Proyecto Web

Alejandro Exojo Piqueras

25 de enero de 2012

Licencia

Resumen

Este proyecto consiste en el desarrollo de un complemento para el software de creación y gestión de bitácoras («weblogs») WordPress, cuyo objetivo será proporcionar a los usuarios del sitio la firma digital del contenido publicado en él (como por ejemplo comentarios o entradas en el diario) mediante certificados X.509.

La firma de los contenidos se realiza mediante la implementación nativa del navegador, a la que se accede mediante una API en el lado del cliente en JavaScript. Dicha API es una extensión propia de Mozilla que tan solo está disponible en los navegadores que usan el motor Gecko (como FireFox o SeaMonkey), y se accede mediante los métodos del objeto `crypto`, en particular para nuestro caso, la función `signText()`. Dicha función interpela al usuario para que haga uso de uno de los certificados personales instalados, y con él realice la firma del texto pasado.

El complemento de WordPress admite que los comentarios y los contenidos principales añadan en sus metadatos un campo adicional con la firma (en formato PKCS#7 «separada» o «detached»). En el momento de la visualización del contenido, WordPress invoca a OpenSSL para la verificación del mismo contra su firma almacenada para la confirmación de que no ha sido alterado tras el momento de la firma.

Índice general

1. Introducción **5**
- 1.1. Objetivos . . . 5
  - 1.1.1. Objetivos durante la realización . . . 5
  - 1.1.2. Objetivos del desarrollo . . . 6
- 1.2. Estado del arte . . . 6
- 1.3. Estructura de la memoria . . . 6

2. Estudio de viabilidad **8**
- 2.1. El alcance del sistema . . . 8
- 2.2. Estudio de la situación actual . . . 9
- 2.3. Requisitos del sistema . . . 9
- 2.4. Estudio de alternativas . . . 9
- 2.5. Valoración de las alternativas . . . 9
- 2.6. Selección de la solución . . . 10

3. Fundamentos teóricos **11**
- 3.1. Familias de estándares implicadas . . . 11
  - 3.1.1. X.509 . . . 11
  - 3.1.2. RFC 2459 . . . 11
  - 3.1.3. PKCS . . . 11
  - 3.1.4. Cryptographic Message Syntax . . . 12
  - 3.1.5. S/MIME . . . 12
  - 3.1.6. Formatos de archivo usados . . . 13
- 3.2. La firma en el cliente . . . 13
- 3.3. La verificación en el servidor . . . 14

4. Fases del proyecto **15**
- 4.1. Planificación inicial . . . 15
- 4.2. Distribución temporal . . . 17
- 4.3. Realización final . . . 18

5. Diseño **19**
5.1. La lógica en el cliente 19
5.2. La lógica en el servidor 19

6. Implementación **22**
6.1. Maqueta inicial del plugin 22
6.1.1. Objetivos de esta fase 22
6.1.2. Interfaz en el perfil de usuario 23
6.1.3. Añadir las «meta boxes» 23
6.1.4. Guardado de la firma en las «meta boxes» 24
6.1.5. Mostrado de la firma 25
6.1.6. Interfaz de comentarios 25
6.1.7. Guardado de comentarios 26
6.1.8. Visualización del comentario 26
6.2. Entorno de desarrollo de firma 26
6.2.1. Importar en el navegador 32
6.3. Prueba de concepto de verificación 32
6.4. Integración de la verificación en el plugin 34

7. Pruebas **35**

8. Conclusiones **37**
8.1. Objetivos conseguidos 37
8.2. Objetivos no conseguidos 38
8.3. Posibles ampliaciones 38

Capítulo 1

Introducción

1.1. Objetivos

Podemos observar dos grupos de objetivos a cumplir. Por un lado tenemos los objetivos que satisfaremos durante la realización del proyecto, y en segundo lugar, los objetivos que se pretenden alcanzar con la culminación del mismo. Podríamos decir que los primeros son objetivos para el desarrollador, y los segundos objetivos para los usuarios del trabajo del desarrollador.

1.1.1. Objetivos durante la realización

En primer lugar tenemos el adentrarnos en el desarrollo de complementos para WordPress [1] , ya que esta herramienta para construir bitácoras cada vez gana más posibilidades en la creación de sitios web en general, y por tanto gana más masa crítica de usuarios y desarrolladores. WordPress es prácticamente el estándar de facto para construir un weblog con herramientas de software libre.

Habrá que explotar y sacar partido a todas las capacidades disponibles de la API de WordPress [2], así como cumplir los estándares, las convenciones de programación y las buenas prácticas de desarrollo que implica un proyecto profesional en este entorno. Esto implicará documentación para los usuarios, los administradores, y si fuera pertinente, de la API creada.

También hará falta que se hagan uso de las interfaces de usuario habituales en WordPress, así que si fuera necesario, se hará uso de su sistema de preferencias del sitio web general (o del usuario en particular), de los menús, sistema de notificaciones, etc.

El siguiente objetivo será el aprender los conocimientos propios de la firma mediante certificados X.509. Hará falta practicar creando una autoridad

de certificación simulada, sus correspondientes claves autofirmadas y con capacidad de firma digital, así como la creación de certificados firmados por dicha autoridad de certificación, y su posterior integración de las mismas en el navegador.

Y finalmente está el desarrollo de una cierta interfaz para interactuar con el objeto «crypto» [8] del navegador. Para acceder a los certificados almacenados en el navegador (por el usuario, o de serie en el dispositivo criptográfico propio), los navegadores basados en el motor Gecko (típicamente los de Mozilla, como FireFox o SeaMonkey) incluyen un objeto propio llamado «window.crypto» o simplemente «crypto». Este objeto permite el acceso a distintos métodos relacionados con criptografía, incluyendo lo que aquí nos atañe: que el navegador firme, previa validación por parte del usuario, un fragmento de texto para ser enviado a un servidor y ser almacenado y/o validado.

1.1.2. Objetivos del desarrollo

Como objetivos para los usuarios, u objetivos a lograr con el completado del proyecto, tenemos el proporcionar a los usuarios de sitios web basados en WordPress un mecanismo para que los certificados personales de su navegador puedan ser usados para dotar de autenticidad y no repudio al contenido creado en esos sitios web.

Los certificados X.509 son un estándar de facto en su implementación actual, y su uso en navegadores, pero tan solo es habitual que se usen los mismos para firmar correo electrónico (mediante S/MIME), como alternativa al mecanismo de «red de confianza» de PGP / GnuPG, que requiere la interacción previa de las dos partes para poder verificar las firmas.

