de generación y firma de documentos PDF generados automáticamente por la plataforma.
- ✓ Desarrollo del proceso de generación y firma de contratos en PDF.
- ✓ Investigación sobre el sellado de tiempo en los documentos PDF.
- ✓ Inclusión del sello de tiempo a todos los documentos generados.

El tiempo total que se requirió para el desarrollo del proyecto fue de 162 días (972 horas)

6. DESCRIPCION Y SUSTENTACION DE LA SOLUCIÓN

En el año de 1976 dos investigadores Norteamericanos desarrollaron la teoría de lo que se denomina la criptografía de clave pública, y como consecuencia de ésta, la firma digital. En 1978 R. Rivest, A. Shamir, y L. Adleman, del MIT proponen método de firma digital, denominado RSA, que es el más usado desde su creación hasta la actualidad.

Este método obedece a los mismos principios que la firma manuscrita, es decir, tiene una acción de firmar y otra de verificación de la firma. La verificación de la firma confirma su exactitud, y hace prácticamente imposible la existencia de falsificaciones.

Con el uso de la firma digital se pretende dar solución a todos los problemas de comercio electrónico, donde la principal barrera era integrar la firma de documentos dentro de sus procesos electrónicos. De esta manera se permitirá firmar digitalmente documentos y contratos entre personas ubicadas en cualquier parte del mundo, sin moverse de sus oficinas, con total validez legal y de manera

cien por ciento segura. La plataforma Web en la cual se realice la operación, será quien la garantice, cumpliendo el rol de notario en estos procesos.

Por otro lado, se incluirá en la firma de los documentos un sello de tiempo que garantizará la existencia de los datos en la fecha y hora en que se realice la operación.

A continuación se detalla como ha sido utilizada la firma digital y el sello de tiempo en algunos procesos dentro de una plataforma de comercio electrónico.

6.1. Obtención de información a partir de la firma digital.

Dentro del proceso de registro o modificación de los datos de una empresa, el sistema incluirá una sección en la cual se puede firmar un documento y, como producto de este proceso, obtener información útil sobre el firmante, como son entre otros:

- ✓ El nombre del firmante.
- ✓ El e-mail del firmante.
- ✓ La empresa del firmante.
- ✓ La entidad certificadora.
- ✓ La certificadora raíz (Autoridad certificadora de más alto nivel. Ver Sección 3.2.2. Autoridad de certificación, CAPITULO 1)
- ✓ El e-mail de la entidad certificadora.
- ✓ La fecha inicio y fecha fin de la validez del certificado.

La información obtenida va a servir como evidencia para comprobar que los datos obtenidos coincidan con los reales.

6.2. Firma digital de documentos.

Para este caso el sistema permitirá firmar documentos (con el sello de tiempo incluido) que han sido generados automáticamente por el sistema, como son: facturas comerciales, albaranes, conocimientos de embarque, etc.

La firma del documento se podrá realizar haciendo uso de un archivo donde se almacena la clave privada (estándar PKCS12) o un dispositivo criptográfico (estándar PKCS11).

Con la firma digital de estos documentos se permite garantizar a las partes interesadas la identidad del firmante, así como la imposibilidad de negar la información firmada (no repudio)

6.3. Firma digital de contratos.

Para el caso de contratos, se pretende que las partes involucradas en éste, firmen un mismo documento.

En primer lugar firmará el vendedor, ya que legalmente es el que menos arriesga dentro de un proceso de compra. Una vez firmado el documento por el vendedor, éste será almacenado. El documento ya firmado por el vendedor no podrá ser visualizado por el comprador hasta que él también estampe su firma digital. Sin embargo ambos tendrán pleno acceso al documento en el estado que estaba antes de ser firmado.

Luego firmará el comprador. A partir de este momento ambas partes podrán tener acceso al documento firmado por ambos.

La razón de no mostrar al comprador el documento firmado solo por el vendedor, pasa por un tema de desventaja jurídica, ya que el comprador contaría con un

documento firmado por el vendedor y podría luego negarse a firmarlo pero si exigir su cumplimiento.

Al igual que en el caso de la firma de documentos, se incluirá simultáneamente con cada firma un sello de tiempo y se podrá utilizar un archivo donde se almacene la clave privada o un dispositivo criptográfico.

CAPITULO 2: DESCRIPCIÓN DEL PROCESO DE FIRMA DIGITAL Y SELLO DE TIEMPO

1. DESCRIPCIÓN DEL PROCESO DE FIRMA DIGITAL

El proceso de firma digital se basa en un proceso de encriptación asimétrica. Con los elementos descritos en el capítulo anterior (Ver Ilustración 7) ya es posible tener una idea sobre la forma en que se realizan los diferentes procesos de firma:

Fuente de imágenes: www.micit.go.cr/docs/Presentacion%203.ppt

Ilustración 7: Elementos utilizados en el proceso de firma digital

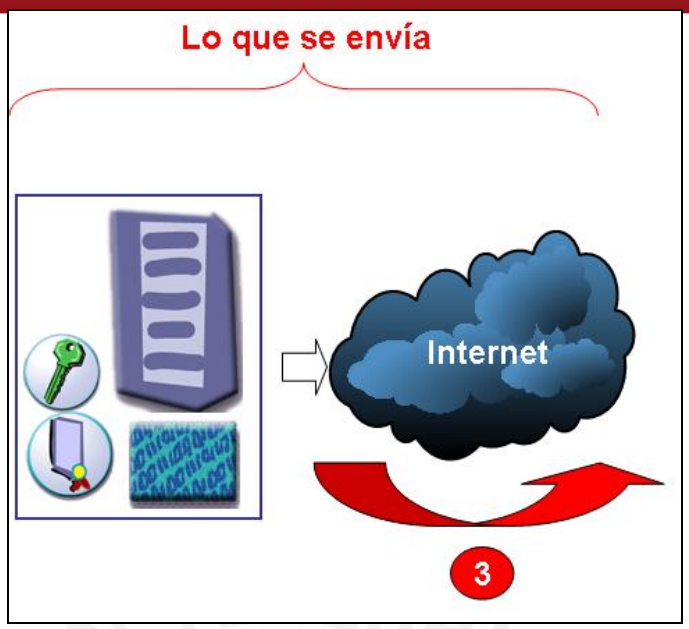
El proceso de firma digital se divide en dos subprocesos, la realización de la firma y la comprobación de la firma:

Realización de la firma (Ver Ilustración 8):

Fuente de imágenes: www.micit.go.cr/docs/Presentacion%203.ppt

Ilustración 8: Primer subproceso del proceso de firma digital - Realización de la firma

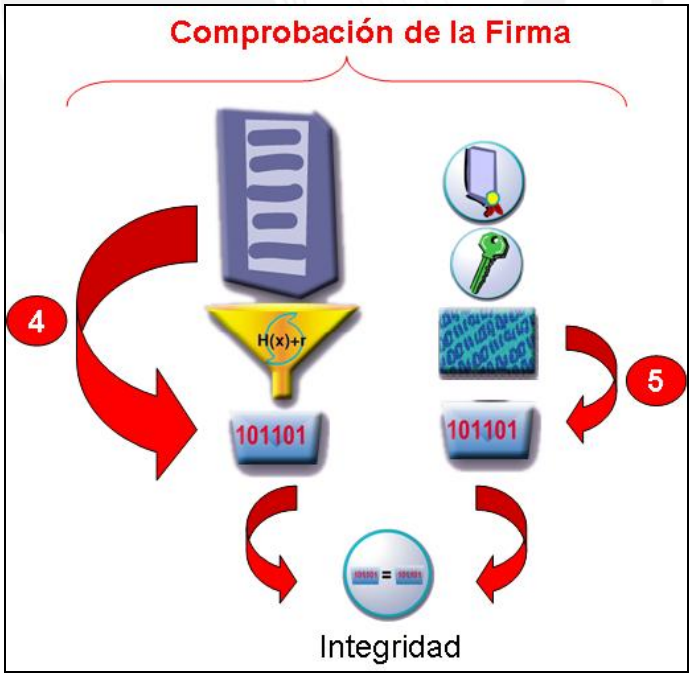
- ✓ 1 - Se tiene un documento original al cual se le aplica una función hash(o MAC) y se obtiene un hash o resumen del mismo.
- ✓ 2 – El resumen obtenido luego de aplicar la función hash, se cifra con la clave privada del titular del certificado.
- ✓ 3 - Se envía al destinatario el conjunto formado por el documento original, el hash (o MAC) firmado y el certificado (Ver Ilustración 9)



Fuente de imágenes: www.micit.go.cr/docs/Presentacion%203.ppt

Ilustración 9: Información que se envía al destinatario luego de la firma en el proceso de firma digital

Comprobación de la firma (Ver Ilustración 10):



Fuente de imágenes**:** www.micit.go.cr/docs/Presentacion%203.ppt

Ilustración 10: Segundo subproceso del proceso de firma digital - **Comprobación de la firma**

- ✓ El receptor recibe el conjunto formado por el documento original, el hash firmado y el certificado.
- ✓ 5 - A partir del certificado se extrae la clave pública del remitente. Con la clave pública desencripto el mensaje que fue cifrado con la clave privada y obtengo el hash que se calculó en origen.
- ✓ 4 - Se calcula el hash a partir del documento original.
- ✓ Comparamos el hash del documento y el que se obtuvo con la clave pública, y deben ser idénticos (Integridad)

2. CARACTERISTICAS DE LA FIRMA DIGITAL

Luego de haber descrito el proceso de firma podemos concluir que un sistema de firma electrónica permite cumplir con las siguientes funciones de seguridad:

- ✓ Integridad
- ✓ Autenticación
- ✓ No repudio
- ✓ Confidencialidad
- ✓ Momento de la firma

Además de estas características existen otros principios de seguridad que igualmente se resuelven, según las características del certificado y del protocolo empleado, como disponibilidad, auditabilidad, fiabilidad, detección de mensajes faltantes en una secuencia, acuse de recibo, validez de poderes y evidencia legal. *[PLA01]* [4]

2.1. Integridad

Consiste en asegurar que la información no ha sufrido cambios no autorizados, ya sea de manera accidental o intencional, una vez firmado.

[4] *Plan Avanza. Manual Factura Electrónica.*

2.2. Autenticación

Consiste en identificar al emisor del mensaje y sus atributos principales, asegurando que es la persona que figura como firmante en el documento.

2.3. No repudio

Esta característica está ligada a la anterior, pues al autenticar la identidad del firmante, se evita que el emisor del mensaje pueda negar haberlo firmado. De esta manera se garantiza que el firmante está de acuerdo con el contenido del documento o se vincula con él de alguna forma (como autor, como revisor, dándose por enterado, etc.)

2.4. Confidencialidad

Consiste en impedir que la información del documento, contrato o firma, sea vista por usuarios no autorizados.

2.5. Momento de la firma

Si la firma se lleva a cabo de forma correcta, incorporando información temporal fehaciente, es posible saber que el documento existía antes de un momento dado, que es el que se indica como momento de la firma.

3. DESCRIPCIÓN DEL PROCESO DE SELLADO DE TIEMPO

El proceso de sellado de tiempo proporciona pruebas de la existencia de determinada información en un instante de tiempo determinado (prueba de existencia). Si la información fue firmada por el peticionario, antes de ser enviado a

la Autoridad de Sellado de Tiempo (TSA), entonces este servicio proporciona una prueba, de la existencia de la información y de que el documento que contiene la información fue firmado en ese instante de tiempo. *[CAM01]* [5]

El proceso de sellado de tiempo está distribuido en las siguientes fases:

a) Usuario peticionario:

- ✓ Petición de sellado: En el proceso de petición, el usuario peticionario debe realizar la preparación del objeto a sellar (Timestamp Request RFC3161).

b) Autoridad de sellado de tiempo:

- ✓ Revisión de la petición de sellado: Este componente está diseñado para revisar que la información consignada en la petición esté completa y sea correcta. Si el resultado es positivo, los datos se envían como entrada a la Generación de Sello de Tiempo.
- ✓ Generación del parámetro tiempo: Este componente usa una fuente de confianza para la distribución de parámetros de tiempo. Estos parámetros serán usados como entrada al proceso de Generación de Sellado de Tiempo.
- ✓ Generación de Sello de Tiempo: Esta función es la responsable de crear un sello de tiempo que asocie el instante de tiempo actual, un número de serie único, los datos proporcionados para el sellado de tiempo y garantizar los requerimientos de política a los que se adhiere.
- ✓ TimeStamp Token (TST): Este componente calcula el indicador del sello de tiempo que se devolverá al cliente. El indicador del sello de tiempo es el que realmente se utiliza para poner el sello a los datos.

c) Usuario peticionario:

- ✓ Recepción del sello: El proceso de verificación del sello, en el que se evalúa la autenticidad del sello de tiempo recibido.

[5] *Proceso de Sello de Tiempo tomado de Camerfirma.*

3.1. Proceso de petición (TimeStamp Request)

El proceso comienza con la petición por parte de un tercero, de un sello de tiempo a aplicar a un determinado documento.

Para ello el peticionario debe generar una petición TimeStampRequest según la RFC3161.

Los parámetros que el peticionario deberá enviar son por ejemplo:

- ✓ Hash del documento a sellar.
- ✓ Nombre del algoritmo de hash a utilizar.
- ✓ Identificador de la política (ID) bajo la cual se proporcionará el sello (opcional)
- ✓ Versión: Número de versión de la sintaxis utilizada (opcional)
- ✓ MessageImprint: Contiene el hash de los datos que se quiere sellar. La longitud del hash tiene que coincidir con la longitud de hash del algoritmo utilizado. El algoritmo utilizado podrá ser SHA1, MD5 o RIPEMD160.
- ✓ ReqPolicy: indica a la TSA la política bajo la cual quiere que se le proporcione el sello de tiempo (opcional)

3.2. Proceso de sellado

En el proceso de sellado, el sistema realiza diferentes acciones, primero realiza una revisión de la petición, verificando la correcta estructuración del objeto TimeStamp Request y el origen de la misma. Durante esta verificación se comprueba que se han introducido los parámetros esperados como el algoritmo de hash y la política de sellado y que son correctos.

Posteriormente se obtiene el parámetro de tiempo de la fuente segura de tiempo y se genera el token de tiempo, que es firmado electrónicamente con las claves privadas de sellado de la Autoridad de Sellado de tiempo.

Finalmente se genera la respuesta TimeStamp Response, siguiendo las especificaciones de la RFC3161, disponiendo de la siguiente representación.

3.3. Proceso de verificación

La entidad que recibe el TimeStamp Response correspondiente a su petición, debe validarlo y extraer de él información necesaria para su almacenamiento.

CAPITULO 3: MARCO LEGAL DE LA FIRMA DIGITAL EN EL PERÚ

1. INTRODUCCIÓN

Los avances técnicos, en materia informática vienen planteando diversos retos al ser humano, tanto sociales como jurídicos, especialmente, debido a la creciente demanda de operaciones electrónicas por medio de las llamadas redes abiertas. Para enfrentar estas nuevas situaciones, que en muchos casos generan consecuencias legales de gran magnitud, se viene regulando el uso de la firma electrónica, así como de las firmas y certificados digitales. Al respecto, en el Perú se han aprobado una serie de cambios legislativos que permiten el uso de tales elementos técnicos, con la finalidad de acreditar fehacientemente a las personas que manifiestan su voluntad por medios electrónicos y evitar de esta forma el repudio de sus operaciones.

Con la promulgación de la Ley 27269 - Ley de Firmas y Certificados Digitales, del año 2000 (Ver Anexo 3: Ley de firmas y certificados digitales), se continua en forma específica, el desarrollo legislativo, iniciado en 1991, con la aprobación del

Decreto Legislativo N° 6811, modificado posteriormente por la Ley 26612, normas que permiten el uso del documentos electrónicos con pleno valor probatorio, previo cumplimiento de ciertos requisitos, tales como: tener equipos debidamente acreditados ante el órgano estatal competente, contar con la participación de un funcionario que otorgue fe pública durante el proceso de conversión de los documentos con soporte en papel a documentos en soporte digital o electrónico, etc. Posteriormente, en mayo del 2002, se aprueba el Reglamento de la Ley de Firmas y Certificados Digitales.

Los aspectos regulados por la Ley de Firmas y Certificados Digitales han permitido que en el Perú comience un desarrollo legislativo paulatino y constante, a través de la aprobación de una serie de normas con carácter jurídico-informático, que vienen siendo aplicables a los diferentes ámbitos de la vida en sociedad. Es así, por ejemplo, que contamos con normas específicas que se aplican al ámbito Civil, por la aprobación de la Ley 27291, que modificó el Código Civil, permitiendo la utilización de los medios electrónicos para comunicar la manifestación de voluntad, así como la utilización de la firma electrónica.

2. LEY DE FIRMAS Y CERTIFICADOS DIGITALES - LEY N° 27269

En el Perú se aprobó en mayo del año 2000 la Ley de Firmas y Certificados Digitales – Ley 27269, que regula la utilización de la firma electrónica, a la cual se le otorga la misma validez y eficacia jurídica que el uso de una firma manuscrita u otra análoga que conlleve manifestación de voluntad. De esta manera, la firma manuscrita y la electrónica quedan en igualdad jurídica; con este hecho el Perú dio un paso importante en la regulación de las medidas técnicas de seguridad, que permiten el flujo permanente y seguro de datos e información a través de redes abiertas. *[LEG01]* [6]

[6] *Régimen Jurídico de la Firma Electrónica en el Perú. José Francisco Espinoza Céspedes.*

3. LAS FIRMAS ELECTRÓNICAS EN LA LEY N° 27269

Conforme a lo establecido en Ley 27269, se entiende por firma electrónica, a cualquier símbolo basado en medios electrónicos, utilizados o adoptados por una parte(firmante) con la intención precisa de vincularse o autenticar un documento, cumpliendo todas o algunas de las funciones características de una firma manuscrita *[LEG]*. En el proceso de autenticación no se otorga certificación notarial ni fe pública. El proceso de autenticación se define como aquel proceso técnico que permite determinar la identidad de la persona que firma electrónicamente, en función del mensaje firmado por éste y al cual se le vincula.