1.2. Estado del arte

Respecto de WordPress, no se ha encontrado ninguna extensión que proporcione ninguna funcionalidad de firma digital de ninguna clase, ya sea sobre comentarios, entradas de diario o cualquier otro contenido.

Existen otras implementaciones de firma digital, pero estas son ajenas a la API de firma de Mozilla, ya que usan «applets» escritos en Java [**?**] para realizar tanto el acceso a los certificados como el cálculo de la firma en sí.

1.3. Estructura de la memoria

Tras esta introducción, pasamos al estudio de viabilidad del proyecto.

A continuación haremos un repaso de los fundamentos teóricos que están ligados a este desarrollo, y que incluyen la criptografía de clave pública/privada, y sobre todo los distintos mecanismos, estándares y formatos que son relevantes en este caso, como los certificados X.509, ya que son los estándares en los agentes de usuario (navegadores) y en el servidor.

Pasamos entonces a detallar las fases en las que se compone el proyecto, con su correspondiente división en tareas y subtareas, y su solapamiento.

Tras ello se explican los detalles de implementación, tanto en la parte de firmado en el cliente, como la parte de verificación en el servidor, así como la integración lo más adecuada posible con los mecanismos de WordPress.

Seguido tenemos la fase de pruebas. En esta etapa haremos una pequeña batería de ejemplos con los que verificar el funcionamiento de la firma en distintas situaciones.

Capítulo 2

Estudio de viabilidad

El objetivo final del estudio de viabilidad es, una vez se conoce en detalle la necesidad a cubrir, llegar a hacer la elección de la solución que satisfaga la dicha necesidad propuesta. Es posible que existan diversas soluciones a considerar, y también que, de todas ellas, puede que ninguna resulte adecuada.

Cada una de las soluciones ha de considerar los siguientes aspectos:

- Económicos
- Técnicos
- Legales
- Operativos

Además, cada aspecto debe valorar el estado inicial y los requisitos pedidos, y teniendo en cuenta que el impacto en la organización, los costes, y los riesgos.

2.1. El alcance del sistema

Este puede ser una de las secciones más delicadas, puesto que una planificación inadecuada del alcance que pueda tener una solución, podría significar retrasos, costes añadidos, y desviación de los objetivos pretendidos.

Un primer paso es hacer una *descripción general* de las necesidades del proyecto. Después, hay que tener en cuenta cómo afectará a *otros proyectos* futuros o en curso, así como a *otras unidades* (como departamentos o secciones) de la organización.

En nuestro caso, el alcance de la implementación está bien acotado y definido. Nuestro objetivo es proporcionar del código necesario para que un sitio

web basado en WordPress pueda solicitar al cliente la firma de los formularios de comentarios o de contenidos, y que el propio sitio web pueda verificar la firma hecha por los clientes, mostrando en su interfaz el estado de dicha firma.

2.2. Estudio de la situación actual

Para ser precisos, en el estudio de la situación actual empezaremos por *identificar* los departamentos, procesos o sistemas que se prevé que puedan verse afectados. A continuación, describiremos cada uno de estos sistemas, haciendo un diagnóstico de los posibles problemas que le afecten.

En nuestro caso no partimos de una implementación en ninguna institución ni organización, así que partimos de una hoja en blanco para hacer la implementación, y trataremos de asumir un entorno genérico, sin requerimientos particulares.

2.3. Requisitos del sistema

Comencemos por hacer una descripción general de los requisitos del proyecto del que se estudia la viabilidad. Es importante definir una prioridad a cada uno de los requisitos que fijemos, para que en caso de necesidad poder distribuir recursos o sacrificar objetivos si fuera necesario.

2.4. Estudio de alternativas

Se listarán las diferentes alternativas que cumplan los requisitos antes mencionados. De cada alternativa se considerarán tanto los aspectos técnicos como los funcionales, si está basado en otro producto (y las consecuencias legales y económicas que puede suponer).

En nuestro caso, sabemos de la existencia de «applets» de Java que pueden ejecutarse en el navegador, así que una implementación posible es esta. Sin embargo, sigue haciendo falta la verificación en el servidor en WordPress.

2.5. Valoración de las alternativas

En este punto, es momento de evaluar las alternativas en función de:

- la viabilidad económica (costos/beneficios), y

- los riesgos que comportan.

2.6. Selección de la solución

El estudio de viabilidad finaliza con la elección de una solución de entre las estudiadas (si es que existe alguna solución que se considere finalmente viable, ya que una consideración importante es el aceptar que podría no haber ninguna). Habrá que tener en cuenta la información estudiada hasta el momento:

- Descripción general y alcance del provecto
- Situación actual del sistema
- Requisitos de la solución adoptada
- Soluciones alternativas consideradas
- Análisis de costos/beneficios de les soluciones, así como los riesgos asociados.

Capítulo 3

Fundamentos teóricos

3.1. Familias de estándares implicadas

3.1.1. X.509

El estándar X.509 fue creado por el ITU-T para gestionar las diferentes infraestructuras de clave pública (PKI, Public Key Infrastructure). Especifica certificados de clave pública, listas de revocación, las cadenas de certificación necesarias para la firma, etc.

En el ámbito del IETF (Internet Engineering Task Force) se suele hablar de PKIX, o Public Key Infrastructure for X.509, un grupo de trabajo que publica RFC u otra documentación sobre el «Certificate profile» y el «CRL (Certificate Revokation Lists) profile» de la tercera versión de X.509.

3.1.2. RFC 2459

La primera de cuatro partes que componen la infraestructura de clave pública para Internet. Esta especificación delimita tan solo el formato y la semántica de los certificados y de las CRL (listas de revocación de certificados).

3.1.3. PKCS

Se trata de un grupo de especificaciones criptográficos publicados por RSA Security. A pesar de su nombre («Public Key Cryptography Standards»), fueron publicadas por esta empresa con el ánimo de promocionar el uso de criptografía de clave pública/privada, para poder obtener regalías de sus patentes sobre ciertos algoritmos.

Algunas de estas especificaciones sí han pasado por un proceso de estandarización más abierto, y se han convertido en estándares del IETF, como los RFC 3447, 2898, 2315, etc.

PKCS#1

Es la especificación principal, y sirve de base para las demás. Define las propiedades criptográficas de las claves públicas y privadas en formato RSA, así como los algoritmos básicos para cifrar, descifrar, firmar y verificar firmas, así como los mecanismos de codificación y relleno.

Está definido en el RFC 3447.

PKCS#7

En este caso nos proporciona el mecanismo para firmar y/o cifrar mensajes en una infraestructura de clave pública. Define también la manera de hacer llegar la clave pública del certificado al receptor. Este estándar (RFC 2315), formaba parte de S/MIME, pero desde 2010 éste se basa en el RFC 5652, una actualización de CMS («Cryptographic Message Syntax»).

PKCS#12

Se trata de un formato de archivo para encapsular distintos objetos en una sola entidad, habitualmente protegidas mediante cifrado de clave simétrica, para que con una contraseña se pueda bloquear el acceso a dicho archivo. Suele contener tanto claves privadas (de ahí la necesidad de contraseña), como claves públicas.

Suele definir un archivo con terminación «.p12», y es un formato que entienden los navegadores de Mozilla, y que aunque está basado en PFX de Microsoft, también ha sido publicado por RSA.

3.1.4. Cryptographic Message Syntax

Estándar del IETF para mensajes cifrados con criptografía. CMS se basa en la sintaxis de PKCS#7, que a su vez se basa en PEM (Privacy-Enhanced Mail). Esta especificación es clave en otros estándares criptográficos que nos atañen, como S/MIME y PKCS#12.

3.1.5. S/MIME

S/MIME [12] , o «Secure/Multipurpose Internet Mail Extensions», es un estándar para el firmado y cifrado de datos en formato MIME. Éste es un

formato inicialmente creado para el correo electrónico (pero en uso en muchos otros ámbitos a día de hoy) que describe la interpretación a dar a texto en distintos formatos o codificaciones de caracteres, o que incluye adjuntos de distinta clase y consideración.

S/MIME es un estándar (RFC 1847) importante en nuestro caso, ya que la verificación de la firma del contenido del formulario introducido por el usuario se hace en el servidor componiendo un mensaje en formato S/MIME.

Este formato se define a base de dos partes: la primera el mensaje a firmar (en un tipo y subtipo MIME a elegir, pero que en nuestro caso se podrá considerar como texto plano), y la segunda la información de control para la verificación de la firma, que en nuestro caso será el contenido generado por el navegador.

3.1.6. Formatos de archivo usados

Hemos mencionado PEM, y P12, y a ellos podemos unir DER. Con ellos tenemos 3 tipos de archivos bastante habituales, y que describen los que más utilizaremos. PEM y DER están interrelacionados, y normalmente ambos serán equivalentes, tan solo que DER es un formato binario, mientras que su correspondencia en PEM la obtenemos mediante la codificación en base 64.

A estos dos queda añadir los ya mencionados de PKCS#12 y S/MIME.

3.2. La firma en el cliente

Nuestra aplicación comienza principalmente en el servidor, pero no tendría sentido si no hubiera algo que firmar, y un mecanismo para realizar la firma. Puesto que la identificación del usuario se hace mediante una clave privada, esta debe residir en un lugar fiable para el mismo, así que es lógico que esta clave resida en el navegador.

El usuario típicamente deberá disponer de su certificado firmado por la Autoridad de Certificación en un archivo en formato PKCS#12, y este deberá ser importado en el navegador. Desgraciadamente, tan solo FireFox (o su «hermano» SeaMonkey) proporcionan de un mecanismo para que un sitio web interactúe con los certificados del navegador mediante JavaScript, así que tan solo estos navegadores están soportados.

Esto es debido a que tan solo estos navegadores presentan el objeto `crypto` y su método `signText()`. Esta API es una extensión originada en los tiempos de Netscape Navigator, y que perdura en los navegadores de Mozilla como una API propietaria.

3.3. La verificación en el servidor

En el servidor, WordPress deberá recibir dos contenidos. El primero, el texto que ha sido firmado, y el segundo, la firma de dicho texto. La API de Mozilla encargada de producir la firma, recibirá un texto crudo, y sobre él obtendrá una firma, que devolverá al control de la aplicación en forma de un objeto PKCS#7 de tipo «signed object» codificado en base64. [6]

Dicho objeto se compondrá a su vez de un objeto «contentInfo», con tipo de contenido «signedData». Dentro del mismo habrá la información propia de la firma (como la versión del archivo, o el algoritmo de «hash», que en este caso será SHA-1), pero también estarán anidados los certificados necesarios para realizar la verificación. En concreto estará toda la cadena de verificación, excluyendo el certificado «trusted», que se considera que ya posee la implementación que desee verificar la firma.

Capítulo 4

Fases del proyecto

Comenzaremos por listar la planificación inicial que se hizo al inicio del desarrollo, y finalizaremos con un repaso a la ejecución final que se ha realizado a la práctica.

4.1. Planificación inicial

Dividiremos el trabajo en las siguientes 9 tareas diferenciadas.

Despliegue del entorno de desarrollo para WordPress

WordPress está muy estrechamente ligado al lenguaje PHP y la base de datos MySQL. Puede utilizar distintos servidores web, pero el más habitual es Apache. Para el desarrollo de este proyecto es necesario asegurarse de disponer del soporte de MySQL, OpenSSL, MCrypt y PCRE en PHP.

En esta tarea deberemos asegurarnos de disponer de todas las dependencias necesarias, e instalar aquellos paquetes que nos falten en nuestro entorno de trabajo.

Familiarización con el modelo de desarrollo de WordPress

En esta etapa nos centraremos en adquirir los conocimientos básicos para desarrollar un plugin de WordPress. Será necesario habituarse a la API de WordPress, y a la interfaz que presenta al usuario.

Desarrollo de un plugin simple como prueba de concepto

Tras obtener los primeros conocimientos del desarrollo en WordPress, pasaremos a hacer una prueba práctica desarrollando un plugin sencillo. Este

plugin ya debe ir orientado a modificar la interfaz de edición de contenido, y/o de publicación de comentarios.

Despliegue del entorno de desarrollo JavaScript

Esta etapa consistirá en sentar las bases para la siguiente, y tendrá las siguientes subtareas:

- Preparar un navegador de la familia Mozilla (como FireFox, o cualquiera que proporcione el objeto «crypto» en el intérprete de JavaScript mediante la biblioteca NSS), para depurar JavaScript (por ejemplo, mediante las extensiones FireBug o WebDeveloper).
- Crear una autoridad de certificación (partiendo de un certificado autofirmado), así como un certificado X.509.
- Importar el certificado a FireFox.
- Hacer una pequeña maqueta de un formulario HTML con algo de lógica en PHP que sea capaz de verificar una firma enviada a través del formulario.

Desarrollo de una pequeña biblioteca JavaScript

Con la infraestructura de la etapa previa completada, ya podemos pasar a desarrollar una biblioteca en JavaScript que pueda firmar una cadena de texto a partir de la cadena en sí, y el certificado de firma.