4. AMBITO DE APLICACIÓN DE LA LEY Nº 27269

La Ley 27269, se aplica a cualquier firma electrónica que se pone, añade o se asocia a un mensaje de datos, con la finalidad de vincular e identificar al firmante, así como garantizar la autenticación e integridad de los mensajes electrónicos. Con la vinculación e identificación previa de los firmantes, se garantiza el no repudio de la información firmada. Se tendrá la certeza de la persona que emitió el mensaje, es decir, la autenticación. La firma también garantiza la integridad, es decir, que el documento no será modificado.

5. LA FIRMA DIGITAL EN LA LEY Nº 27269

La Ley 27269 define la firma digital como una firma electrónica que utiliza una técnica de criptografía asimétrica, con la cual se asegura la integridad del mensaje de datos y vincula al titular de la firma con el mensaje remitido.

Esta técnica criptográfica hace uso de dos claves, una privada, que solo la conoce el titular y la usa para generar la firma digital y una clave pública con la cual el receptor verifica la firma.

6. EL CERTIFICADO DIGITAL

Un certificado digital es un documento electrónico generado y firmado digitalmente por una entidad de certificación, la cual vincula un par de claves con una persona determinada, confirmando su identidad.

La información que contiene el certificado digital son datos que identifican indubitablemente al suscriptor, los datos que identifiquen a la Entidad de Certificación, la clave pública, la metodología de verificación de la firma digital del suscriptor, el número de serie del certificado, la vigencia del certificado y la firma digital de la Entidad de Certificación. Cualquier información adicional debe ser solicitada a la entidad pertinente, la cual deberá comprobar fehacientemente la veracidad de ésta. *[LEG]*

7. INSTITUCIONES EN EL MARCO DE LA LEY N° 27269

La Ley de Firmas y Certificados Digitales hace mención a las instituciones sobre las que se sentarán las bases para el desarrollo de las firmas y certificados digitales en nuestro país, estas son:

a) La Autoridad Administrativa Competente, que es el organismo público responsable de acreditar a las entidades de certificación y a las entidades de registro o verificación, de reconocer los estándares tecnológicos aplicables en la Infraestructura Oficial de Firma Electrónica, de supervisar dicha Infraestructura, así como la reglamentación y prestación de servicios de valor añadido relacionados con la misma y las otras funciones señaladas en el Reglamento o aquéllas que requiera en el transcurso de sus operaciones.

b) La Entidad de Certificación, es la persona jurídica que presta indistintamente servicios de producción, emisión, gestión, cancelación u otros servicios inherentes a la certificación digital. Asimismo, puede asumir las funciones de registro o verificación.

c) La Entidad de Certificación Extranjera, es aquella que cumple las mismas funciones señaladas en b), pero no se encuentra domiciliada en el país, ni inscrita en los Registros Públicos del Perú, conforme a la legislación de la materia, es decir, la Nueva Ley General de Sociedades - LEY Nº 26887.

d) La Entidad de Registro o Verificación, es aquella persona jurídica encargada del levantamiento de datos, la comprobación de éstos respecto a un solicitante de certificado digital, la aceptación y autorización de las solicitudes para la emisión de certificados digitales, así como de la aceptación y autorización de las solicitudes de cancelación de certificados digitales.

Es necesario resaltar, que la entidad de registro recabará los datos personales del solicitante de la firma digital, directamente de éste y para los fines señalados en la Ley. En cuanto a la información relativa a las claves privadas y datos que no sean materia de certificación, ésta se mantendrá bajo la reserva correspondiente. Sólo puede ser levantada por orden judicial o pedido expreso del suscriptor de la firma digital. *[LEG]*

8. LOS CERTIFICADOS EMITIDOS POR ENTIDADES EXTRANJERAS

Conforme a la modificatoria establecida en la Ley Nº 27269, la autoridad administrativa competente puede reconocer certificados de firmas digitales emitidos en el extranjero por alguna Entidad Extranjera, teniendo estos la misma validez y eficacia jurídica reconocidas en la presente Ley.

La Ley en su capítulo tercero, trata los siguientes conceptos:

a) Certificación Cruzada, es el acto por el cual una entidad de certificación acreditada reconoce la corrección y validez de un certificado digital emitido por otra entidad de certificación, sea nacional, extranjera o internacional, previa autorización de la autoridad administrativa competente; y asume tal certificado como si fuera de propia emisión, bajo su responsabilidad. *[LEG]*

b) Acuerdos de Reconocimiento Mutuo, son aquellos por medio del cual la autoridad administrativa competente puede suscribir acuerdos de reconocimiento mutuo con entidades similares del extranjero, a fin de reconocer la validez de certificados digitales otorgados en el extranjero y extender la validez de la Infraestructura Oficial de Firma Digital. Los acuerdos de reconocimiento mutuo deben garantizar en forma equivalente las funciones exigidas por la Ley.

c) Reconocimiento, es el proceso a través del cual la autoridad administrativa competente, equipara y reconoce oficialmente a las entidades de certificación extranjeras. *[LEG]*

CAPITULO 4: CONSTRUCCIÓN DEL SOFTWARE

1. INTRODUCCIÓN

Las aplicaciones de firma digital se han desarrollado dentro de una plataforma de comercio electrónico de diferentes productos a nivel mundial. La plataforma tiene como objetivo, entre otras cosas, eliminar las barreras idiomáticas y hacer del comercio entre países, exportaciones e importaciones, una tarea menos complicada y tediosa. Bajo estas circunstancias, se ha pensado en un sistema experto que ayudaría a las pequeñas y medianas empresas a realizar exportaciones e importaciones sin la necesidad de entender el idioma de la empresa con la cual se negocia, ya que cada empresa verá los documentos en su propio lenguaje.

Por otro lado, al hablar de documentación (facturas, contratos, etc.) es evidente que el tema de seguridad legal es muy importante y determinante; aquí es donde entra a tallar la firma digital como un elemento fundamental para dar seguridad legal a dichos documentos.

A continuación, un ejemplo que permita entender lo antes explicado:

Si una empresa francesa esta interesada en algún producto de una empresa peruana y quiere llegar a un acuerdo comercial, probablemente de la manera convencional sería un poco complicado y costoso concretar dicho acuerdo, por los siguientes aspectos:

- ✓ Se necesitaría la presencia física de los representantes de ambas empresas para concretar el acuerdo y firmar los documentos.
- ✓ La documentación sería administrada directamente por cada empresa.
- ✓ La diferencia idiomática podría ocasionar malas interpretaciones.

La plataforma desarrollada, permite, entre otras cosas, eliminar dichos inconvenientes, de la siguiente manera:

- ✓ Todas las características del acuerdo pueden ser realizadas con las opciones de la plataforma y la documentación puede ser firmada haciendo uso de una firma digital.
- ✓ La plataforma es un sistema que te ayuda a determinar que tipo de documentación se necesita para exportar o importar a un país determinado.
- ✓ La plataforma permite ver la documentación en el idioma propio de cada empresa.

El ejemplo antes mencionado permite ver, entre otras cosas, la necesidad de la firma digital. El presente proyecto de tesis se ha enfocado en el desarrollo de las aplicaciones de firma digital para la plataforma, que son:

- ✓ Obtención de datos del autor de la firma, a partir de un documento firmado.
- ✓ Firma de documentos generados automáticamente por la plataforma.
- ✓ Firma de contratos.

- ✓ Inclusión del sello de tiempo en la firma de documentos y contratos.

2. LIBRERÍAS UTILIZADAS

Para la implementación de la firma digital y sello de tiempo dentro de la plataforma de comercio se hizo uso de las librerías iTEXT 1.4, BouncyCastle e IAIK.

iText es una biblioteca Open Source para crear y manipular archivos PDF, RTF, y HTML en Java. Fue escrita por Bruno Lowagie, Paulo Soares, y otros; está distribuida bajo la Mozilla Public License con la LGPL como licencia alternativa [LIB01] (Ver http://www.lowagie.com/iText/)

La librería BouncyCastle es un desarrollo en lenguaje Java, con una ingente cantidad de algoritmos de criptografía. Hace uso del lenguaje Java sin ningún tipo de restricción, por lo que se convierte en una plataforma ideal para los procesos de encriptar la información [LIB02] (Ver http://www.bouncycastle.org/)

IAIK (Institute for Applied Information Processing and Communication) define un conjunto de APIs e implementaciones de funciones criptográficas basadas en Java. Es gratuita, para fines de investigación únicamente [LIB03] (Ver http://jce.iaik.tugraz.at/)

3. CÓMO FIRMAR Y PONER EL SELLO DE TIEMPO PROGRAMÁTICAMENTE A UN PDF EN JAVA

En esta sección se revisará cómo se logra introducir programáticamente la firma digital y el sello de tiempo dentro de un PDF en java, haciendo uso de las librerías antes mencionadas. (Para código más extenso Ver Anexo 2: Código fuente para firma digital y sello de tiempo)

Se puede firmar y poner el sello de tiempo a un PDF usando el siguiente método:

```
public void signPDF(SignerKeystore sks, PdfStamper stp, TSAClient tsc) {
        try {
            PdfSignatureAppearance sap = stp.getSignatureAppearance();
            setAppearance(sks,sap);

            // Configure PDF signature dictionary (PdfName.ADOBE_PPKLITE works too)
            PdfName filter = PdfName.ADOBE_PPKMS;
            PdfName subfilter = PdfName.ADBE_PKCS7_SHA1;
            PdfSignature dic = new PdfSignature(filter,subfilter);

            // dic.setName (null); // optional - PdfPKCS7 uses the certificate CN
            dic.setReason(sap.getReason());
            dic.setLocation(sap.getLocation());
            dic.setContact(sap.getContact());
            dic.setDate(new PdfDate(sap.getSignDate())); // time-stamp will over-rule this
            sap.setCryptoDictionary(dic);

            // Estimate signature size, creating a 'fake' one using fake data
            // (SHA1 length does not depend upon the data length)
            ByteArrayInputStream  bais= new ByteArrayInputStream("fake".getBytes());
            byte[] estSignature = genPKCS7Signature(bais, null);
            int contentEst = estSignature.length + ((tsc == null) ? 0 :tsc.getTokenSizeEstimate());

            // Preallocate excluded byte-range for the signature content (hex encoded)
            HashMap exc = new HashMap();
            exc.put(PdfName.CONTENTS, new Integer(contentEst * 2 + 2));
            sap.preClose(exc);

            // Get the true data signature, including a true time stamp token
            byte[] encodedSig = genPKCS7Signature(sap.getRangeStream(),tsc);
            if (contentEst + 2 < encodedSig.length) {
               throw new Exception("Timestamp size estimate " + contentEst +
                                   " is too low for actual " +
                                   encodedSig.length);
            }
            // Copy signature into a zero-filled array, padding it up to estimate
            byte[] paddedSig = new byte[contentEst];
            System.arraycopy(encodedSig, 0, paddedSig, 0, encodedSig.length);

            // Finally, load zero-padded signature into the signature field /Content
            PdfDictionary dic2 = new PdfDictionary();
            dic2.put(PdfName.CONTENTS, new PdfString(paddedSig).setHexWriting(true));
            sap.close(dic2); // closes sap.close()
            System.out.println("Fin de firmado");

        } catch (Throwable t) {
            System.out.println("Signing failed" + t);
            t.printStackTrace();
        }
 }
```

A continuación se explicará cada una de las secciones del método:

public void signPDF(SignerKeystore sks, PdfStamper stp, TSAClient tsc).

Analizando los parámetros del método se tiene:

- ✓ ***SignerKeystore sks:*** interfaz que provee acceso al certificado.
- ✓ ***PdfStamper stp:*** clase que permite manejar contenido dentro de un PDF, permite crear una o múltiples firma digitales. Para crear una firma usamos el método ***static PdfStamper createSignature(…)*** de la siguiente manera:

createSignature(PdfReader reader, OutputStream os, char pdfVersion)
Crea una firma digital en un documento.

createSignature(PdfReader reader, OutputStream os, char pdfVersion, File tempFile)
Crea una firma digital en un documento.

createSignature(PdfReader reader, OutputStream os, char pdfVersion, File tempFile, boolean append)
Crea una firma digital en un documento, posiblemente como una nueva revisión, permite múltiples firmas.

Parametros:
reader – documento original. No puede ser reusado.
os - output stream.
pdfVersion – la nueva versión del pdf o '\0' para mantener la misma versión como documento original.
tempFile – ubicación del archivo temporal. Si es un directorio el archivo deberá crearse ahí. Si es un archivo será usado directamente. El archivo será borrado al terminar a menos que sea nulo. En ese caso el documento puede ser recuperado directamente del archivo temporal. Si es nulo ningún archivo temporal será creado y se usará la memoria.
append – si es true la firma y todo el contenido será añadido como una nueva revisión, así no invalidará las firmas existentes.
Returns:
PdfStamper
Throws:
DocumentException - si error
IOException - si error

Ejemplo de uso:

```
// Para una firma
PdfStamper stp = PdfStamper.createSignature(reader, fout, '\0');

// Para multiples firmas
PdfStamper stp = PdfStamper.createSignature(reader, fout, '\0', null, true);
```

Fuente: http://www.1t3xt.info/api/com/lowagie/text/pdf/PdfStamper.html

- ✓ ***TSAClient tsc:*** Interfase utilizada por el constructor de firma digital PdfPKCS7 para llamar a la autoridad de sello de tiempo (Time Stamp Authority) proporcionando el token de sello de tiempo compatible con el estándar RFC 3161 (Request for Comments) Dicha interfaz será implementada, en este caso, por la clase TSAClientBouncyCastle y su invocación se hará de la siguiente manera:

```
String sT="http://200.16.7.78:8000/tsa"; // serverTimestamp
String uT=""; // usernameTimestamp
String pT=""; // passwordTimestamp
Int tokSzEstimate=4900; // Tamaño estimado en bytes del time stamp token.

TSAClient  tsc = new TSAClientBouncyCastle(sT, uT, pT,tokSzEstimate);
```

Lo que se pretende es crear un objeto TSAClientBouncyCastle e inicializar los valores que se necesitan para realizar el sellado de tiempo como: servidor de TS, usuario y contraseña del servidor, tamaño del token de sello de tiempo. El constructor de la clase será el siguiente:

```
/**
* Constructor.
* Note the token size estimate is updated by each call, as the token
* size is not likely to change (as long as we call the same TSA using
* the same imprint length).
* @param url String - Time Stamp Authority URL (i.e."http://tsatest1.digistamp.com/TSA")
* @param username String - user(account) name
* @param password String - password
* @param tokSzEstimate int - estimated size of received time stamp token (DER encoded)
*/

public TSAClientBouncyCastle(String url, String username, String password, int tokSzEstimate) {
    this.tsaURL = url;
    this.tsaUsername = username;
    this.tsaPassword = password;
    this.tokSzEstimate = tokSzEstimate;
}
```

Fuente: http://threebit.net/mail-archive/itext-questions/msg06121.html

Luego de haber analizado los parámetros, ahora se verá cada una de las secciones del método:

- ✓ Se inicializa la apariencia del PDF: con el objeto *PdfStamper* se obtiene una instancia de la clase *PdfSignatureAppearance*, esta clase se encargará de las opciones de cifrado y la apariencia que tiene una firma. Se agregará, por ejemplo, razón de la firma, ubicación, podemos indicar si queremos una firma visible o no, etc.

```
PdfSignatureAppearance sap = stp.getSignatureAppearance();
setAppearance(sks,sap);

/**
* Setup signature appearance. Override to define specifics.
* @param sap PdfSignatureAppearance
*/
protected void setAppearance(SignerKeystore sks,PdfSignatureAppearance sap) {
        sap.setCrypto(null, sks.getChain(), null, PdfSignatureAppearance.WINCER_SIGNED);
        sap.setLocation("Peru");
        sap.setReason("Firma de documento");
        // Comentar la siguiente linea para firma invisible
        sap.setVisibleSignature(new Rectangle(50, 15, 200, 65), 1, null);
}
```

- ✓ Se configura el diccionario de la firma digital del PDF: Se conoce como diccionario al conjunto de atributos que tiene una firma digital como son el nombre del firmante, fecha y hora de la firma, lugar, razón, etc. Este diccionario será introducido dentro de la instancia del PdfSignatureAppearance.

```
// Configure PDF signature dictionary (PdfName.ADOBE_PPKLITE works too)
PdfName filter = PdfName.ADOBE_PPKMS;
PdfName subfilter = PdfName.ADBE_PKCS7_SHA1;
PdfSignature dic = new PdfSignature(filter,subfilter);

// dic.setName (null); // optional - PdfPKCS7 uses the certificate CN
dic.setReason(sap.getReason());
dic.setLocation(sap.getLocation());
dic.setContact(sap.getContact());
dic.setDate(new PdfDate(sap.getSignDate())); // time-stamp will over-rule this
sap.setCryptoDictionary(dic);
```

- ✓ Se estima el tamaño de la firma digital y el sello de tiempo y reservamos el espacio en el PDF: se crea un objeto falso para estimar el tamaño de la firma y sello de tiempo; luego se reserva el espacio en el *PdfSignatureAppearance* del PDF.

```
// Estimate signature size, creating a 'fake' one using fake data
// (SHA1 length does not depend upon the data length)
ByteArrayInputStream  bais= new ByteArrayInputStream("fake".getBytes());
byte[] estSignature = genPKCS7Signature(bais, null);
int contentEst = estSignature.length + ((tsc == null) ? 0 :tsc.getTokenSizeEstimate());

// Preallocate excluded byte-range for the signature content (hex encoded)
HashMap exc = new HashMap();
exc.put(PdfName.CONTENTS, new Integer(contentEst * 2 + 2));
sap.preClose(exc);
```

- ✓ Se obtiene la firma y el sello de tiempo reales (expresado como un arreglo de bytes) a través del método *genPKCS7Signature*. Este método recibe

información suficiente para obtener la firma y sello de tiempo, como son: los datos a firmar y un objeto con información de sellado de tiempo.

```
// Get the true data signature, including a true time stamp token
byte[] encodedSig = genPKCS7Signature(sap.getRangeStream(),tsc);
if (contentEst + 2 < encodedSig.length) {
            throw new Exception("Timestamp size estimate " + contentEst +
                        " is too low for actual " + encodedSig.length);
}

/**
* Generate the PKCS7 encoded signature
* @param data InputStream - data to digest
* @param doTimestamp boolean - true to include time-stamp
* @return byte[]
* @throws Exception
*/
protected byte[] genPKCS7Signature(InputStream data, TSAClient tsc)throws  Exception {
      // assume sub-filter is adobe.pkcs7.sha1 entonces hasRSAdata=true
      PdfPKCS7 sgn = new PdfPKCS7(sks.getPrivateKey(), sks.getChain(),null,"SHA1",
sks.getProvider().getName(),true);
      byte[] buff = new byte[2048];
      int len = 0;
      while ((len = data.read(buff)) > 0) {
          sgn.update(buff, 0, len);
      }
      return sgn.getEncodedPKCS7(null, null, tsc);
}
```

✓ Se copia la firma en un nuevo arreglo con el tamaño estimado: se genera un nuevo objeto para rellenar todo el espacio reservado.

```
// Copy signature into a zero-filled array, padding it up to estimate
byte[] paddedSig = new byte[contentEst];
System.arraycopy(encodedSig, 0, paddedSig, 0, encodedSig.length);
```

✓ Se carga el arreglo rellenado en el campo de firma: aquí se crea un nuevo diccionario donde se pone la firma, representada como un arreglo de bytes, y se cierra la instancia del PdfSignatureAppearance con el diccionario.

```
// Finally, load zero-padded signature into the signature field /Content
PdfDictionary dic2 = new PdfDictionary();
dic2.put(PdfName.CONTENTS, new PdfString(paddedSig).setHexWriting(true));
sap.close(dic2); // closes sap.close()
System.out.println("Fin de firmado");
```

Finalmente la invocación al método se hará de la siguiente manera:

```
String sT="http://200.16.7.78:8000/tsa"; // serverTimestamp
String uT=""; // usernameTimestamp
String pT=""; // passwordTimestamp

// is: InputStream del archivo ".pfx"
// strPIN: contraseña del archivo
SignerKeystore sks = new SignerKeystorePKCS12(is, strPIN);
TSAClient      tsc = new TSAClientBouncyCastle(sT, uT, pT);
SignerPdf  signer = new SignerPdf(sks, tsc);
signer.signPDF(sks,reader, fout);
```

Todo el extracto de código aquí mostrado representa una solución específica, existen muchas variantes y formas para poder firmar y poner el sello de tiempo a un PDF. En el anexo 2 (Código fuente para firma digital y sello de tiempo) se encuentran las clases y métodos más importantes para cumplir con el objetivo.

4. OBTENCIÓN DE INFORMACIÓN A PARTIR DE LA FIRMA DIGITAL

La sección del registro dentro de la plataforma permite que los usuarios al momento de registrar su empresa puedan usar su firma digital, de esta manera ellos dan una imagen mucho más seria y fiable a las empresas que estén interesadas en negociar con ellos, ya que la firma digital tiene por si misma un respaldo legal que la protege y la hace confiable a nivel mundial.

En base a este requerimiento, se planteó en un primer momento el siguiente esquema (Ver Ilustración 11):

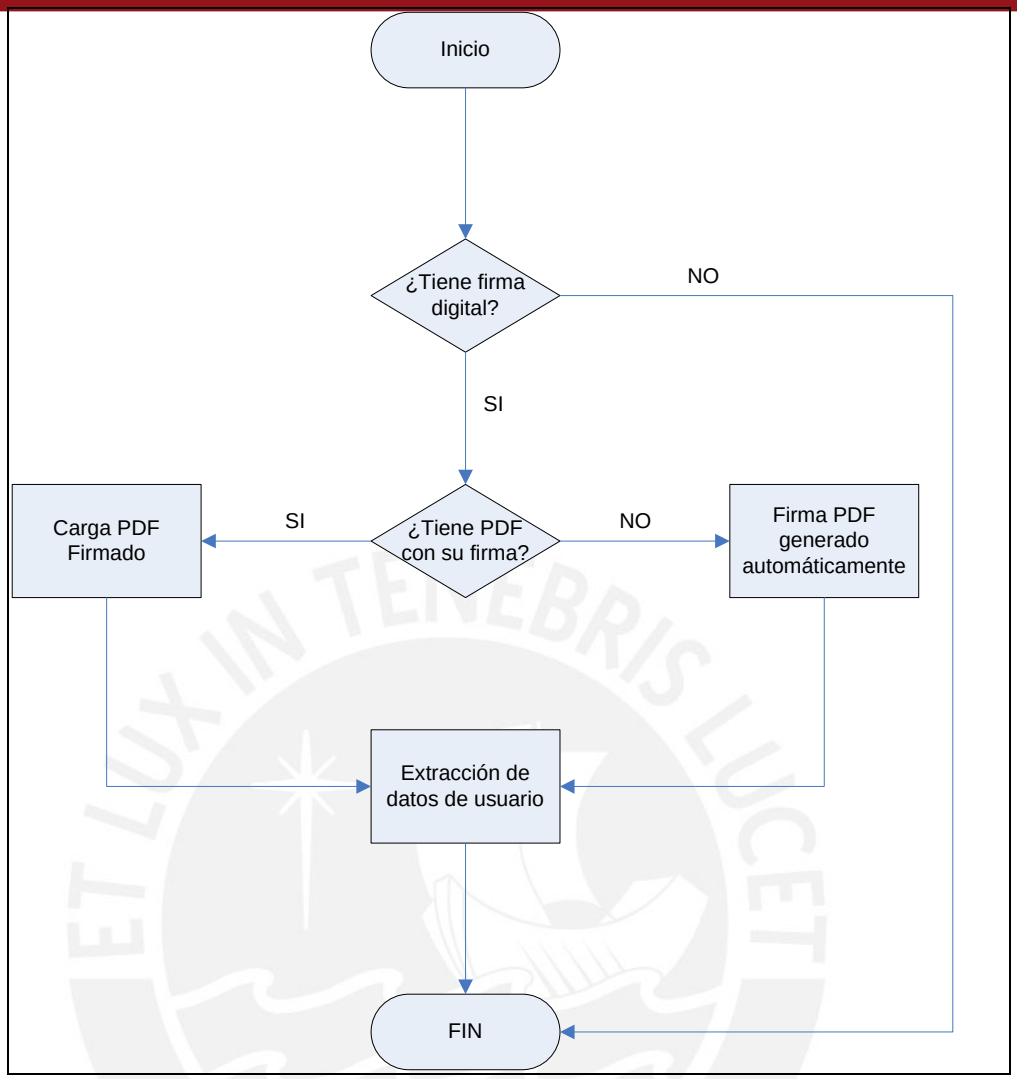


Ilustración 11: Esquema inicial para la obtención de datos de firma digital en el proceso de registro

Según este esquema los datos del usuario podrían ser extraídos firmando un documento o cargando un PDF firmado.

El problema que surgía con este esquema es que el usuario podría subir un PDF firmado digitalmente por una persona que no fuera él y suplantar su identidad. Es por ello que ese primer esquema fue modificado y se obliga al usuario, si este cuenta con firma digital, firmar un PDF con la plataforma, lo cual reduce considerablemente el riesgo de suplantar a otro (Ver Ilustración 12)

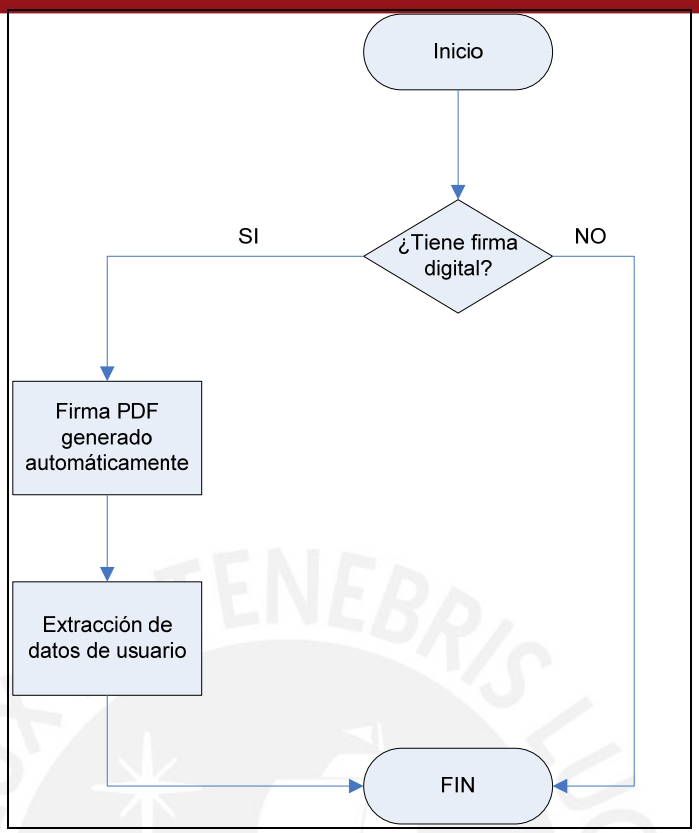


Ilustración 12: Esquema final para la obtención de datos de firma digital en el proceso de registro

Producto de este proceso de registro se podrá obtener información útil sobre el firmante (Ver Ilustración 13), como son:

- ✓ El nombre del firmante.
- ✓ La entidad certificadora.
- ✓ La empresa del firmante.
- ✓ La fecha inicio y fecha fin de la validez del certificado, entre otros.

La información obtenida va a servir como evidencia para comprobar que los datos obtenidos coincidan con los reales.

Se puede firmar un archivo usando un almacén de clave privada o usando algún dispositivo criptográfico:

a) Si se firma usando un almacén de claves privadas.

La información solicitada es la siguiente:

- ✓ Archivo PDF para firmar.
- ✓ KeyStore - Archivo de almacenamiento de claves (.pfx o .p12).
- ✓ Pin.

b) Si se firma usando un dispositivo criptográfico.

La información solicitada es la siguiente:

- ✓ Archivo PDF para firmar.
- ✓ Certificado digital.
- ✓ Pin.

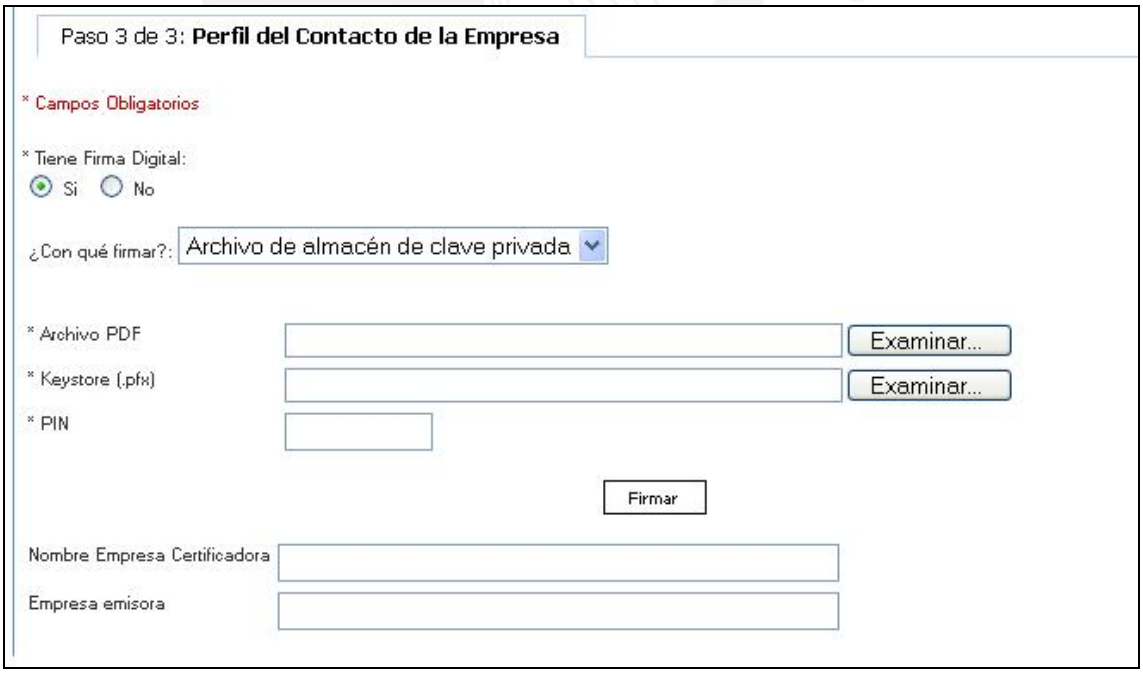


Ilustración 13: Proceso de firma de documento de registro

5. FLUJO PRINCIPAL DE FIRMA DE DOCUMENTOS Y CONTRATOS

La firma de documentos y contratos se puede hacer usando un almacén de clave privada o usando algún dispositivo criptográfico. Al presionar el botón firmar, el flujo es el siguiente:

- ✓ Le aparecerá un cuadro de diálogo (Ver Ilustración 14) donde debe seleccionar con que firmar: chip criptográfico, chip de memoria, archivo de almacén de claves privadas.

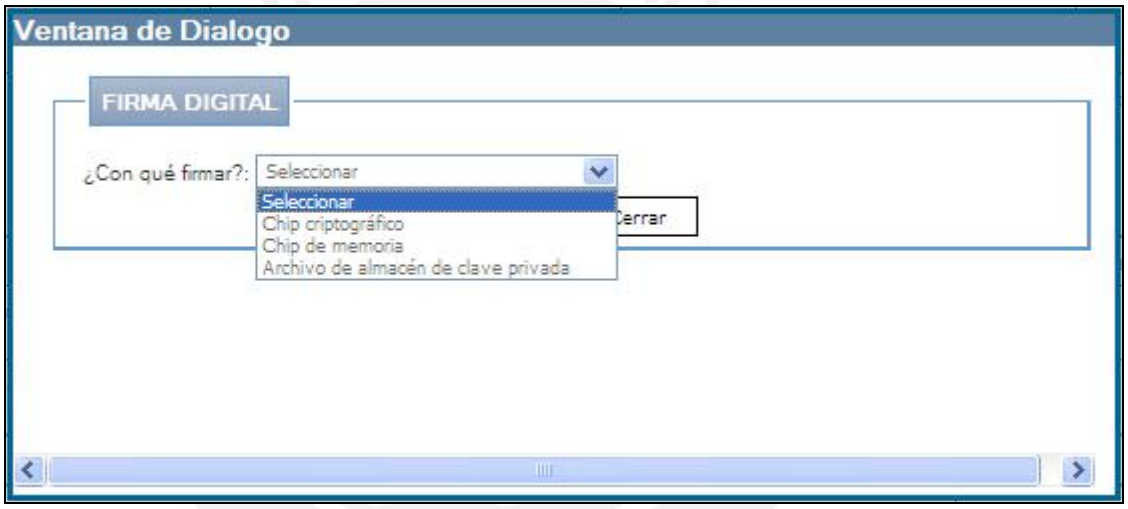


Ilustración 14: Ventana para firmar documentos

a) Si la opción seleccionada es “Archivo de almacén de clave privada” se debe seleccionar un archivo de almacén de clave privada, dar su pin y presionar “Enviar” (Ver Ilustración 15)

Ilustración 15: Firma con un archivo de almacén de claves privadas

b) Si la opción seleccionada es “Chip criptográfico” o “Chip de memoria” se debe seleccionar el certificado digital, dar su pin y presionar “Enviar” (Ver Ilustración 16)

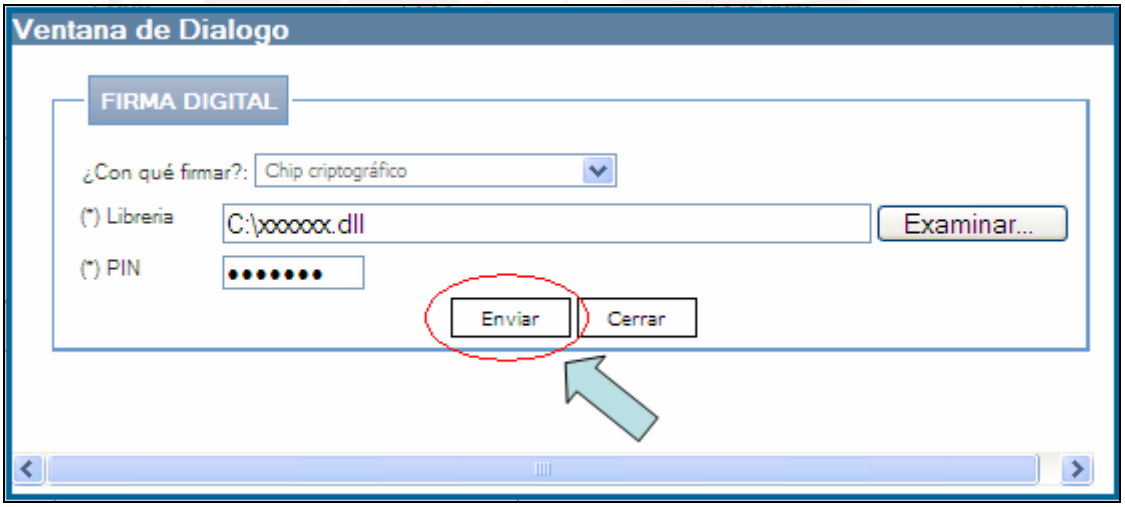


Ilustración 16: Firma con un dispositivo criptográfico

6. FIRMA DIGITAL DE DOCUMENTOS

Para este caso el sistema permitirá firmar documentos (con el sello de tiempo incluido) que han sido generados automáticamente por el sistema. Los documentos que se han considerado con este esquema son los siguientes:

- ✓ Facturas comerciales.
- ✓ Albaranes.
- ✓ Factura Interna.
- ✓ Conocimiento de embarque multimodal - FFI.
- ✓ Conocimiento de embarque marítimo - BL.
- ✓ Conocimiento de embarque por carretera - CMR
- ✓ Packing List.