En este punto también verificaremos las funciones en PHP que hacen la comprobación de la firma.

Evolución del plugin sencillo

Ahora partiremos del plugin inicial que no era más que una prueba de concepto, para construir un plugin más similar al definitivo, ya que ahora debe presentar al autor de contenido del sistema (entradas, páginas o comentarios) la interfaz para firmar dicho contenido.

En esta etapa aún no es necesario usar la API JavaScript, sino que se puede usar un «mock object» que simule el comportamiento del esa API.

Integración de la biblioteca JavaScript

Consistirá en lograr que el plugin pase a utilizar el código JavaScript creado anteriormente para enviar realmente el resultado de la firma.

Desarrollo y ejecución de una batería de pruebas

Pasamos ahora a completar y ejecutar una batería de pruebas finales que sirvan para acabar de detectar fallos en el sistema de firma.

Rendimiento de la solución (opcional)

Esta última tarea es opcional, y consiste en comparar lo desarrollado con otros plugins similares basados en applets en Java.

4.2. Distribución temporal

- 2 semanas: Familiarización con WordPress y creación del plugin como prueba de concepto.
- 2 semanas: Entorno de desarrollo JavaScript. Incluye la maqueta del formulario con una lógica en PHP de comprobación de la firma, así como la creación del certificado X.509.
- 2 semana: Biblioteca de firma en JavaScript.
- 2 semanas: Evolución del plugin inicial al definitivo, sin uso de JavaScript.
- 1 semana: Integración de la biblioteca JavaScript.
- 2 semana: Pruebas.
- 1 semana: Comparación del rendimiento respecto al de otras implementaciones.
- Siguientes semanas: memoria, posibles correcciones, defensa.

Actividad	Inicio	Fin	Duración
Familiarización con WordPress	3 octubre	16 octubre	2 semanas
Entorno de desarrollo JavaScript	17 octubre	30 octubre	2 semanas
Biblioteca de firma JavaScript	31 octubre	13 noviembre	2 semanas
Evolución del plugin WordPress	14 noviembre	27 noviembre	2 semanas
Integración de la biblioteca en el plugin	28 noviembre	4 diciembre	1 semana
Pruebas	5 diciembre	18 diciembre	2 semanas
Comparación de rendimiento	19 diciembre	25 diciembre	1 semana
Memoria, posibles correcciones, defensa	26 diciembre	según necesidad	según necesidad

4.3. Realización final

En la realización final del proyecto, se han seguido los pasos especificados muy a grandes rasgos. Las fases de estudio inicial de WordPress, de los conceptos de firma y de certificados implicados, etc., fueron más largas de lo previsto. La documentación al respecto resultó no ser todo lo idónea que se hubiera deseado, y fue un duro escollo a superar. En concreto, la documentación de Mozilla sobre su API es prácticamente nula. En los sitios web oficiales de Mozilla no existe ninguna información sobre la función `signText`, ni de los parámetros a recibir, ni del formato de retorno de la misma. Toda la documentación de utilidad se encontró en sitios web de terceros, y archivos antiguos de los documentos de Netscape.

En la familiarización con WordPress el proceso fue similar. La documentación general de WordPress no es mala, pero es algo carente de organización, así que para hallar la API correcta para el uso que se le quería dar al plugin, hubo que recurrir de nuevo a documentación de terceros. La fase de desarrollo del plugin fue brevemente más larga de lo planeado, pero no en exceso.

La biblioteca o API de firma JavaScript no fue realizada como tal. Toda la funcionalidad necesaria del objeto «crypto» está en el método «signText», por lo que no ha sido necesario llevar a cabo ninguna rutina significativa que haga uso de ella para ofrecer un API de más alto nivel.

Capítulo 5

Diseño

El diseño del plugin es extremadamente sencillo. Su composición se divide en tres únicos archivos, cada uno correspondiente a cada rol. El principal es el archivo PHP, que como es lógico, contiene toda la lógica en el lado del servidor. Además, se le añade un archivo JavaScript que incorpora la lógica en el cliente, y finalmente una hoja de estilos CSS para presentar los mensajes propios del plugin en un formato destacado.

5.1. La lógica en el cliente

Puesto que WordPress incluye en su distribución varias bibliotecas de JavaScript, nuestro plugin saca partido de ello. Se podría hacer uso de cualquiera de ellas, pero se ha optado por jQuery, ya que esta es la que más se usa en WordPress, ya que las acciones dinámicas del panel de administración están hechas con jQuery UI.

Aprovechándonos de esta circunstancia, añadimos una simple función de inicialización en el evento `$(document).ready()`.

Esta función de inicialización registra un manejador de eventos para el evento `onClick` sobre el botón que nuestro plugin añade en las distintas interfaces.

Dicho manejador es el que invoca el método `signText` del navegador.

5.2. La lógica en el servidor

El desarrollo de extensiones de WordPress es un proceso en principio sencillo, pero que se complica al querer controlar ciertos aspectos. El principio básico sobre el que sea asienta, es que un plugin es un archivo o un directorio con código PHP, en el cual el sistema explora las primeras líneas en

búsqueda de un comentario especialmente formateado, en el cual WordPress buscará algo de información que mostrar al administrador del sitio web, como el nombre del plugin y una breve descripción.

En nuestro caso partimos de un pequeño directorio, aunque inicialmente el plugin no era más que un único archivo PHP, pero como se ha comentado, se le han añadido el archivo JS y la hoja de estilo.

Cada plugin de WordPress se compone de declaraciones de métodos que el desarrollador desea ejecutar, además de unas llamadas que especifican cuando deben ser ejecutados. Estos forman lo que se llama un sistema de «ganchos» («hooks»). WordPress pone a disposición de terceras partes (o de sí mismo) una serie de momentos en la ejecución del código en las que cualquier desarrollador puede «engancharse» al sistema, y ejecutar una llamada. De estos ganchos existen de dos tipos: filtros y acciones.

Ambos son muy similares, y su distinción está más bien en la manera de interactuar con el contenido. En nuestro caso usamos tan solo dos ganchos de tipo filtro (para los posts y para los comentarios), y el resto son de tipo acción.

Nuestro diseño viene por tanto íntimamente ligado al paradigma que impone WordPress a los autores. Nuestra extensión será una colección de métodos, cada uno pensado para ejecutarse en un instante determinado, junto con sus correspondientes llamadas a `add_action` o `add_filter`.