La firma del documento se podrá realizar haciendo uso de un archivo de almacén de clave privada (estándar PKCS12) o un dispositivo criptográfico (estándar PKCS11).

Con la firma digital de estos documentos se permite garantizar a las partes interesadas la identidad firmante del documento, así como la imposibilidad de negar la información firmada.

El proceso es el siguiente:

- ✓ El afiliado al momento de abrir un archivo, selecciona la opción firmar (Ver Ilustración 17).

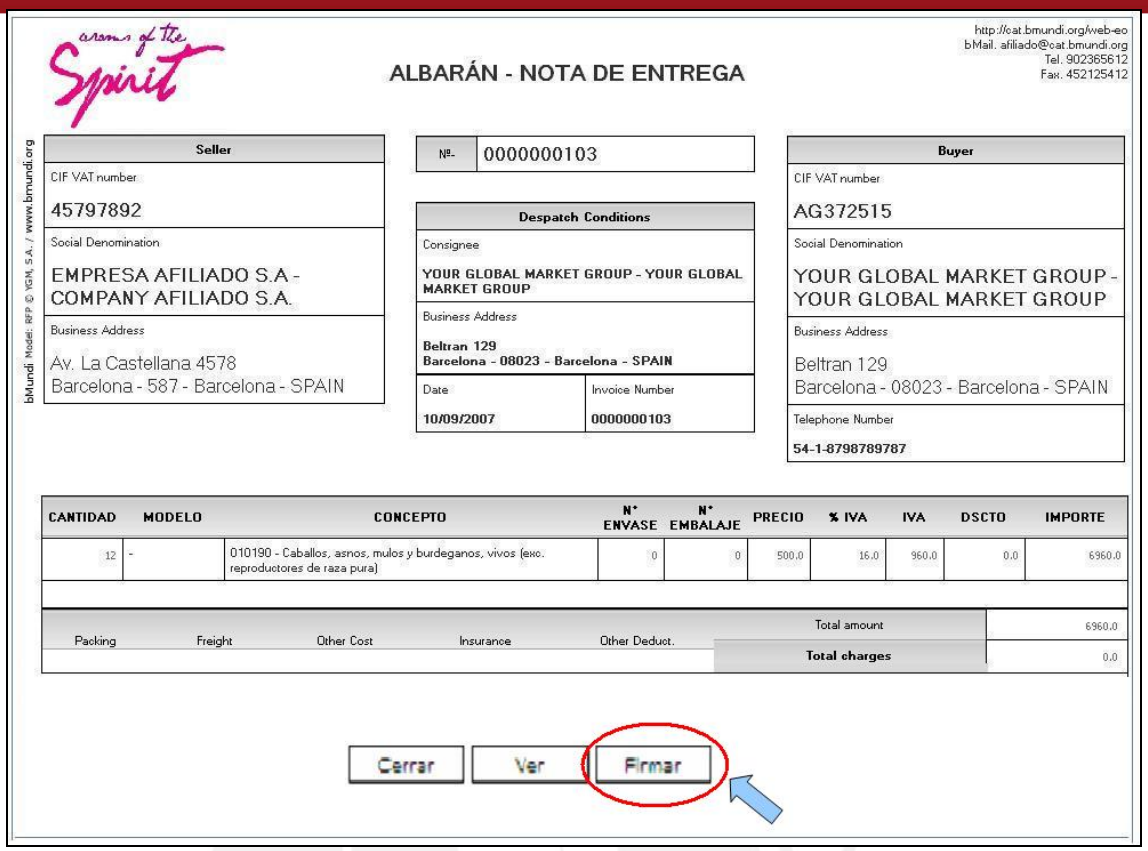
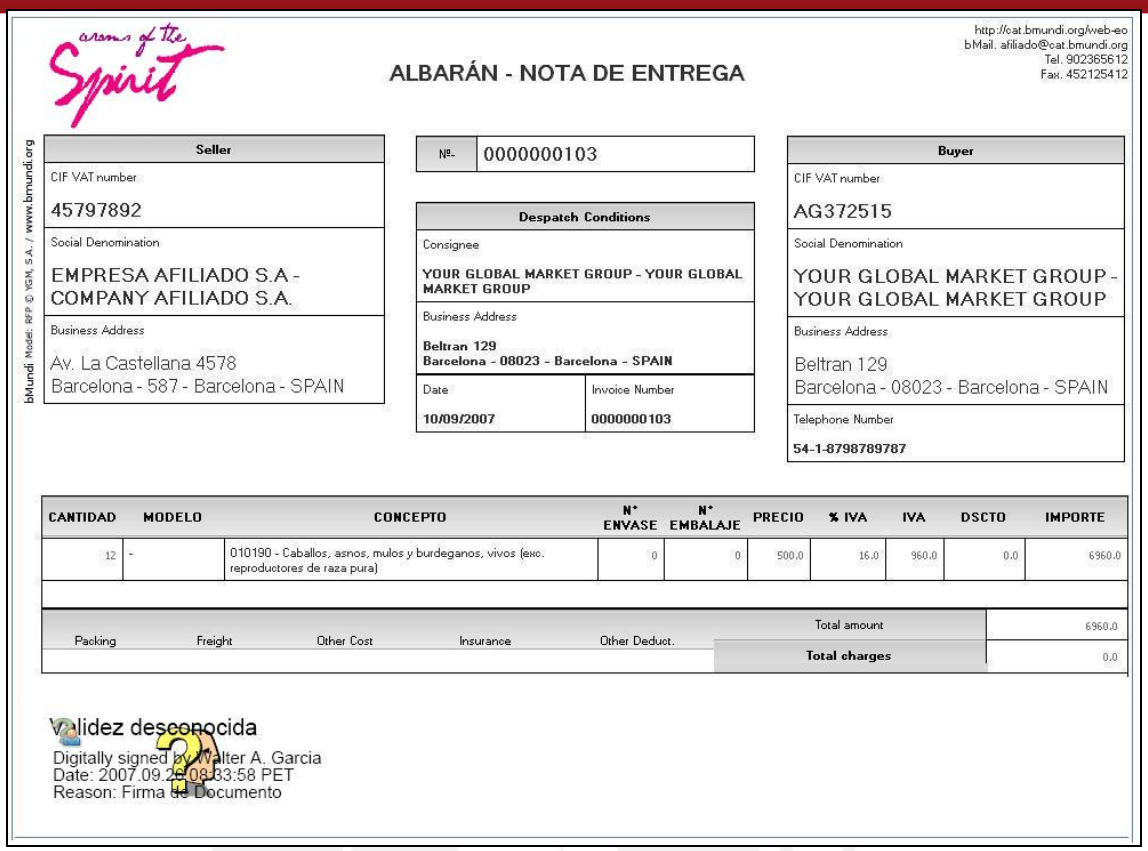


Ilustración 17: Firma de documentos

- ✓ Sigue el flujo principal (Ver Capítulo 4, Sección 4)
- ✓ Resultado del flujo la firma se estampa en la parte inferior de la primera hoja del documento (Ver Ilustración 18).



arms of the Spirit

ALBARÁN - NOTA DE ENTREGA

http://cat.bmundi.org/web-eo
bMail. afiliado@cat.bmundi.org
Tel. 902365612
Fax. 452125412

bMundi Model: RFP © YGM, S.A. / www.bmundi.org

Seller
CIF VAT number
45797892
Social Denomination
EMPRESA AFILIADO S.A - COMPANY AFILIADO S.A.
Business Address
Av. La Castellana 4578 Barcelona - 587 - Barcelona - SPAIN

Nº-	0000000103

Despatch Conditions	
Consignee	
YOUR GLOBAL MARKET GROUP - YOUR GLOBAL MARKET GROUP	
Business Address	
Beltran 129 Barcelona - 08023 - Barcelona - SPAIN	
Date	Invoice Number
10/09/2007	0000000103

Buyer
CIF VAT number
AG372515
Social Denomination
YOUR GLOBAL MARKET GROUP - YOUR GLOBAL MARKET GROUP
Business Address
Beltran 129 Barcelona - 08023 - Barcelona - SPAIN
Telephone Number
54-1-8798789787

CANTIDAD	MODELO	CONCEPTO	Nº ENVASE	Nº EMBALAJE	PRECIO	% IVA	IVA	DSCTO	IMPORTE
12	-	010190 - Caballos, asnos, mulos y burdeganos, vivos (exc. reproductores de raza pura)	0	0	500.0	16.0	960.0	0.0	6960.0

Packing	Freight	Other Cost	Insurance	Other Deduct.	Total amount	6960.0
					Total charges	0.0

Validez desconocida
Digitally signed by Walter A. Garcia
Date: 2007.09.26 08:33:58 PET
Reason: Firma de Documento

Ilustración 18: Documento firmado

7. FIRMA DIGITAL DE CONTRATOS

Para el caso de contratos, se pretende que las partes involucradas (comprador y vendedor) firmen un mismo documento. Por cada contrato se generarán dos documentos PDF, un documento que será firmado por ambas partes, otro que será una copia del anterior que siempre estará disponible sin firmas para ser visualizados en cualquier momento.

En un primer momento, es el vendedor quien firmará ya que legalmente es el que menos arriesga dentro de un proceso de compra. Una vez firmado el documento este será almacenado y no podrá ser visualizado por el comprador hasta que el también estampe su firma.

En segunda instancia firmará el comprador y a partir de este momento ambas partes podrán tener acceso al documento firmado por ambos.

La razón de no mostrar el documento firmado solo por el vendedor, pasa por un tema de desventaja jurídica, ya que una de las partes contaría con un documento firmado por la otra parte y podría luego negarse a firmarlo.

Al igual que la firma de documentos se incluirá en cada firma el sello de tiempo y se podrá utilizar un archivo de almacén de clave privada o un dispositivo criptográfico.

Los contratos considerados para el presente proyecto son:

- ✓ Contrato Compra Venta
- ✓ Contrato Sistemático de Agente
- ✓ Contrato Sistemático de Representante
- ✓ Contrato Sistemático de Distribuidor

La firma de estos contratos se realizará únicamente por el Administrador de la empresa, debido a que es el máximo representante de la empresa y es el que asumirá la responsabilidad que acarree la firma de estos; pero la negociación podrá ser realizada por cualquier colaborador de la empresa.

El proceso es el siguiente:

- ✓ El vendedor al momento de abrir un contrato, selecciona la opción firmar (Ver Ilustración 19).

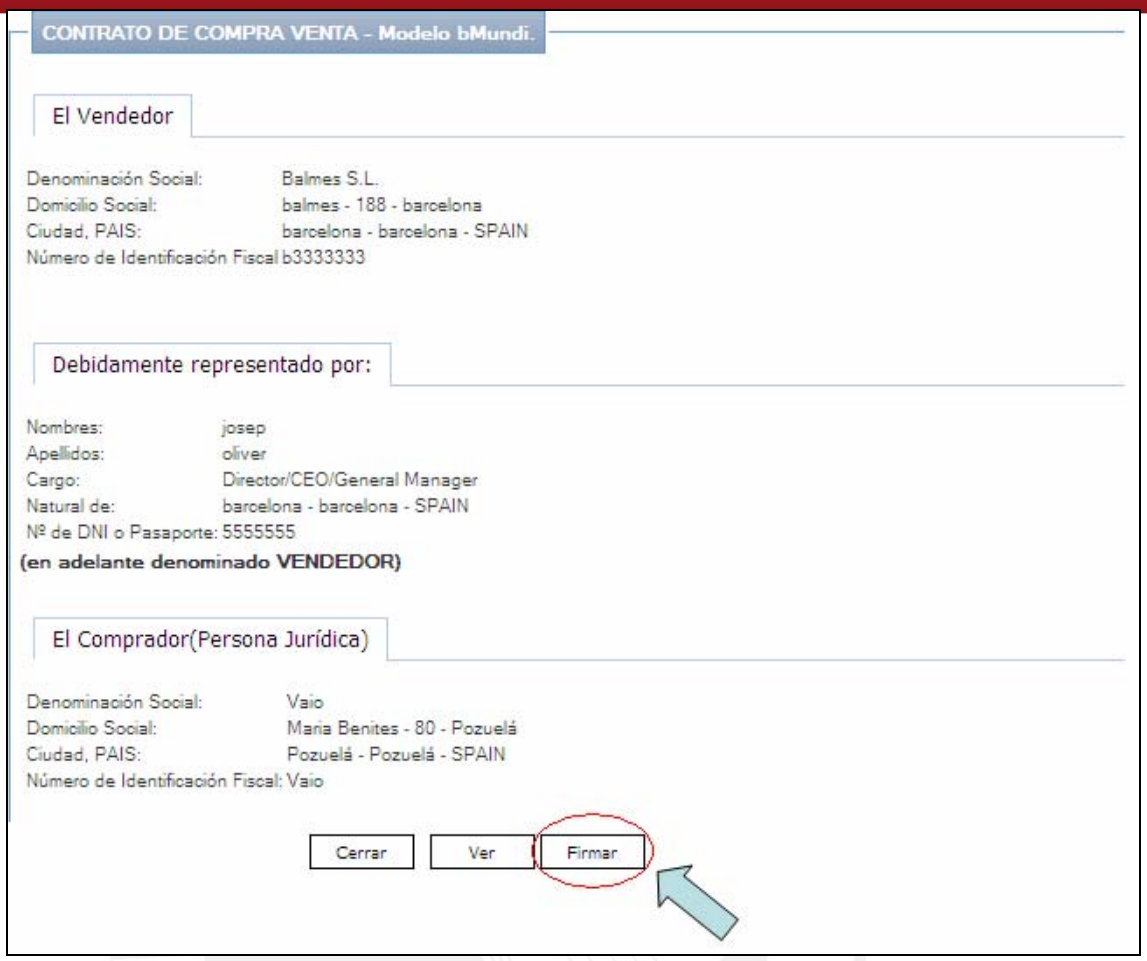
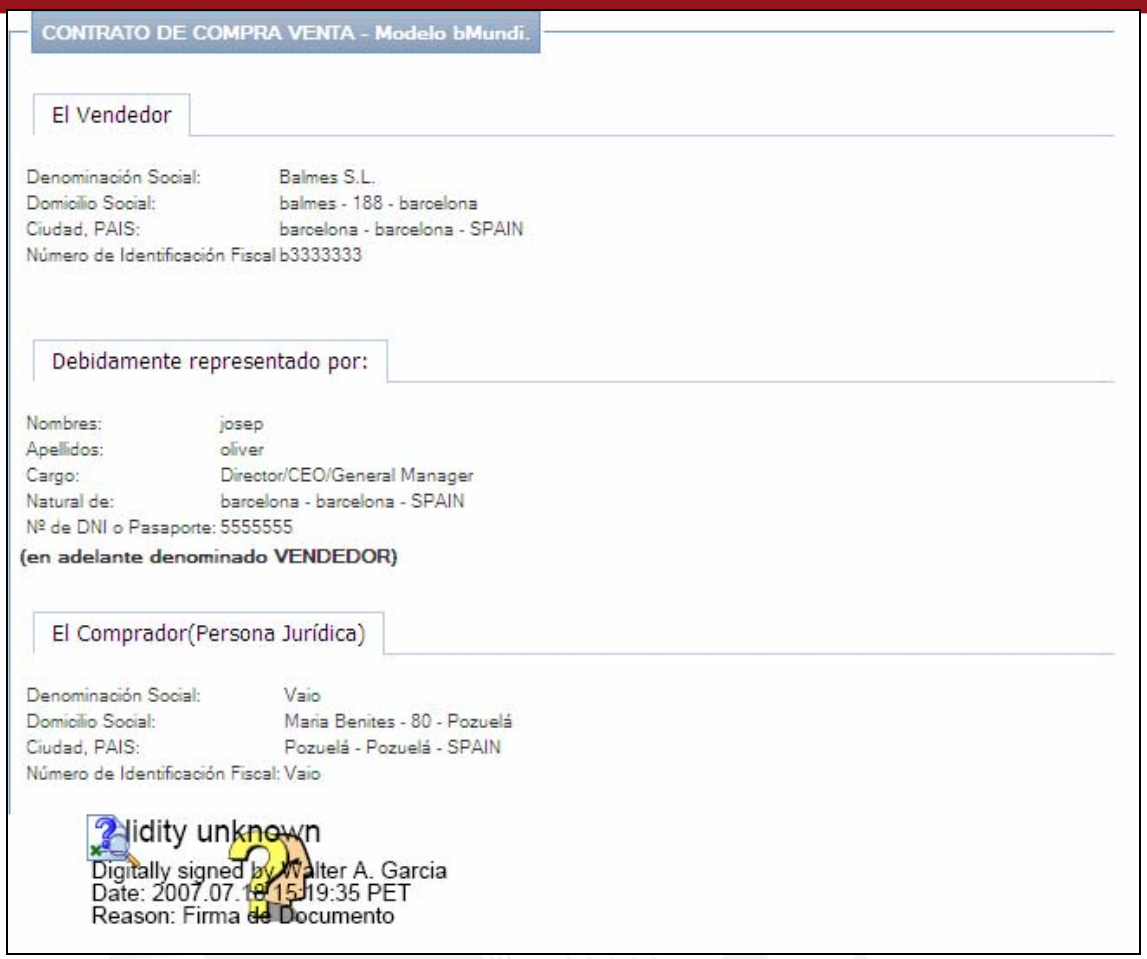


Ilustración 19: Firma de contrato por el vendedor

- ✓ Sigue el flujo principal (Ver Capítulo 4, Sección 4)
- ✓ Luego de haber sido firmado por el vendedor ya se podrá visualizar el contrato firmado (solo por el vendedor) (Ver Ilustración 20):


CONTRATO DE COMPRA VENTA - Modelo bMundi.

El Vendedor

Denominación Social: Balmes S.L.
Domicilio Social: balmes - 188 - barcelona
Ciudad, PAIS: barcelona - barcelona - SPAIN
Número de Identificación Fiscal b3333333

Debidamente representado por:

Nombres: josep
Apellidos: oliver
Cargo: Director/CEO/General Manager
Natural de: barcelona - barcelona - SPAIN
Nº de DNI o Pasaporte: 5555555
(en adelante denominado VENDEDOR)

El Comprador(Persona Jurídica)

Denominación Social: Vaio
Domicilio Social: Maria Benites - 80 - Pozuelá
Ciudad, PAIS: Pozuelá - Pozuelá - SPAIN
Número de Identificación Fiscal: Vaio

lidity unknown
Digitally signed by Walter A. Garcia
Date: 2007.07.18 15:19:35 PET
Reason: Firma de Documento

Ilustración 20: Contrato firmado por el vendedor

- ✓ Una vez firmado el documento por parte del vendedor, el comprador ya tiene la opción de firmar el contrato (Ver Ilustración 21).

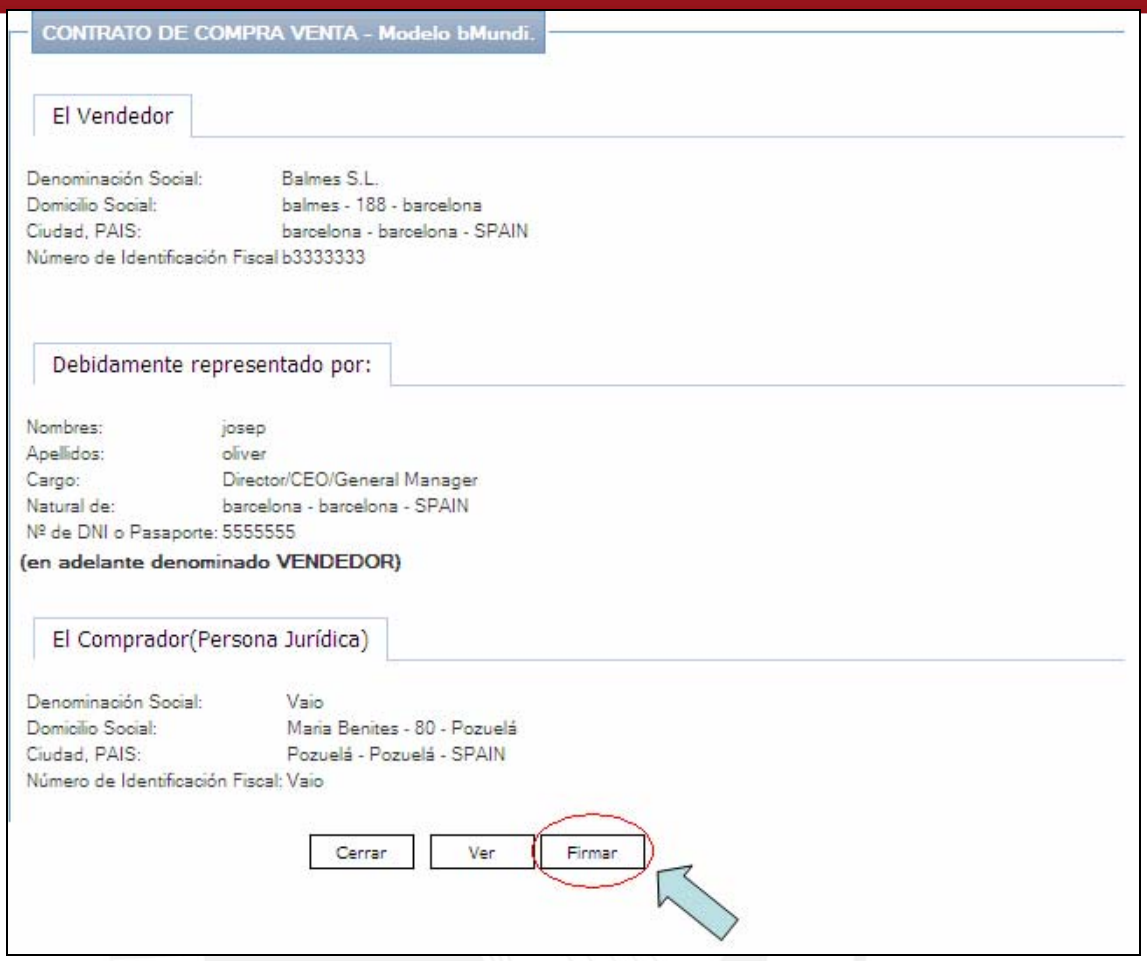
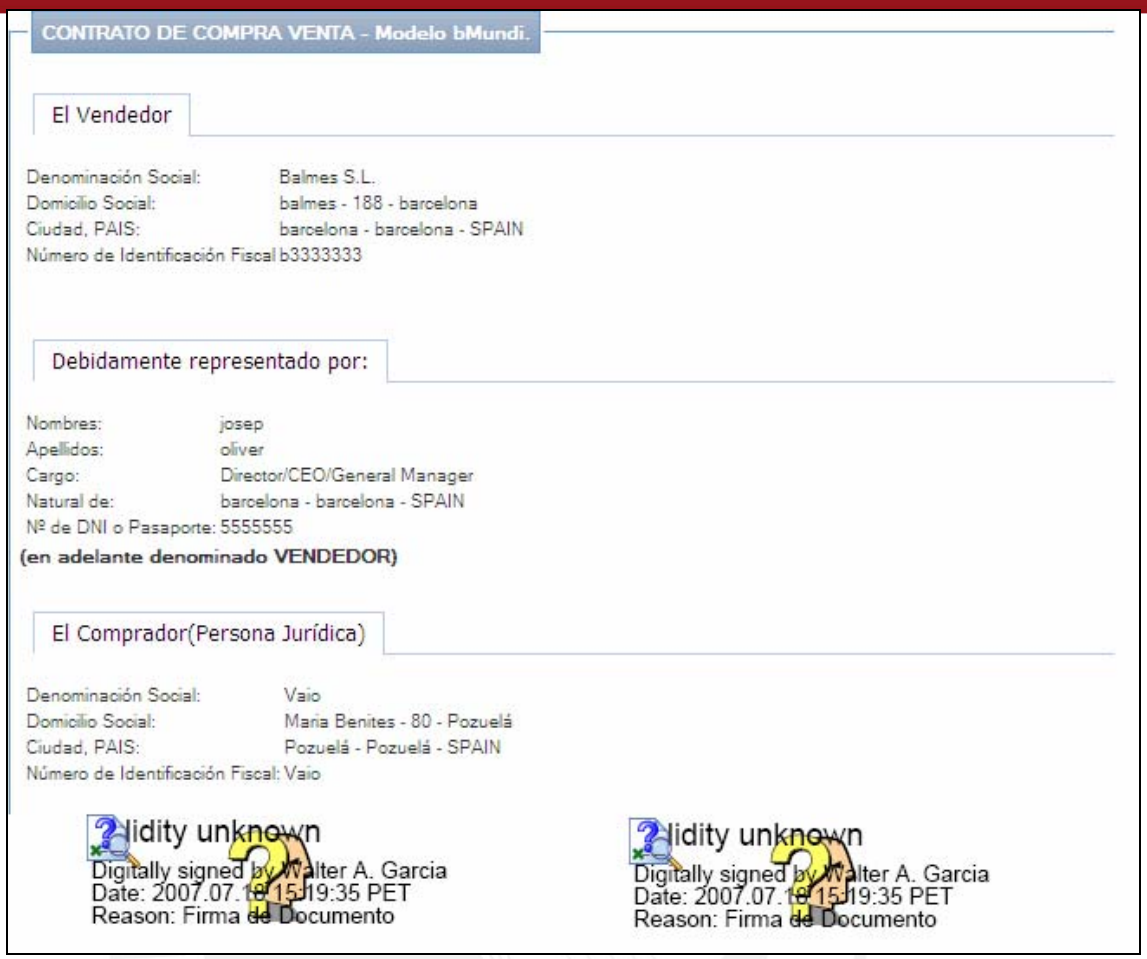


Ilustración 21: Firma de contrato por el comprador

✓ Sigue el flujo principal (Ver Capítulo 4, Sección 4)

✓ Luego de haber sido firmado por el comprador ya se podrá visualizar el contrato firmado por ambas partes (Ver Ilustración 22)


CONTRATO DE COMPRA VENTA - Modelo bMundi.

El Vendedor

Denominación Social: Balmes S.L.
Domicilio Social: balmes - 188 - barcelona
Ciudad, PAIS: barcelona - barcelona - SPAIN
Número de Identificación Fiscal b3333333

Debidamente representado por:

Nombres: josep
Apellidos: oliver
Cargo: Director/CEO/General Manager
Natural de: barcelona - barcelona - SPAIN
Nº de DNI o Pasaporte: 5555555
(en adelante denominado VENDEDOR)

El Comprador(Persona Jurídica)

Denominación Social: Vaio
Domicilio Social: Maria Benites - 80 - Pozuelá
Ciudad, PAIS: Pozuelá - Pozuelá - SPAIN
Número de Identificación Fiscal: Vaio

lidity unknown
Digitally signed by Walter A. Garcia
Date: 2007.07.18 15:19:35 PET
Reason: Firma de Documento

lidity unknown
Digitally signed by Walter A. Garcia
Date: 2007.07.18 15:19:35 PET
Reason: Firma de Documento

Ilustración 22: Contrato firmado por ambas partes

8. INFORMACIÓN DE CONTACTOS CON FIRMA DIGITAL

Es muy importante que en las negociaciones entre empresas a través de medios electrónicos haya evidencia de que la otra parte interesada en negociar sea una entidad seria, aquí el único medio fiable, seguro y legal para confiar en una empresa, muy posiblemente ubicada en otra parte del mundo, de forma simultánea y sin necesidad de testigos es la firma digital; bajo este esquema es imprescindible conocer la información sobre el certificado digital de la persona con la cual se pretende negociar y firmar contratos.

Para integrarse con la plataforma se ha desarrollado un módulo que permite conocer la información del certificado digital de los representantes de las empresas con las cuales se va a negociar, esta información fue obtenida al momento del registro o modificación de los datos del administrador de la empresa.

La información a mostrar en el tablón de Lista de Contactos con Firma Digital Homologada (Ver Ilustración 23) es:

- ✓ País de origen de la empresa.
- ✓ Nombre de la empresa con la que se negocia.
- ✓ Nombre del representante de la empresa que podrá firmar los documentos.
- ✓ Fecha de vencimiento de su certificado digital.

Ilustración 23: Información del tablón de Contactos con Firma Digital homologada

Por otro lado, también se ha incluido en el tablón de ofertas y productos de la plataforma, un campo que contiene la fecha de vencimiento del certificado digital del administrador de la empresa que hizo el anuncio (Ver Ilustración 24).

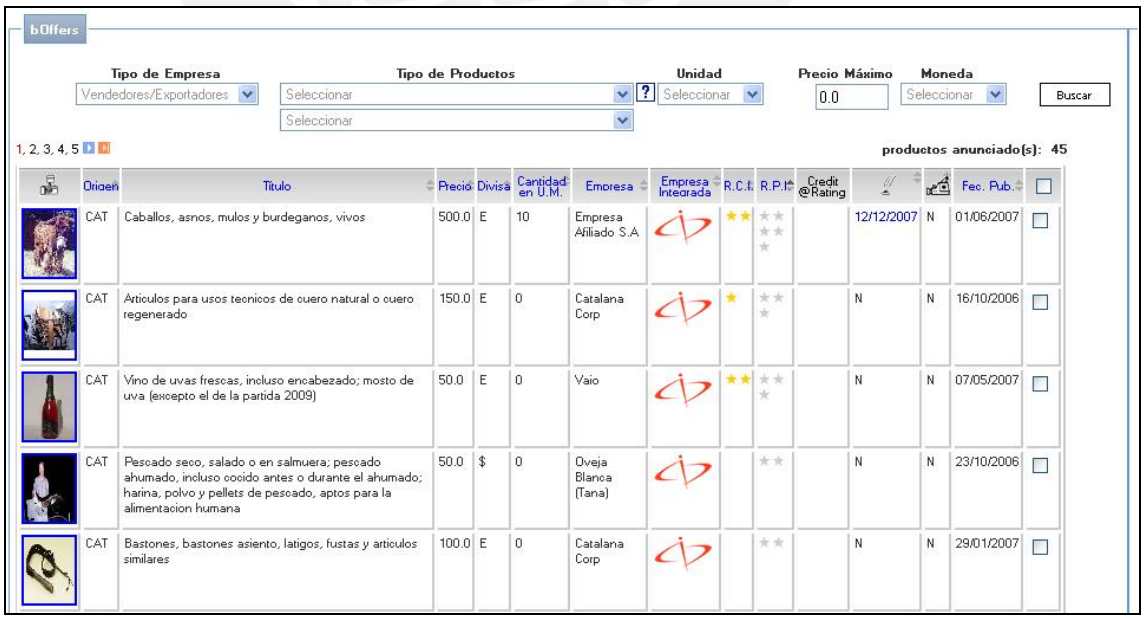


Ilustración 24: Tablón de ofertas y productos

Al hacer clic en la fecha de vencimiento del certificado se mostrará la información del certificado digital del representante (Ver Ilustración 25). La información contenida es la siguiente:

- ✓ Nombre del dueño del certificado.
- ✓ E-mail del dueño.
- ✓ Nombre de la empresa del dueño del certificado.
- ✓ Nombre de la entidad emisora del certificado.
- ✓ E-mail de la entidad emisora.
- ✓ Nombre de la empresa certificadora raíz.
- ✓ Vigencia del certificado.

Ilustración 25: Ficha de información del Certificado Digital

Con la información obtenida en este módulo tomaremos la decisión de negociar o no con la otra empresa.

CAPITULO 5: CONCLUSIONES Y RECOMENDACIONES

1. CONCLUSIONES

- ✓ La obtención de datos de la firma digital incrustada en los documentos PDF se ha aplicado sólo para documentos que se firman con la plataforma en el proceso de registro o modificación de datos en el mismo momento que esto ocurre, descartándose la posibilidad de obtener los datos de un documento ya firmado con cualquier otra herramienta, ya que no existe la certeza de que corresponda con la persona que realiza el registro o modificación de los datos.

- ✓ El esquema de firma de contratos incluye que se generen hasta tres copias por cada contrato: un contrato original sin firmas, un contrato con la firma del vendedor y otro con la firma de ambas partes; esto para tener evidencia de cada etapa del proceso para futuros reclamos legales que puedan suscitarse.

- ✓ El primero en firmar un contrato será el vendedor ya que legalmente es el que menos arriesga en un proceso de compra.

- ✓ El contrato PDF se visualizará sin ninguna firma o cuando contenga las firmas de ambas partes para no generar ningún tipo de desventaja legal para los involucrados comportándose la plataforma como un notario virtual.

- ✓ Tanto la incrustación de la firma digital como la del sello de tiempo dentro de los documentos PDF son tecnologías factibles.

- ✓ Todo proceso de firma digital y sellado de tiempo son tecnologías que tienen cien por ciento de valor legal en localidades que cuenten con autoridades de certificación autorizadas o que tengan leyes definidas que los avalen; así, los certificados emitidos fuera del Perú tienen valor legal dentro del territorio.

2. RECOMENDACIONES

- ✓ En el proceso de firma digital de documentos PDF se debe proteger de mejor manera la información personal del firmante, por lo cual, se recomienda el desarrollo de aplicaciones que eviten que se transfiera información privada, o lo menos posible, a través de la red manejándose esta en la máquina del cliente únicamente (p.e. applet)

- ✓ Las aplicaciones de firma digital pueden ser muy útiles si lo que se requiere es integridad de la información, autenticación del emisor, si se requiere evitar el no repudio, o si se requiere confidencialidad en los mensajes.