Se han organizado los distintos fragmentos de código en una especie de grupos, o secciones para intentar mantener la claridad en todo momento. Estas son:

Configuración del sitio Consiste en crear una página en el grupo de ajustes principal, una sección en ella, y bajo la sección, un campo para el certificado de confianza adicional. Todo ello usando los mecanismos de WordPress para mayor integración.

Configuración de usuario Añade un campo de tipo `textarea` al perfil del usuario para que añada su clave pública.

Interfaz para el contenido Añade en la interfaz de edición (usada para la creación de entradas en el diario o páginas, pero también para la edición de comentarios ya introducidos) los llamados `metaboxes`. Estas cajas son la interfaz de WordPress para las páginas de administración.

Interfaz de comentarios Los contenidos de las entradas o páginas pueden ser comentados por los usuarios, y para ello existe una interfaz distinta para extender el formulario.

Estilo y scripts Puesto que nuestro plugin requiere que la página tenga algunos estilos propios, y por supuesto también el código JavaScript antes mencionado para la lógica en el cliente, necesitamos indicarle al sistema que archivos deberá añadir, y bajo que condiciones. En nuestro caso añadimos el script siempre (para la edición de comentarios), y le indicamos que tiene como dependencia jQuery.

OpenSSL Y finalmente, las rutinas de verificación de firmas. Para mayor consistencia, se han creado rutinas que són invocadas tanto para la verificación de comentarios como para la verificación de contenidos.

Capítulo 6

Implementación

6.1. Maqueta inicial del plugin

La primera etapa fundamental del desarrollo será el crear una maqueta (o esqueleto, boceto, etc.) de un plugin para WordPress que satisfaga nuestras necesidades. Estará lógicamente orientado a proporcionar las primeras funcionalidades que finalmente tendrán que estar presentes en el proyecto terminado.

6.1.1. Objetivos de esta fase

En esta fase del proyecto intentaremos adentrarnos en WordPress para aplanar el camino en las siguientes etapas. Intentaremos que nuestra maqueta de plugin tenga ya la funcionalidad siguiente:

- Interfaz para añadir un campo al perfil de usuario (y actualizarla o borrarla según convenga), que corresponderá a la clave pública.
- Nuevamente, una interfaz para añadir un campo al publicar nuevos contenidos (páginas, entradas, etc.) y editarlos.
- Las rutinas correspondientes para salvar los contenidos de dichos campos.
- Las funciones correspondientes que *muestren* si los comentarios o los contenidos han sido firmados.

Debido a la organización de la API de WordPress, cada una de estas tareas debe dividirse en otras subtareas.

Quedarán pendientes para fases posteriores el realizar realmente la firma, y que esta se verifique en el servidor.

6.1.2. Interfaz en el perfil de usuario

Empezamos por añadir un campo en el perfil de usuario donde podrá introducir la clave pública de su certificado. Para ello WordPress cuenta con un par de acciones a las que deberemos conectarnos: `show_user_profile` y `personal_options_update`. En la primera nos basta con imprimir por pantalla el formulario, si fuera preciso recuperando de la base de datos, la clave pública del usuario.

En el segundo caso nuestra función podrá leer mediante la variable `$_POST` el contenido de lo enviado por el formulario. De ahí extraerá la clave, y podrá validarla, procesarla e introducirla en la base de datos.

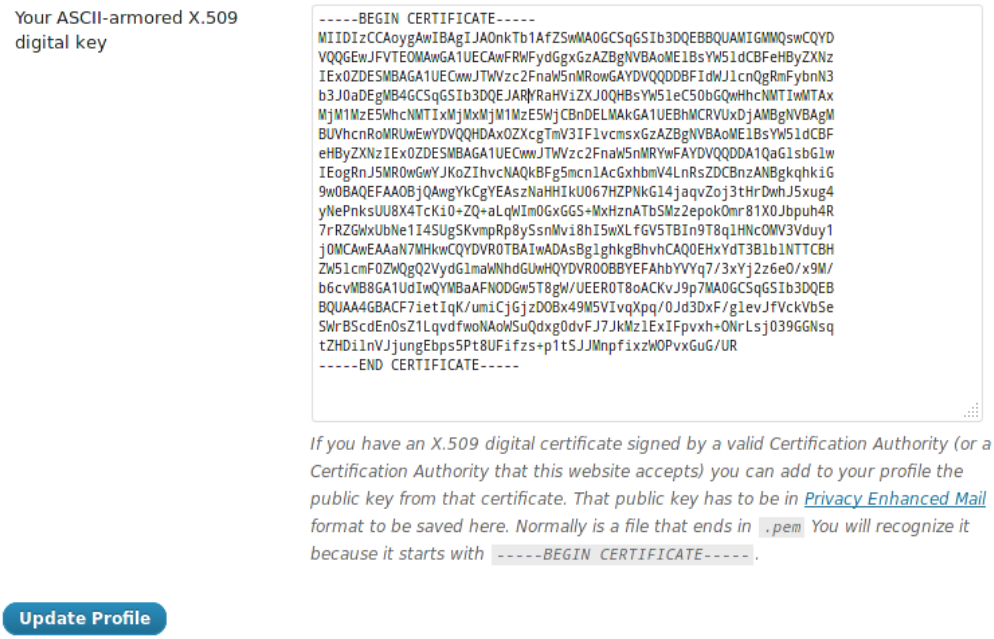

Figura 6.1: Captura de pantalla de la página de perfil de usuario

6.1.3. Añadir las «meta boxes»

En WordPress, la interfaz de administración o edición de contenidos puede extenderse mediante las llamadas «meta boxes». Estas son bloques en los que los complementos pueden introducir campos adicionales. En nuestro caso, como es lógico introduciremos una caja con una casilla de verificación que servirá para preguntar al usuario si desea firmar el contenido del que

está siendo autor. La firma en sí, es guardada y transmitida en un campo oculto del formulario.

Para añadir las cajas necesitamos dos pasos. Primero, debemos usar la acción `add_meta_boxes`, y pasarle una función que se ejecutará en el momento que WordPress necesite saber que cajas debe presentar al usuario. El segundo es esa función, que hará varias llamadas a `add_meta_box`, una para cada tipo de contenido (`post`, `page` y `comment` en nuestro caso).

En esa llamada se pasará una nueva función de nuestra extensión, que debe hacer las comprobaciones pertinentes de autenticación, verificar si es posible hacer firma o no, y caso de que todos los resultados sean positivos, escribir en la salida el formulario para que el usuario pueda marcar la casilla de verificación.