- ✓ El proceso de firma digital al ser una solución computacional permite reducir los cuellos de botella generados en procesos de trámites documentarios masivos, mediante la definición de políticas de uso.

BIBLIOGRAFÍA

1. [LEY01] Reglamento de la Ley de Firmas y Certificados Digitales Ley Nº 27269. Título 1, Capítulo 1, Artículo 4to. Definición de Entidad de Certificación.

2. [IND01] INDECOPI. Instituto Nacional de Defensa de la Competencia y de la Protección de la Propiedad Intelectual. Definición de Firma Digital.
   URL: http://www.indecopi.gob.pe/

3. [PRO01] Software PROFIRMA.
   URL: http://www128.ibm.com/developerworks/library/jstruts/?dwzone=java

4. [PLA01] Manuales Plan Avanza. “La factura electrónica”. España, 2008.

5. [CAM01] CAMERFIRMA. Proceso de sellado de tiempo.
   URL: http://www.camerfirma.com/
   URL (documento):
   http://www.camerfirma.com/mod_web/servicios/DescripcionSelladodeTiempoACCamerfirma.pdf

6. [LEG01] Régimen jurídico de la firma electrónica en el Perú. José Francisco Espinoza Céspedes.
   URL: http://www.ieid.org/congreso/ponencias/Espinoza%20Cespedes,%20JF.pdf

7. [LIB01] Librería ITEXT
   URL: http://www.lowagie.com/iText/

8. [LIB02] Librería BOUNCYCASTLE
   URL: http://www.bouncycastle.org/

9. [LIB03] Librería IAIK
   URL: http://jce.iaik.tugraz.at/

10. [LEY00] Ley de Firmas y Certificados Digitales. Congreso de la República.
    URL (prepublicación):
    http://www.idea.edu.pe/alegales/reglamenteodelaleydefirmas27269.pdf

URL (orden de publicación):
http://www.minjus.gob.pe/normatividad/Ley27269.pdf

ANEXOS

IMPLEMENTACIÓN DE FIRMA DIGITAL EN UNA PLATAFORMA DE COMERCIO ELECTRÓNICO

Anexo 1: Glosario de términos

1. **PDF**

   "Portable Document Format", formato de documento portátil creado por Adobe Systems.

2. **XML**

   "Extensible Markup Language", es una potente herramienta para definir estructuras de datos susceptibles de ser procesadas por una gran variedad de aplicaciones para realizar un eficiente intercambio electrónico de datos.

3. **TS**

   Siglas de TimeStamp (Sello de tiempo), es una secuencia de caracteres, que denotan la hora y fecha (o alguna de ellas) en la cual ocurrió determinado evento.

4. **PKCS**

   "Public Key Cryptography Standars", grupo de estándares de criptografía de clave pública publicado y concebido por los laboratorios de RSA.

5. **Hash**

   Un hash o resumen es el resultado de la aplicación de un función o algoritmo hash.

6. **RSA**

   Algoritmo asimétrico para cifrado de bloques. Este algoritmo es reversible, es decir, además de permitir cifrar con la clave pública y descifrar con la privada, permite cifrar con la clave privada y descifrar con la clave pública. Fue desarrollado por Rivest, Shamir y Adleman.

7. **DSA**

   DSA (Digital Signature Algorithm, en español Algoritmo de Firma digital) es un estándar del Gobierno Federal de los Estados Unidos de América o FIPS para firmas digitales.

8. **MD5**

   MD5 (Message-Digest Algorithm 5 o Algoritmo de Firma de Mensajes 5) es el algoritmo hash más usado, desarrollado por Ron Rivest. Procesa mensajes de una

longitud arbitraria en bloques de 512 bits generando un compendio de 128 bits. Debido a la capacidad de procesamiento actual esos 128 bits son insuficientes, además de que una serie de ataques criptoanalíticos han puesto de manifiesto algunas vulnerabilidades del algoritmo. Puede ser útil para comprobar la integridad de un fichero tras una descarga, por ejemplo, pero ya no es aceptable desde el punto de vista criptoanalítico.

9. SHA-1

SHA-1 (Secure Hash Algorithm 1 o Algorimo de Hash Seguro 1) es un algoritmo hash. El SHA-1 tomo como entrada un mensaje de longitud máxima 264 bits (más de dos mil millones de Gigabytes) y produce como salida un resumen de 160 bits. Este número es mayor que el que se utilizaba en el algoritmo SHA original, 128 bits. Ya existen nuevas versiones de SHA que trabajan con resúmenes de 224, 256, 384 e incluso 512 bits.

10. DES o DEA

Data Encryption Standard (DES) o Data Encryption Algorithm (DEA) es un algoritmo de cifrado, es decir, un método para cifrar información, escogido como FIPS en los Estados Unidos en 1976, y cuyo uso se ha propagado ampliamente por todo el mundo.

11. Función inyectiva

En matemáticas, una función $f: X \to Y$ es inyectiva o uno es a uno si cada valor en la imagen de f corresponde un único origen en el dominio.

Por ejemplo, la función de números reales $f : \mathbb{R} \to \mathbb{R}$, dada por $f(x) = x^2$ no es inyectiva, puesto que el valor 4 puede obtenerse como f(2) y f(− 2). Pero si el dominio se restringe a los números positivos, obteniendo así una nueva función $g : \mathbb{R}^+ \to \mathbb{R}^+$ entonces sí se obtiene una función inyectiva.

12. Criptoanálisis

Criptoanálisis (del griego kryptós, "escondido" y analýein, "desatar") es el estudio de los métodos para obtener el sentido de una información cifrada, sin acceso a la información secreta requerida para obtener este sentido normalmente. Típicamente, esto se traduce en conseguir la clave secreta.

13. PGP

Pretty Good Privacy o PGP (privacidad bastante buena) es un programa desarrollado por Phil Zimmermann y cuya finalidad es proteger la información distribuida a través de Internet mediante el uso de criptografía de clave pública, así como facilitar la autenticación de documentos gracias a firmas digitales.

14. GnuPG

GPG o GNU Privacy Guard es una herramienta para cifrado y firmas digitales, que viene a ser un reemplazo del PGP (Pretty Good Privacy) pero con la principal diferencia que es software libre licenciado bajo la GPL. GPG utiliza el estandar del IETF denominado OpenPGP.

Anexo 2: Código fuente para firma digital y sello de tiempo

A continuación se detallan las clases más importantes que se utilizaron para el presente trabajo de tesis. Se hará una descripción de los atributos y métodos de las siguientes clases:

- ✓ PdfPKCS7: Clase que contiene todo el proceso relativo al firmado y verificación de una firma PKCS7. Esta clase fue desarrollada por Paulo Soares y modificada por Aiken Sam.
- ✓ SignPdf: Clase que contiene los métodos necesarios para firmar un Pdf haciendo uso de un archivo PFX. Para la implementación de esta clase se hace uso de la librería iText.
- ✓ TSAClientBouncyCastle: Clase que contiene los métodos necesarios para incrustar el sello de tiempo a un PDF. Para la implementación de esta clase se hace uso de la librería BouncyCastle.
- ✓ PdfStamper: Clase que controla el contenido extra de un PDF como marcadores, enlaces, etc. Clase de la librería iText.
- ✓ PdfSignatureAppearance: Clase que se encarga de las opciones de criptografía y la apariencia de la firma en el PDF. Clase de la librería iText.
- ✓ PdfSignature: Clase que implementa el diccionario de la firma. Clase de la librería iText.
- ✓ PdfDictionary: Clase que maneja el diccionario (conjunto de atributos) en los PDF. Clase de la librería iText.

Class PdfPKCS7

java.lang.Object
└ **com.lowagie.text.pdf.PdfPKCS7**

public class **PdfPKCS7**

This class does all the processing related to signing and verifying a PKCS#7 signature. It's based in code found at org.bouncycastle

Author:
Paulo Soares (psoares@consiste.pt)
Modified by:
Aiken Sam, 2006-11-15: PKCS#7 timestamp feature added

URL de referencia: http://threebit.net/mail-archive/itext-questions/msg06121.html

Field Summary

```
private byte sigAttr[];
private byte digestAttr[];
private int version, signerversion;
private Set digestalgos;
private Collection certs, crls;
private X509Certificate signCert;
private byte[] digest;
private MessageDigest messageDigest;
private String digestAlgorithm, digestEncryptionAlgorithm;
private Signature sig;
private transient PrivateKey privKey;
private byte RSAdata[];
private boolean verified;
private boolean verifyResult;
private byte externalDigest[];
private byte externalRSAdata[];
private static final String ID_PKCS7_DATA = "1.2.840.113549.1.7.1";
private static final String ID_PKCS7_SIGNED_DATA = "1.2.840.113549.1.7.2";
private static final String ID_MD5 = "1.2.840.113549.2.5";
private static final String ID_MD2 = "1.2.840.113549.2.2";
private static final String ID_SHA1 = "1.3.14.3.2.26";
private static final String ID_RSA = "1.2.840.113549.1.1.1";
private static final String ID_DSA = "1.2.840.10040.4.1";
private static final String ID_CONTENT_TYPE = "1.2.840.113549.1.9.3";
private static final String ID_MESSAGE_DIGEST = "1.2.840.113549.1.9.4";
private static final String ID_SIGNING_TIME = "1.2.840.113549.1.9.5";
```

```
private static final String ID_MD2RSA = "1.2.840.113549.1.1.2";
private static final String ID_MD5RSA = "1.2.840.113549.1.1.4";
private static final String ID_SHA1RSA = "1.2.840.113549.1.1.5";
// Holds value of property reason.
private String reason;
// Holds value of property location.
private String location;
//Holds value of property signDate.
private Calendar signDate;
// Holds value of property signName.
private String signName;
// Holds value of properties for timestamp server.
private String usernameTimestamp = null;
private String passwordTimestamp = null;
private String serverTimestamp = null;
```

Constructor Summary
PdfPKCS7(byte[] contentsKey, byte[] certsKey, String provider) throws SecurityException, InvalidKeyException, CertificateException, NoSuchProviderException, NoSuchAlgorithmException, IOException Creates new PdfPKCS7. Verifies a signature using the sub-filter adbe.x509.rsa_sha1
PdfPKCS7(byte[] contentsKey, String provider) throws SecurityException, CRLException, InvalidKeyException, CertificateException, NoSuchProviderException, NoSuchAlgorithmException, StreamParsingException Creates new PdfPKCS7. Verifies a signature using the sub-filter adbe.pkcs7.detached or adbe.pkcs7.sha1.
PdfPKCS7(PrivateKey privKey, Certificate[] certChain, CRL[] crlList, String hashAlgorithm, String provider, boolean hasRSAdata) throws SecurityException, InvalidKeyException, NoSuchProviderException, NoSuchAlgorithmException Creates new PdfPKCS7. Generates a signature.

Method Summary	
void	**update**(byte[] buf, int off, int len) throws SignatureException Update the digest with the specified bytes. This method is used both for signing and verifying.
Certificate[]	**getCertificates**() Get the X.509 certificates associated with this PKCS#7 object (at) return the X.509 certificates associated with this PKCS#7 object.
Collection	**getCRLs**() Get the X.509 certificate revocation lists associated with this PKCS#7 object.
X509Certificate	**getSigningCertificate**() Get the X.509 certificate actually used to sign the digest.
int	**getVersion**()
int	**getSigningInfoVersion**()

static KeyStore	**loadCacertsKeyStore**() Loads the default root certificates at <java.home>/lib/security/cacerts with the default provider
byte[]	**getEncodedPKCS7**() Gets the bytes for the PKCS7SignedData object.
String	**getReason**() Getter for property reason.
void	**setReason**(String reason) Setter for property reason.
String	**getLocation**()
void	**setLocation**(String location)
Calendar	**getSignDate**() Getter for property signDate.
void	**setSignDate**(Calendar signDate) Setter for property signDate.
String	**getSignName**()
void	**setSignName**(String signName)
String	**getUsernameTimestamp**()
void	**setUsernameTimestamp**(String usernameTimestamp)
String	**getPasswordTimestamp**()
void	**setPasswordTimestamp**(String passwordTimestamp)
String	**getServerTimestamp**()
Void	**setServerTimestamp**(String serverTimestamp)
boolean	**Verify**() Verify the digest.
String	**getDigestAlgorithm**() Get the algorithm used to calculate the message digest.
String	**getHashAlgorithm**() Returns the algorithm.
static KeyStore	**loadCacertsKeyStore**(String provider) Loads the default root certificates at <java.home>/lib/security/cacerts.
static String	**verifyCertificate**(X509Certificate cert, Collection crls, Calendar calendar) Verifies a single certificate.
static Object[]	**verifyCertificates**(Certificate certs[], KeyStore keystore, Collection crls, Calendar calendar) Verifies a certificate chain against a KeyStore.
static DERObject	**getIssuer**(byte[] enc) Get the "issuer" from the TBSCertificate bytes that are passed in.

static DERObject	**getSubject**(byte[] enc) Get the "issuer" from the TBSCertificate bytes that are passed in.
static X509Name	**getIssuerFields**(X509Certificate cert) Get the issuer fields from an X509 Certificate.
static X509Name	**getSubjectFields**(X509Certificate cert) Get the subject fields from an X509 Certificate.
byte[]	**getEncodedPKCS1**() Gets the bytes for the PKCS#1 object.
void	**setExternalDigest**(byte digest[], byte RSAdata[], String digestEncryptionAlgorithm) Sets the digest/signature to an external calculated value.
byte[]	**getEncodedPKCS7**(byte secondDigest[], Calendar signingTime) Gets the bytes for the PKCS7SignedData object. Optionally the authenticatedAttributes in the signerInfo can also be set. If either of the parameters is <CODE>null</CODE>, none will be used.
byte[]	**getAuthenticatedAttributeBytes**(byte secondDigest[], Calendar signingTime) When using authenticatedAttributes the authentication process is different. The document digest is generated and put inside the attribute. The signing is done over the DER encoded authenticatedAttributes. This method provides that encoding and the parameters must be exactly the same as in { (at) link #getEncodedPKCS7(byte[],Calendar)
byte[]	**getEncodedPKCS7**(byte secondDigest[], Calendar signingTime, TSAClient tsaClient) Gets the bytes for the PKCS7SignedData object. Optionally the authenticatedAttributes in the signerInfo can also be set, OR a time-stamp-authority client may be provided.
ASN1EncodableVector	**buildUnauthenticatedAttributes**(byte[] timeStampToken) Added by Aiken Sam, 2006-11-15, modifed by Martin Brunecky 07/12/2007 to start with the timeStampToken (signedData 1.2.840.113549.1.7.2). Token is the TSA response without response status, which is usually handled by the (vendor supplied) TSA request/response interface).

Class SignPDF

Esta clase se utilizó para firmar un PDF. Para la implementación de esta clase se hizo uso de la librería de iText.

```
package FirmaDigital;

import java.io.*;
import java.util.*;
import com.lowagie.text.*;
import com.lowagie.text.pdf.*;

/**
 * Using iText to digitally sign PDF document with a valid time-stamp
 * Dependecies:
 * SignerKeystore - interface providing signing certificate access
 * SignerKeystorePKCS12 - implemnation importing PKCS12 (.pfx) certificate
 */

public class SignerPdf {
    private SignerKeystore sks;
    private TSAClient def;

    public SignerPdf(SignerKeystore sks, TSAClient tsc) throws Exception
    {
        this.sks = sks;
        this.def = tsc;
    }



    public void signPDF(SignerKeystore sks,PdfReader reader, FileOutputStream fout, int
estadoDoc) throws Exception {
        // Prepare for PDF file handling (copy, stamp)

        /* Para aceptar una sola firma:
         * estadoDoc= 0 - documento simple
         * PdfStamper stp = PdfStamper.createSignature(reader, fout, '\0');
         * Para aceptar multiples firmas:
         * estadoDoc= 1 - vendedor
         * estadoDoc= 2 - comprador
         * Applies a digital signature to a document, possibly as a new revision, making possible
multiple signatures.
         *
         * PdfStamper stp = PdfStamper.createSignature(reader, fout, '\0', new File("/temp"), true);
         * PdfStamper stp = PdfStamper.createSignature(reader, fout, '\0', null, true);
         * PdfStamper stp = PdfStamper.createSignature(reader, fout, '\0', new File("/temp"));
         * */
        PdfStamper stp;
        if (estadoDoc!=0){
            stp = PdfStamper.createSignature(reader, fout, '\0', null, true);
            signPDF(sks, stp, estadoDoc, reader, fout, def);
        }else{
            stp = PdfStamper.createSignature(reader, fout, '\0');
            signPDF(sks, stp, estadoDoc, reader, fout, def);
        }

    }
```

```
    public void signPDF(SignerKeystore sks,PdfStamper stp, int estadoDoc, PdfReader reader,
FileOutputStream fout, TSAClient tsc) {
        try {
            PdfSignatureAppearance sap = stp.getSignatureAppearance();
            setAppearance(sks,sap,estadoDoc);

            // Configure PDF signature dictionary (PdfName.ADOBE_PPKLITE works too)
            PdfSignature dic = new
PdfSignature(PdfName.ADOBE_PPKMS,PdfName.ADBE_PKCS7_SHA1);

            // dic.setName (null); // optional - PdfPKCS7 uses the certificate CN
            dic.setReason(sap.getReason());
            dic.setLocation(sap.getLocation());
            dic.setContact(sap.getContact());
            dic.setDate(new PdfDate(sap.getSignDate())); // time-stamp will over-rule this
            sap.setCryptoDictionary(dic);

            // Estimate signature size, creating a 'fake' one using fake data
            // (SHA1 length does not depend upon the data length)
            byte[] estSignature = genPKCS7Signature(new
ByteArrayInputStream("fake".getBytes()), null);
            int contentEst = estSignature.length + ((tsc == null) ? 0 :tsc.getTokenSizeEstimate());

            // Preallocate excluded byte-range for the signature content (hex encoded)
            HashMap exc = new HashMap();
            exc.put(PdfName.CONTENTS, new Integer(contentEst * 2 + 2));
            sap.preClose(exc);

            // Get the true data signature, including a true time stamp token
            byte[] encodedSig = genPKCS7Signature(sap.getRangeStream(),tsc);
            if (contentEst + 2 < encodedSig.length) {
                throw new Exception("Timestamp size estimate " + contentEst +
                                    " is too low for actual " +
                                    encodedSig.length);
            }

            // Copy signature into a zero-filled array, padding it up to estimate

            byte[] paddedSig = new byte[contentEst];
            System.arraycopy(encodedSig, 0, paddedSig, 0, encodedSig.length);

            // Finally, load zero-padded signature into the signature field /Content
            PdfDictionary dic2 = new PdfDictionary();
            dic2.put(PdfName.CONTENTS, new PdfString(paddedSig).setHexWriting(true));
            sap.close(dic2); // closes sap.close()

            System.out.println("Fin de firmado");

        } catch (Throwable t) {
            System.out.println("Signing failed" + t);
            t.printStackTrace();
        }

    }

    /**
     * Setup signature appearance. Override to define specifics.
     * @param sap PdfSignatureAppearance
     */
```

```
    protected void setAppearance(SignerKeystore sks,PdfSignatureAppearance sap,int
estadoDoc) {

      sap.setCrypto(null, sks.getChain(), null, PdfSignatureAppearance.WINCER_SIGNED);

       //sap.setLocation("Peru");

       if (estadoDoc==2){
          // El documento ya fue firmado por el vendedor - firma el comprador
          sap.setReason("bMundi - Firma de contrato");
          sap.setVisibleSignature(new Rectangle(400, 15, 550, 65), 1, null);
       }else{
          // El documento no esta firmado - firma el vendedor o se firma un documento simple
          if (estadoDoc==0){
                  sap.setReason("bMundi - Firma de documento");
          }else{
                  sap.setReason("bMundi - Firma de contrato");
          }
          sap.setVisibleSignature(new Rectangle(50, 15, 200, 65), 1, null);
       }

       // empty makes field invisible
    }

    /**
     * Generate the PKCS7 encoded signature
     * @param data InputStream - data to digest
     * @param doTimestamp boolean - true to include time-stamp
     * @return byte[]
     * @throws Exception
     */
    protected byte[] genPKCS7Signature(InputStream data, TSAClient tsc)throws  Exception {
       // assume sub-filter is adobe.pkcs7.sha1 entonces hasRSAdata=true
       PdfPKCS7 sgn = new PdfPKCS7(sks.getPrivateKey(), sks.getChain(),null,"SHA1",
sks.getProvider().getName(),true);
       byte[] buff = new byte[2048];
       int len = 0;
       while ((len = data.read(buff)) > 0) {
          sgn.update(buff, 0, len);
       }
       return sgn.getEncodedPKCS7(null, null, tsc);
    }
}
```

Class TSAClientBouncyCastle

Esta clase se utilizó para poner el sello de tiempo a un PDF. Para la implementación de esta clase se hizo uso de la librería BouncyCastle.

Autor:
Paulo Soares (psoares@consiste.pt)
Modificado por:
Aiken Sam, PKCS#7 timestamp feature added.

URL de referencia: http://threebit.net/mail-archive/itext-questions/msg06121.html

```
package FirmaDigital;
import java.io.*;
import java.math.*;
import java.net.*;
import org.bouncycastle.asn1.cmp.*;
import org.bouncycastle.asn1.x509.*;
import org.bouncycastle.tsp.*;

/**
 * Time Stamp Authority Client interface implementation using Bouncy Castle
 * org.bouncycastle.tsp package.
 * <p>
 * Created by Aiken Sam, 2006-11-15, refactored by Martin Brunecky, 07/15/2007
 * for ease of subclassing.
 * </p>
 */
public class TSAClientBouncyCastle implements TSAClient {
    protected String tsaURL;
    protected String tsaUsername;
    protected String tsaPassword;
    protected int    tokSzEstimate;

    public TSAClientBouncyCastle(String url) {
        this(url, null, null, 4096);
    }

    public TSAClientBouncyCastle(String url, String username, String password) {
        this(url, username, password, 4096);
    }

    /**
     * Constructor.
     * Note the token size estimate is updated by each call, as the token
     * size is not likely to change (as long as we call the same TSA using
     * the same imprint length).
     * @param url String - Time Stamp Authority URL (i.e. "http://tsatest1.digistamp.com/TSA")
     * @param username String - user(account) name
     * @param password String - password
     * @param tokSzEstimate int - estimated size of received time stamp token (DER encoded)
     */
    public TSAClientBouncyCastle(String url, String username, String password, int
tokSzEstimate) {
        this.tsaURL       = url;
        this.tsaUsername  = username;
```

```
        this.tsaPassword  = password;
        this.tokSzEstimate = tokSzEstimate;
    }

    /**
     * Get the token size estimate.
     * Returned value reflects the result of the last succesfull call, padded
     * @return int
     */
    public int getTokenSizeEstimate() {
        return tokSzEstimate;
    }

    public byte[] getTimeStampToken(PdfPKCS7 caller, byte[] imprint) throws Exception {
        return getTimeStampToken(imprint);
    }

    /**
     * Get timestamp token - Bouncy Castle request encoding / decoding layer
     */
    protected byte[] getTimeStampToken(byte[] imprint) throws Exception {
        byte[] respBytes = null;
        try {
            // Setup the time stamp request
            TimeStampRequestGenerator tsqGenerator = new TimeStampRequestGenerator();
            tsqGenerator.setCertReq(true);
            // tsqGenerator.setReqPolicy("1.3.6.1.4.1.601.10.3.1");
            BigInteger nonce = BigInteger.valueOf(System.currentTimeMillis());
            TimeStampRequest request =
tsqGenerator.generate(X509ObjectIdentifiers.id_SHA1.getId() , imprint, nonce);
            byte[] requestBytes = request.getEncoded();

            // Call the communications layer
            respBytes = getTSAResponse(requestBytes);

            // Handle the TSA response
            TimeStampResponse response = new TimeStampResponse(respBytes);

            // validate communication level attributes (RFC 3161 PKIStatus)
            response.validate(request);
            PKIFailureInfo failure = response.getFailInfo();
            int value = (failure == null) ? 0 : failure.intValue();
            if (value != 0) {
                // @todo: Translate value of 15 error codes defined by PKIFailureInfo to string
                throw new Exception("Invalid TSA '" + tsaURL + "' response, code " + value);
            }
            // @todo: validate the time stap certificate chain (if we want
            //        assure we do not sign using an invalid timestamp).

            // extract just the time stamp token (removes communication status info)
            TimeStampToken  tsToken = response.getTimeStampToken();
            if (tsToken == null) {
                throw new Exception("TSA '" + tsaURL + "' failed to return time stamp token");
            }
            TimeStampTokenInfo info = tsToken.getTimeStampInfo(); // to view details
            byte[] encoded = tsToken.getEncoded();
            long stop = System.currentTimeMillis();

            // Update our token size estimate for the next call (padded to be safe)
            this.tokSzEstimate = encoded.length + 32;
```

```
            return encoded;
        }
        catch (Exception e) {
            throw e;
        }
        catch (Throwable t) {
            throw new Exception("Failed to get TSA response from '" + tsaURL +"'", t);
        }
    }

    /**
     * Get timestamp token - communications layer
     * @return - byte[] - TSA response, raw bytes (RFC 3161 encoded)
     */
    protected byte[] getTSAResponse(byte[] requestBytes) throws Exception {
        // Setup the TSA connection
        URL url = new URL(tsaURL);
        URLConnection tsaConnection = (URLConnection) url.openConnection();

        tsaConnection.setDoInput(true);
        tsaConnection.setDoOutput(true);
        tsaConnection.setUseCaches(false);
        tsaConnection.setRequestProperty("Content-Type", "application/timestamp-query");
        // tsaConnection.setRequestProperty("Content-Transfer-Encoding", "base64");
        tsaConnection.setRequestProperty("Content-Transfer-Encoding", "binary");

        if ((tsaUsername != null) && !tsaUsername.equals("") ) {
            String userPassword = tsaUsername + ":" + tsaPassword;
            tsaConnection.setRequestProperty("Authorization", "Basic " +
                new String(new sun.misc.BASE64Encoder().encode(userPassword.getBytes())));
        };
        OutputStream out = tsaConnection.getOutputStream();
        out.write(requestBytes);
        out.close();

        // Get TSA response as a byte array
        InputStream inp = tsaConnection.getInputStream();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int bytesRead = 0;
        while ((bytesRead = inp.read(buffer, 0, buffer.length)) >= 0) {
            baos.write(buffer, 0, bytesRead);
        }
        byte[] respBytes = baos.toByteArray();

        String encoding = tsaConnection.getContentEncoding();
        if (encoding != null && encoding.equalsIgnoreCase("base64")) {
            sun.misc.BASE64Decoder dec = new sun.misc.BASE64Decoder();
            respBytes = dec.decodeBuffer(new String(respBytes));
        }
        return respBytes;
    }

}
```

A continuación veremos una descripción de las clases más relevantes de la librería iText que se usaron para el presente trabajo.

Class PdfStamper

java.lang.Object
└ **com.lowagie.text.pdf.PdfStamper**
All Implemented Interfaces:
PdfEncryptionSettings, PdfViewerPreferences

public class **PdfStamper**
extends Object
implements PdfViewerPreferences, PdfEncryptionSettings

Applies extra content to the pages of a PDF document. This extra content can be all the objects allowed in PdfContentByte including pages from other Pdfs. The original PDF will keep all the interactive elements including bookmarks, links and form fields.

It is also possible to change the field values and to flatten them. New fields can be added but not flattened.

Author:
Paulo Soares (psoares@consiste.pt)

Field Summary	
private boolean	**hasSignature**
private HashMap	**moreInfo**
private PdfSignatureAppearance	**sigApp**
protected PdfStamperImp	**stamper** The writer

Constructor Summary
PdfStamper(PdfReader reader, OutputStream os) Starts the process of adding extra content to an existing PDF document.
PdfStamper(PdfReader reader, OutputStream os, char pdfVersion) Starts the process of adding extra content to an existing PDF document.
PdfStamper(PdfReader reader, OutputStream os, char pdfVersion, boolean append) Starts the process of adding extra content to an existing PDF document, possibly as a new revision.

Method Summary	
void	**addAnnotation**(PdfAnnotation annot, int page) Adds an annotation of form field in a specific page.

void	**addComments**(FdfReader fdf) Adds the comments present in an FDF file.
void	**addFileAttachment**(String description, byte[] fileStore, String file, String fileDisplay) Adds a file attachment at the document level.
void	**addFileAttachment**(String description, PdfFileSpecification fs) Adds a file attachment at the document level.
void	**addJavaScript**(String js) Adds a JavaScript action at the document level.
void	**addViewerPreference**(PdfName key, PdfObject value) Adds a viewer preference
void	**close**() Closes the document.
static PdfStamper	**createSignature**(PdfReader reader, OutputStream os, char pdfVersion) Applies a digital signature to a document.
static PdfStamper	**createSignature**(PdfReader reader, OutputStream os, char pdfVersion, File tempFile) Applies a digital signature to a document.
static PdfStamper	**createSignature**(PdfReader reader, OutputStream os, char pdfVersion, File tempFile, boolean append) Applies a digital signature to a document, possibly as a new revision, making possible multiple signatures.
AcroFields	**getAcroFields**() Gets the AcroFields object that allows to get and set field values and to merge FDF forms.
PdfImportedPage	**getImportedPage**(PdfReader reader, int pageNumber) Gets a page from other PDF document.
HashMap	**getMoreInfo**() Gets the optional String map to add or change values in the info dictionary.
PdfContentByte	**getOverContent**(int pageNum) Gets a PdfContentByte to write over the page of the original document.
Map	**getPdfLayers**() Gets the PdfLayer objects in an existing document as a Map with the names/titles of the layers as keys.
PdfReader	**getReader**() Gets the underlying PdfReader.
PdfSignatureAppearance	**getSignatureAppearance**() Gets the signing instance.
PdfContentByte	**getUnderContent**(int pageNum) Gets a PdfContentByte to write under the page of the original document.

PdfWriter	**getWriter**() Gets the underlying PdfWriter.
void	**insertPage**(int pageNumber, Rectangle mediabox) Inserts a blank page.
boolean	**isFullCompression**() Gets the 1.5 compression status.
boolean	**isRotateContents**() Checks if the content is automatically adjusted to compensate the original page rotation.
void	**makePackage**(PdfCollection collection) Adds or replaces the Collection Dictionary in the Catalog.
void	**makePackage**(PdfName initialView) This is the most simple way to change a PDF into a portable collection.
boolean	**partialFormFlattening**(String name) Adds name to the list of fields that will be flattened on close, all the other fields will remain.
void	**replacePage**(PdfReader r, int pageImported, int pageReplaced) Replaces a page from this document with a page from other document.
void	**setDuration**(int seconds, int page) Sets the display duration for the page (for presentations)
void	**setEncryption**(boolean strength, String userPassword, String ownerPassword, int permissions) Sets the encryption options for this document.
void	**setEncryption**(byte[] userPassword, byte[] ownerPassword, int permissions, boolean strength128Bits) Sets the encryption options for this document.
void	**setEncryption**(byte[] userPassword, byte[] ownerPassword, int permissions, int encryptionType) Sets the encryption options for this document.
void	**setEncryption**(Certificate[] certs, int[] permissions, int encryptionType) Sets the certificate encryption options for this document.
void	**setEncryption**(int encryptionType, String userPassword, String ownerPassword, int permissions) Sets the encryption options for this document.
void	**setFormFlattening**(boolean flat) Determines if the fields are flattened on close.
void	**setFreeTextFlattening**(boolean flat) Determines if the FreeText annotations are flattened on close.

void	**setFullCompression**() Sets the document's compression to the new 1.5 mode with object streams and xref streams.
void	**setMoreInfo**(HashMap moreInfo) An optional String map to add or change values in the info dictionary.
void	**setOutlines**(List outlines) Sets the bookmarks.
void	**setPageAction**(PdfName actionType, PdfAction action, int page) Sets the open and close page additional action.
void	**setRotateContents**(boolean rotateContents) Flags the content to be automatically adjusted to compensate the original page rotation.
void	**setThumbnail**(Image image, int page) Sets the thumbnail image for a page.
void	**setTransition**(PdfTransition transition, int page) Sets the transition for the page
void	**setViewerPreferences**(int preferences) Sets the viewer preferences.
void	**setXmpMetadata**(byte[] xmp) Sets the XMP metadata.

Class PdfSignatureAppearance

java.lang.Object
└ **com.lowagie.text.pdf.PdfSignatureAppearance**

public class **PdfSignatureAppearance**
extends Object

This class takes care of the cryptographic options and appearances that form a signature.

Nested Class Summary	
private static class	**PdfSignatureAppearance.RangeStream**
static interface	**PdfSignatureAppearance.SignatureEvent** An interface to retrieve the signature dictionary for modification.

Field Summary	
private boolean	**acro6Layers** Holds value of property acro6Layers.
private PdfTemplate[]	**app**
private byte[]	**bout**
private int	**boutLen**
private Certificate[]	**certChain**
private int	**certificationLevel**
static int	**CERTIFIED_FORM_FILLING**
static int	**CERTIFIED_FORM_FILLING_AND_ANNOTATIONS**
static int	**CERTIFIED_NO_CHANGES_ALLOWED**
private String	**contact** Holds value of property contact.
private CRL[]	**crlList**
private PdfDictionary	**cryptoDictionary**
private String	**digestEncryptionAlgorithm**

private HashMap	**exclusionLocations**
private byte[]	**externalDigest**
private byte[]	**externalRSAdata**
private String	**fieldName**
private PdfName	**filter**
private PdfTemplate	**frm**
private Image	**image** Holds value of property image.
private float	**imageScale** Holds value of property imageScale.
private Font	**layer2Font** Holds value of property layer2Font.
private String	**layer2Text**
private String	**layer4Text** Holds value of property layer4Text.
private String	**location**
private static float	**MARGIN**
private boolean	**newField**
static int	**NOT_CERTIFIED**
private OutputStream	**originalout**
private int	**page**
private Rectangle	**pageRect**
private boolean	**preClosed**
private PrivateKey	**privKey**
private String	**provider**
static String	**questionMark**

	Commands to draw a yellow question mark in a stream content
private RandomAccessFile	**raf**
private int[]	**range**
private String	**reason**
private Rectangle	**rect**
private int	**render**
private int	**runDirection** Holds value of property runDirection.
static PdfName	**SELF_SIGNED** The self signed filter.
private PdfSignatureAppearance.SignatureEvent	**signatureEvent** Holds value of property signatureEvent.
private Image	**signatureGraphic**
static int	**SignatureRenderDescription** The rendering mode is just the description
static int	**SignatureRenderGraphicAndDescription** The rendering mode is an image and the description
static int	**SignatureRenderNameAndDescription** The rendering mode is the name of the signer and the description
private Calendar	**signDate**
private ByteBuffer	**sigout**
private PdfSigGenericPKCS	**sigStandard**
private PdfStamper	**stamper**
private File	**tempFile**
private static float	**TOP_SECTION**
static PdfName	**VERISIGN_SIGNED** The VeriSign filter.
static PdfName	**WINCER_SIGNED** The Windows Certificate Security.

private PdfStamperImp	**writer**

Constructor Summary
PdfSignatureAppearance(PdfStamperImp writer)

Method Summary	
private void	**addDocMDP**(PdfDictionary crypto)
void	**close**(PdfDictionary update) This is the last method to be called when using external signatures.
static float	**fitText**(Font font, String text, Rectangle rect, float maxFontSize, int runDirection) Fits the text to some rectangle adjusting the font size as needed.
PdfTemplate	**getAppearance**() Gets the main appearance layer.
Certificate[]	**getCertChain**() Gets the certificate chain.
int	**getCertificationLevel**() Gets the certified status of this document.
String	**getContact**() Gets the signing contact.
CRL[]	**getCrlList**() Gets the certificate revocation list.
PdfDictionary	**getCryptoDictionary**() Gets the user made signature dictionary.
String	**getFieldName**() Gets the field name.
PdfName	**getFilter**() Gets the filter used to sign the document.
Image	**getImage**() Gets the background image for the layer 2.
float	**getImageScale**() Gets the scaling to be applied to the background image.
PdfTemplate	**getLayer**(int layer) Gets a template layer to create a signature appearance.
Font	**getLayer2Font**() Gets the n2 and n4 layer font.
String	**getLayer2Text**() Gets the signature text identifying the signer if set by setLayer2Text().