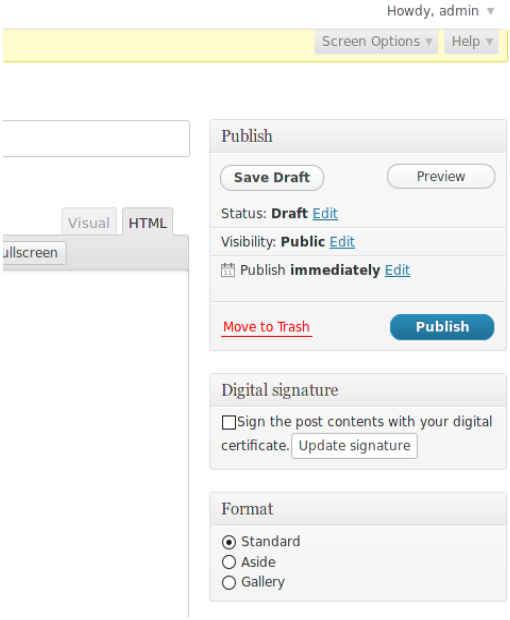

Figura 6.2: Captura de pantalla de la interfaz de edición de contenidos

6.1.4. Guardado de la firma en las «meta boxes»

Una nueva acción de WordPress debe ser tenida en cuenta para que el formulario que hemos presentado anteriormente sea procesado. Es la acción `save_post`, que como su nombre indica, será ejecutada una vez el usuario haya pulsado sobre la opción de guardar (o la función de autoguardado haya sido ejecutado automáticamente).

En este método tan solo comprobaremos que está presente la firma, y si lo está, actualizamos su estado en la base de datos, y si no lo está, la borramos del sistema. Es de destacar que esta acción solo actúa sobre los contenidos centrales de WordPress («posts», páginas, etc.), no sobre comentarios.

6.1.5. Mostrado de la firma

Una nueva situación en que debemos interactuar con WordPress, pero en esta ocasión lo hacemos con un filtro en lugar de una acción (las dos clases de «ganchos» que tiene esta aplicación).

El filtro actuará sobre `the_content`, y antes de realizar acción alguna deberá comprobar en la base de datos si está firmado el contenido, y caso de estarlo, lo notificará añadiendo un mensaje concatenado al contenido original a modo de sufijo.

Este contenido extra se ha optado porque sea un pequeño elemento `div` con un un párrafo que contenga el mensaje con el estado de la firma.

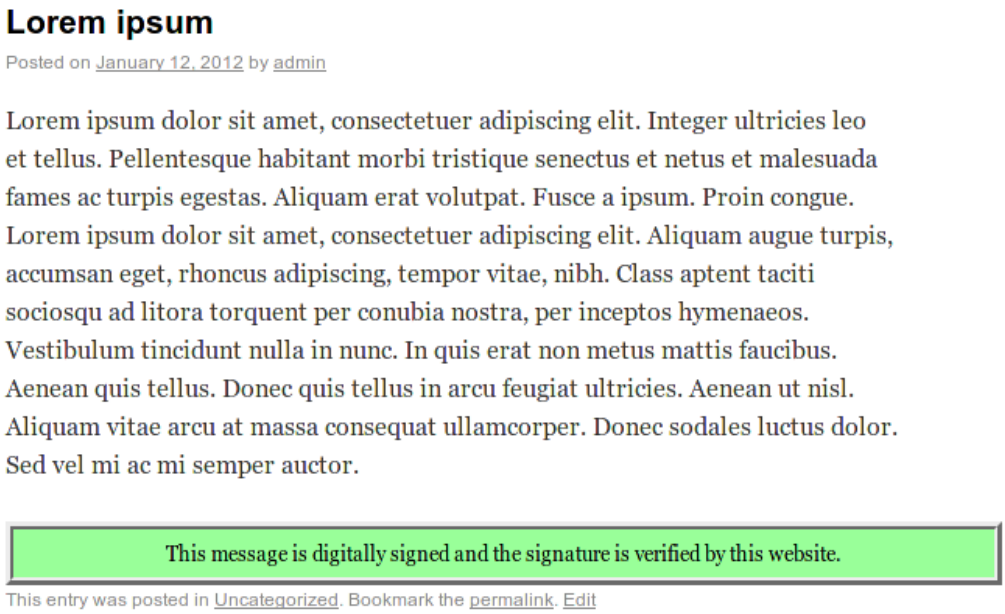

Figura 6.3: Entrada en el diario firmada por el usuario

6.1.6. Interfaz de comentarios

Los pasos anteriores son suficentes para firmar, guardar la firma y mostrarla en los contenidos principales, pero no los comentarios, así que debemos

proceder de manera similar, pero usando otras acciones de WordPress.

Para presentar campos adicionales en el formulario de comentarios de WordPress, usaremos la acción `comment_form_defaults`. A esta acción debemos proporcionarle un método que reciba como parámetro los valores por omisión del formulario de comentario nuevo (en un array asociativo). A dichos valores les añadiremos una casilla para firmar el comentario, o un mensaje informativo caso de que el usuario no haya añadido una clave a su perfil.

6.1.7. Guardado de comentarios

Esta función es muy sencilla, y tan solo guarda el valor de la firma recibido mediante la variable `$_POST` en el campo pertinente de la base de datos, asociada a su comentario.

También es necesario usar la acción `comment_edit_redirect`, que ejecuta una función que actualizará o eliminará el contenido del comentario.

6.1.8. Visualización del comentario

De nuevo haremos uso de una acción de WordPress: `comment_text`. En el método invocado por esta acción obtendremos el estado de la firma, y concatenaremos un mensaje afirmativo si la firma se ha producido (del mismo estilo que con la visualización de los contenidos principales).

El método funciona con comentarios a cualquier nivel de anidación, y permite a los administradores moderar comentarios a discreción, pero por supuesto no permite modificar ni alterar en ninguna forma los comentarios, puesto que esto supondría un error en la verificación.

6.2. Entorno de desarrollo de firma

En este momento pasamos a profundizar en los aspectos plenamente relacionados con la firma. Nuestra tarea será empezar a probar el soporte de certificados en FireFox, y para ello deberemos crear nuestro propio certificado. Puesto que el proceso de creación de un certificado válido implica que este sea firmado por una autoridad de certificación, vemos que tenemos dos posibilidades.

O bien solicitar que nuestro certificado sea firmado por una autoridad (proceso que supone un coste económico y un tiempo de espera), o bien crear una autoridad de certificación propia, sin ningún valor legal ni fuera del ámbito de nuestro desarrollo, pero que es útil para nuestro propósito.