String	**getLayer4Text**() Gets the text identifying the signature status if set by setLayer4Text().
String	**getLocation**() Gets the signing location.
String	**getNewSigName**() Gets a new signature fied name that doesn't clash with any existing name.
(package private) OutputStream	**getOriginalout**()
int	**getPage**() Gets the page number of the field.
Rectangle	**getPageRect**() Gets the rectangle that represent the position and dimension of the signature in the page.
PrivateKey	**getPrivKey**() Gets the private key.
String	**getProvider**() Returns the Cryptographic Service Provider that will sign the document.
InputStream	**getRangeStream**() Gets the document bytes that are hashable when using external signatures.
String	**getReason**() Gets the signing reason.
Rectangle	**getRect**() Gets the rectangle representing the signature dimensions.
int	**getRender**() Gets the rendering mode for this signature.
int	**getRunDirection**() Gets the run direction.
PdfSignatureAppearance.SignatureEvent	**getSignatureEvent**() Getter for property signatureEvent.
Image	**getSignatureGraphic**() Gets the Image object to render.
Calendar	**getSignDate**() Gets the signature date.
(package private) ByteBuffer	**getSigout**()
PdfSigGenericPKCS	**getSigStandard**() Gets the instance of the standard signature dictionary.
PdfStamper	**getStamper**() Gets the PdfStamper associated with this instance.

File	**getTempFile**() Gets the temporary file.
PdfTemplate	**getTopLayer**() Gets the template that aggregates all appearance layers.
boolean	**isAcro6Layers**() Gets the Acrobat 6.0 layer mode.
boolean	**isInvisible**() Gets the visibility status of the signature.
boolean	**isNewField**() Checks if a new field was created.
boolean	**isPreClosed**() Checks if the document is in the process of closing.
void	**preClose**() This is the first method to be called when using external signatures.
void	**preClose**(HashMap exclusionSizes) This is the first method to be called when using external signatures.
void	**setAcro6Layers**(boolean acro6Layers) Acrobat 6.0 and higher recommends that only layer n2 and n4 be present.
void	**setCertificationLevel**(int certificationLevel) Sets the document type to certified instead of simply signed.
void	**setContact**(String contact) Sets the signing contact.
void	**setCrypto**(PrivateKey privKey, Certificate[] certChain, CRL[] crlList, PdfName filter) Sets the cryptographic parameters.
void	**setCryptoDictionary**(PdfDictionary cryptoDictionary) Sets a user made signature dictionary.
void	**setExternalDigest**(byte[] digest, byte[] RSAdata, String digestEncryptionAlgorithm) Sets the digest/signature to an external calculated value.
void	**setImage**(Image image) Sets the background image for the layer 2.
void	**setImageScale**(float imageScale) Sets the scaling to be applied to the background image.
void	**setLayer2Font**(Font layer2Font) Sets the n2 and n4 layer font.
void	**setLayer2Text**(String text) Sets the signature text identifying the signer.

void	**setLayer4Text**(String text) Sets the text identifying the signature status.
void	**setLocation**(String location) Sets the signing location.
(package private) void	**setOriginalout**(OutputStream originalout)
void	**setProvider**(String provider) Sets the Cryptographic Service Provider that will sign the document.
void	**setReason**(String reason) Sets the signing reason.
void	**setRender**(int render) Sets the rendering mode for this signature.
void	**setRunDirection**(int runDirection) Sets the run direction in the n2 and n4 layer.
void	**setSignatureEvent**(PdfSignatureAppearance.SignatureEvent signatureEvent) Sets the signature event to allow modification of the signature dictionary.
void	**setSignatureGraphic**(Image signatureGraphic) Sets the Image object to render when Render is set to SignatureRenderGraphicAndDescription
void	**setSignDate**(Calendar signDate) Sets the signature date.
(package private) void	**setSigout**(ByteBuffer sigout)
(package private) void	**setStamper**(PdfStamper stamper)
(package private) void	**setTempFile**(File tempFile)
void	**setVisibleSignature**(Rectangle pageRect, int page, String fieldName) Sets the signature to be visible.
void	**setVisibleSignature**(String fieldName) Sets the signature to be visible.

Class PdfSignature

java.lang.Object
  └ com.lowagie.text.pdf.PdfObject
    └ com.lowagie.text.pdf.PdfDictionary
      └ **com.lowagie.text.pdf.PdfSignature**

Direct Known Subclasses:
PdfSigGenericPKCS

public class **PdfSignature**
extends PdfDictionary

Implements the signature dictionary.

Author:
Paulo Soares (psoares@consiste.pt)

Field Summary

Fields inherited from class com.lowagie.text.pdf.PdfDictionary
CATALOG, FONT, hashMap, OUTLINES, PAGE, PAGES

Constructor Summary
PdfSignature(PdfName filter, PdfName subFilter) Creates new PdfSignature

Method Summary	
void	**setByteRange**(int[] range)
void	**setCert**(byte[] cert)
void	**setContact**(String name)
void	**setContents**(byte[] contents)
void	**setDate**(PdfDate date)
void	**setLocation**(String name)
void	**setName**(String name)
void	**setReason**(String name)

Class PdfDictionary

java.lang.Object
  └ com.lowagie.text.pdf.PdfObject
    └ **com.lowagie.text.pdf.PdfDictionary**

Direct Known Subclasses:
PdfAcroForm, PdfAction, PdfAnnotation, PdfBorderDictionary, PdfCollection, PdfCollectionField, PdfCollectionItem, PdfCollectionSchema, PdfCollectionSort, PdfDocument.PdfCatalog, PdfDocument.PdfInfo, PdfFileSpecification, PdfGState, PdfLayer, PdfLayerMembership, PdfMediaClipData, PdfOCProperties, PdfOutline, PdfPage, PdfRendition, PdfResources, PdfShadingPattern, PdfSignature, PdfStream, PdfStructureElement, PdfStructureTreeRoot, PdfTargetDictionary, PdfTransparencyGroup, PdfWriter.PdfTrailer, PRAcroForm

public class **PdfDictionary**
extends PdfObject

PdfDictionary is the Pdf dictionary object.

A dictionary is an associative table containing pairs of objects. The first element of each pair is called the *key* and the second element is called the *value*. Unlike dictionaries in the PostScript language, a key must be a PdfName. A value can be any kind of PdfObject, including a dictionary. A dictionary is generally used to collect and tie together the attributes of a complex object, with each key-value pair specifying the name and value of an attribute.
A dictionary is represented by two left angle brackets (<<), followed by a sequence of key-value pairs, followed by two right angle brackets (>>).
This object is described in the 'Portable Document Format Reference Manual version 1.7' section 3.2.6 (page 59-60).

See Also:
PdfObject, PdfName, BadPdfFormatException

Field Summary	
static PdfName	**CATALOG** This is a possible type of dictionary
private PdfName	**dictionaryType** This is the type of this dictionary
static PdfName	**FONT** This is a possible type of dictionary
protected HashMap	**hashMap** This is the hashmap that contains all the values and keys of the dictionary
static PdfName	**OUTLINES** This is a possible type of dictionary
static PdfName	**PAGE** This is a possible type of dictionary

static PdfName	**PAGES** This is a possible type of dictionary

Constructor Summary
PdfDictionary() Constructs an empty PdfDictionary-object.
PdfDictionary(PdfName type) Constructs a PdfDictionary-object of a certain type.

Method Summary	
boolean	**contains**(PdfName key)
PdfObject	**get**(PdfName key) Gets a PdfObject with a certain key from the PdfDictionary.
PdfArray	**getAsArray**(PdfName key)
PdfBoolean	**getAsBoolean**(PdfName key)
PdfDictionary	**getAsDict**(PdfName key) All the getAs functions will return either null, or the specified object type This function will automatically look up indirect references.
PdfIndirectReference	**getAsIndirectObject**(PdfName key)
PdfName	**getAsName**(PdfName key)
PdfNumber	**getAsNumber**(PdfName key)
PdfStream	**getAsStream**(PdfName key)
PdfString	**getAsString**(PdfName key)
PdfObject	**getDirectObject**(PdfName key) This function behaves the same as 'get', but will never return an indirect reference, it will always look such references up and return the actual object.
Set	**getKeys**()
boolean	**isCatalog**() Checks if a Dictionary is of the type CATALOG.
boolean	**isFont**() Checks if a Dictionary is of the type FONT.
boolean	**isOutlineTree**() Checks if a Dictionary is of the type OUTLINES.

boolean	**isPage**() Checks if a Dictionary is of the type PAGE.
boolean	**isPages**() Checks if a Dictionary is of the type PAGES.
void	**merge**(PdfDictionary other)
void	**mergeDifferent**(PdfDictionary other)
void	**put**(PdfName key, PdfObject value) Adds a PdfObject and its key to the PdfDictionary.
void	**putAll**(PdfDictionary dic)
void	**putEx**(PdfName key, PdfObject value) Adds a PdfObject and its key to the PdfDictionary.
void	**remove**(PdfName key) Removes a PdfObject and its key from the PdfDictionary.
int	**size**()
void	**toPdf**(PdfWriter writer, OutputStream os) Returns the PDF representation of this PdfDictionary.
String	**toString**() Returns the String-representation of this PdfObject.

Anexo 3: Ley de firmas y certificados digitales

Orden de publicación de la Ley de firmas y certificados digitales.
Fuente: http://www.derechomatica.com/derechomatica/pdf/peruley.pdf

PERU
Ley nº 27.269 Ley de firmas y certificados digitales.
LEY Nº 27269
EL PRESIDENTE DE LA REPUBLICA
POR CUANTO:
El Congreso de la República ha dado la Ley siguiente:
EL CONGRESO DE LA REPÚBLICA;
Ha dado la Ley siguiente
LEY DE FIRMAS Y CERTIFICADOS DIGITALES

Artículo I. Objeto de la ley
La presente ley tiene por objeto regular la utilización de la firma electrónica otorgándole la misma validez y eficacia jurídica que el uso de una firma manuscrita u otra análoga que conlleve manifestación de voluntad.
Entiéndase por firma electrónica a cualquier símbolo basado en medios electrónicos utilizado o adoptado por una parte con la intención precisa de vincularse o autenticar un documento cumpliendo todas o algunas de las funciones características de la firma manuscrita.

Artículo 2. Ambito de aplicación
La presente ley se aplica a aquellas firmas electrónicas que, puestas sobre un mensaje de datos o añadidas o asociadas lógicamente a los mismos, puedan vincular e identificar al firmante, así como garantizar la autenticación e integridad de los documentos electrónicos.

Artículo 3. Firma digital
La firma digitales aquella firma electrónica que utiliza una técnica de criptografía asimétrica, basada en el uso de un par de claves único; asociadas una clave privada y una clave pública relacionadas matemáticamente entre sí, de tal forma que las personas que conocen la clave pública no puedan derivar de ella la clave privada.

Artículo 4. Titular de la firma digital
El titular de la firma digitales la persona a la que se le atribuye de manera exclusiva un certificado digital que contiene una firma digital, identificándolo objetivamente en relación con el mensaje de datos.

Artículo 5. Obligaciones del titular de la firma digital
El titular de la firma digital tiene la obligación de brindar a las entidades de certificación ya los terceras con quienes se relacione a través de la utilización de la firma digital, declaraciones o manifestaciones materiales exactas y completas.

Artículo 6. Certificado digital
El certificado digital es el documento electrónico generado y firmado digitalmente por una entidad de certificación, la cual vincula una parte claves con una persona determinada confirmando su identidad.

Artículo 7. Contenido del certificado digital
Los certificados digitales emitidos por las entidades de certificación deben contener al menos:

1. Datos que identifiquen indubitablemente al suscriptor.
2. Datos que identifiquen a la Entidad de Certificación.
3. La clave pública.
4. La metodología para verificar la firma digital del suscriptor impuesta a un mensaje de datos.
5. Número de serie del certificado.
6. Vigencia del certificado.
7. Firma digital de la Entidad de Certificación

Artículo 8.- Confidencialidad de la información
La entidad de registro recabará los datos personales del solicitante de la firma digital directamente de éste y para los finas señalados en la presente ley.
Asimismo la información relativa a las claves privadas y datos que no sean materia de certificación se mantiene bajo la reserva correspondiente. Sólo puede ser levantada por orden judicial o pedido expreso del suscriptor de la firma digital.

Artículo 9. Cancelación del certificado digital
La cancelación del certificado digital puede darse:
1. A solicitud del titular de la firma digital,
2. Por revocatoria de la entidad certificante o
3. Por expiración del plazo de vigencia
4. Por cese de operaciones de la Entidad de Certificación.

Artículo 10. Revocación del certificado digital
La Entidad de Certificación revocará el certificado digital en los siguientes casos:
1. Se determine que la información contenida en el certificado digital sea inexacta o haya sido modificada.
2. Por muerte del titular de la firma digital.
3. Por incumplimiento derivado de la relación contractual con la Entidad de certificación.

Artículo 11. Reconocimiento del certificados emitidos por entidades extranjeras
Los Certificadas de Firmas Digitales emitidos por entidades extranjeras tendrán la misma validez y eficacia jurídica reconocida en la presente ley, siempre y cuando tales certificadas sean reconocidos por una entidad de certificación nacional que garantice, en la misma forma que lo hace con sus propias certificados, el cumplimiento dé los requisitos, del procedimiento, así como la validez y la vigencia del certificado.

Artículo 12. Entidad de Certificación
La Entidad de Certificación cumple con la función de emitir o cancelar certificados digitales, así como brindar otros servicios inherentes al propio certificado o aquellos que brinden seguridad al sistema de certificados en particular o del comercio electrónico en general.

Artículo 13. Entidad de Registro o Verificación
La Entidad de Registro o Verificación cumple con la función de levantamiento de datos y comprobación de la información de un solicitante de certificado digital; identificación y autenticación del suscriptor de firma digital; aceptación y autorización de solicitudes de emisión de certificados digitales; aceptación y autorización de las solicitudes de cancelación de certificados digitales.

Artículo 14. Depósito de los Certificados Digitales
Cada Entidad de Certificación debe contar con un Registro disponible en forma permanente, que servirá para constatar la clave pública de determinado certificado y no podrá ser usado para fines distintos a los estipulados en la presente ley.

El Registro contará con una sección referida a los certificados digitales que hayan sido emitidos y figurarán las circunstancias que afecten la cancelación o vigencia de los mismos, debiendo constar la fecha y hora de inicio y fecha y hora de finalización.
A dicho Registro podrá accederse por medios telemáticos y su contenido estará a disposición de las personas que lo soliciten.

Artículo 15. Inscripción de Entidades de Certificación y de Registro o Verificación
El Poder Ejecutivo, por Decreto Supremo, determinará la autoridad administrative competente y señalará sus funciones y facultades.
La autoridad competente se encargará del Registro de Entidades de Certificación y Entidades de Registro o Verificación, las mismas que deberán cumplir con los estándares técnicos internacionales.
Los datos que contendrá el referido Registro deben cumplir principalmente con la function de identificar a las Entidades de Certificación y Entidades de Registro o Verificación.

Artículo 16. Reglamentación
DISPOSICIONES COMPLEMENTARIAS, TRANSITORIAS Y FINALES

PRIMERA. Mientras se cree el Registro señalado en el Artículo 150, la validez de los actos celebrados por Entidades de Certificación y Entidades de Registro o Verificación, en el ámbito de la presente ley, está condicionada a la inscripción respectiva dentro de los 45 (cuarenta y cinco) días siguientes a la creación el referido Registro.

SEGUNDA. El Reglamento de la presente ley incluirá un glosario de términos referidos a esta ley y a las firmas electrónicas en general, observando las definiciones establecidas por los organismos internacionales de los que el Perú es parte.

TERCERA. La autoridad competente podrá aprobar la utilización de otras tecnologías de firmas electrónicas siempre que cumplan con los requisitos establecidos en la presente ley, debiendo establecer el Reglamento las disposiciones que sean necesarias para su adecuación.

Comuníquese al señor Presidente de la República para su promulgación.

En Lima, a los ocho días del mes de mayo del dos mil.