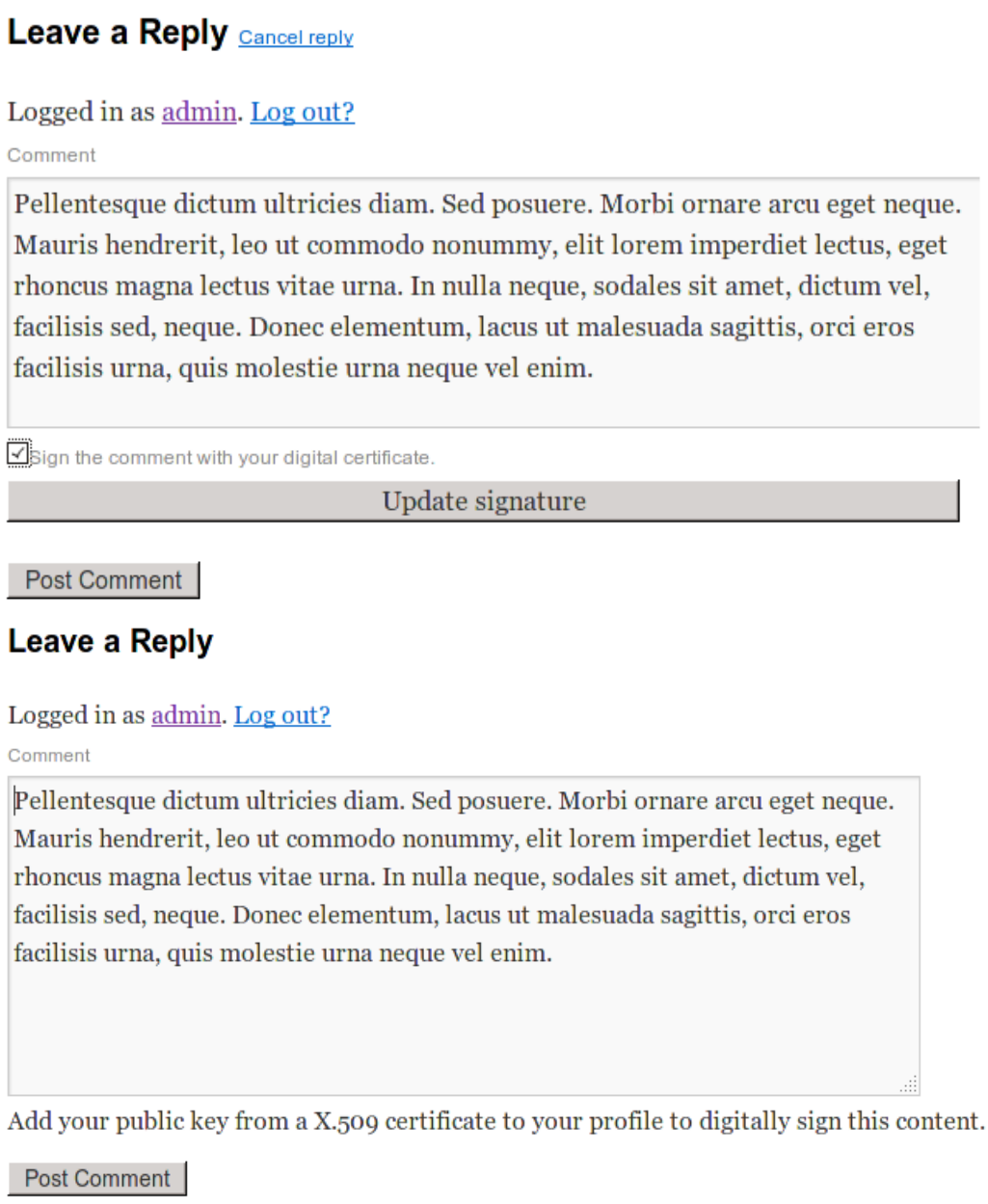

Figura 6.4: Formulario de edición de comentarios

Figura 6.5: Comentarios firmados, con verificación errónea y satisfactoria

Obviamente optamos por lo segundo, ya que esto nos permite probar la creación de firmas tan numerosas veces como deseemos, y sin miedo a que el error en las pruebas suponga ningún coste. La creación de la autoridad es meramente conceptual, ya que tan solo implica asignar valores y nombres a campos meramente identificativos, como el nombre. Por supuesto no tiene que ver con constituir ninguna clase de organización real.

La creación de firmas se puede hacer con distintas herramientas libres. La más habitual, OpenSSL, es la que se utilizó finalmente, pero también se evaluó usar GNU TLS, debido a que dispone de una documentación oficial bastante detallada.

La documentación incluida con OpenSSL es algo más escueta, pero existen muchos documentos que lo facilitan en la red. [10] También resultó de mucha utilidad un programa incluido en la distribución de OpenSSL, el script `CA.pl`, una herramienta ya orientada justo al propósito exacto de usar OpenSSL de la forma adecuada. No es más que un programa que invoca a la instrucción `openssl` entre bastidores, así que las claves o certificados que genere serán igual de válidos en todos los sentidos que usando directamente `openssl`.

Los pasos seguidos son los siguientes.

Crear la Autoridad de Certificación

Pasamos al asistente la opción `-newca` y nos aseguramos de introducir los datos obligatorios, como el *Common Name* y el *Organization Name*.

```
$ /usr/lib/ssl/misc/CA.pl -newca
CA certificate filename (or enter to create)

Making CA certificate ...
Generating a 1024 bit RSA private key
....++++++
..............++++++
writing new private key to './demoCA/private/cakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ZZ
State or Province Name (full name) [Some-State]:World
Locality Name (eg, city) []:None
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Planet Express
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:planex.tld
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            a2:52:f0:b0:35:cd:ba:a8
        Validity
            Not Before: Jan 25 16:55:14 2012 GMT
            Not After : Jan 24 16:55:14 2015 GMT
        Subject:
            countryName               = ZZ
            stateOrProvinceName       = World
            organizationName          = Planet Express
```

```
            commonName                = planex.tld
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                64:E2:11:E7:C3:BA:73:D2:64:F2:9F:63:A9:9B:F9:4F:15:F4:32:4A
            X509v3 Authority Key Identifier:
                keyid:64:E2:11:E7:C3:BA:73:D2:64:F2:9F:63:A9:9B:F9:4F:15:F4:32:4A

            X509v3 Basic Constraints:
                CA:TRUE
Certificate is to be certified until Jan 24 16:55:14 2015 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
```

Esta instrucción nos creará algunos archivos importantes (como `cakey.pem`), que en un escenario real sería crítico conservar a buen recaudo y bajo las medidas apropiadas, puesto que implican claves privadas. A pesar de ello, la propia clave privada viene protegida con el habitual cifrado simétrico.

Otro archivo de especial relevancia es `cacert.pem`, el certificado de la autoridad de certificación (autofirmado en este caso, como era de esperar), aunque en este caso no se trata de un dato confidencial, y sus valores son los que aparecen por pantalla y que se muestran en el listado.

Generar el certificado

Ahora ya podemos ponernos en la situación del usuario que desea generar un certificado y que éste sea verificado por una autoridad. Es por ello que primero se genera no un certificado en sí, sino una solicitud.

```
$ /usr/lib/ssl/misc/CA.pl -newreq
Generating a 1024 bit RSA private key
..................++++++
.............................................................++++++
writing new private key to 'newkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:XX
State or Province Name (full name) [Some-State]:Universe
```

```
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Messaging
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:Phillip J. Fry
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Request is in newreq.pem, private key is in newkey.pem
```

De nuevo tenemos una clave privada a guardar a buen recaudo, y un archivo que será el próximo sobre el que actuaremos, la petición de certificado, guardada en el archivo `newreq.pem`.

Tras obtener la petición, pasamos a interpretar el rol de la Autoridad de Certificación, ya que deberemos firmarla:

```
$ /usr/lib/ssl/misc/CA.pl -sign
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            a2:52:f0:b0:35:cd:ba:a9
        Validity
            Not Before: Jan 25 17:33:03 2012 GMT
            Not After : Jan 24 17:33:03 2013 GMT
        Subject:
            countryName               = XX
            stateOrProvinceName       = Universe
            organizationName          = Messaging
            commonName                = Phillip J. Fry
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                7B:C9:39:94:C0:47:AE:5E:52:9D:1A:D5:4D:A2:D5:87:9B:55:A7:1C
            X509v3 Authority Key Identifier:
                keyid:64:E2:11:E7:C3:BA:73:D2:64:F2:9F:63:A9:9B:F9:4F:15:F4:32:4A

Certificate is to be certified until Jan 24 17:33:03 2013 GMT (365 days)
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
Signed certificate is in newcert.pem
```

Podemos dar el proceso casi por terminado, ya que finalmente tenemos lo que nos hace falta: un certificado válido (dentro de la validez de la Autoridad de Certificación). Tan solo nos falta un último toque final.

6.2.1. Importar en el navegador

El navegador esperará un único archivo en formato PKCS#12 que incluya el certificado y la clave privada. Esto es debido a que, lógicamente, el propósito de importar al navegador es realizar firmas, y para realizar firmas es necesaria la clave privada. Es el paso más sencillo de todos, y consiste tan solo en lo siguiente:

```
$ openssl pkcs12 -export -in newcert.pem -inkey newkey.pem -out key+certificate.p12
Enter pass phrase for newkey.pem:
Enter Export Password:
Verifying - Enter Export Password:
```

Habrá que tener una pequeña precaución al importar al navegador, y es que para que FireFox acepte dicho certificado como válido, deberá tener entre sus firmas una autoridad de confianza. Puesto que tan solo está firmado por nuestra autoridad ficticia, deberemos importar también dicha autoridad entre la lista de entidades de confianza.

6.3. Prueba de concepto de verificación

En este punto, una vez que ya tenemos ubicado en su sitio el código para que WordPress tenga un plugin con las interfaces con el usuario necesarias, será preciso hacer que dicho plugin haga realmente la verificación de las firmas que los usuarios hagan sobre el contenido. Por supuesto también será necesario incluir la lógica en el cliente (mediante JavaScript), para que cuando sea pertinente, el usuario firme el contenido con el certificado de su navegador.

En esta prueba de concepto (incluida en el árbol de código de la entrega de la memoria), se decidió descartar la necesidad de crear un API o biblioteca de JavaScript para la firma. El API expuesta por el navegador para la firma, es lo suficientemente sencilla como para no necesitar de añadidos adicionales.

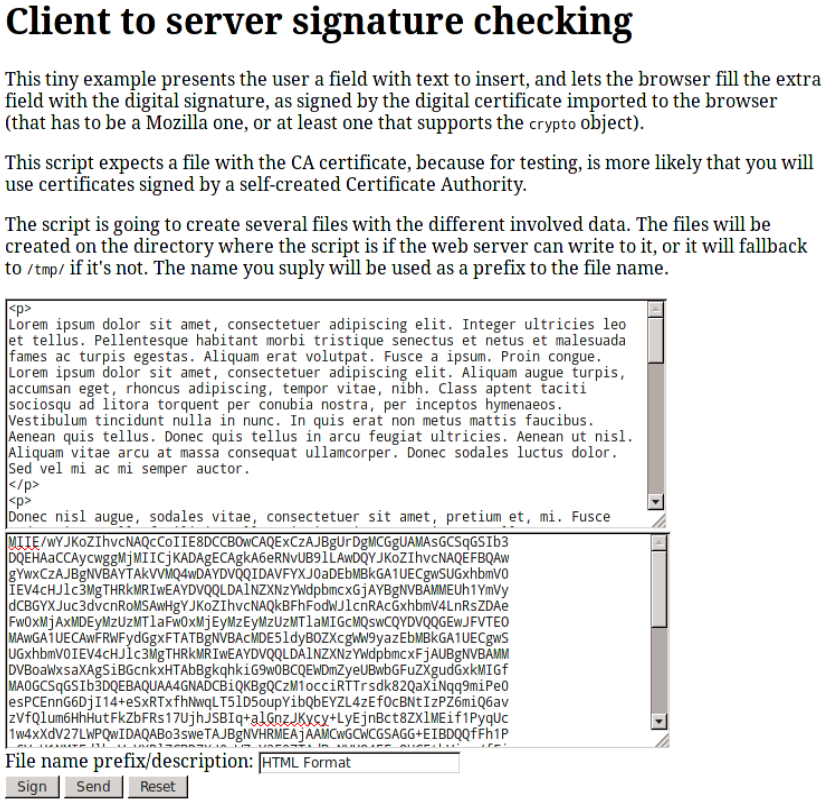

Figura 6.6: Ejemplo en cliente y servidor

6.4. Integración de la verificación en el plugin

Esta es la fase final, en la que extendemos el añadido desarrollado anteriormente para probar si el código de la prueba de concepto funciona en el contexto de WordPress.

Los primeros pasos previos a la integración, es añadir alguna funcionalidad más a la interfaz, puesto que durante el desarrollo anterior, constatamos que es bastante útil, si no directamente imprescindible, que el sitio web pueda indicarle al sistema que se usará una Autoridad de Certificación adicional.

En otros contextos, la configuración para añadir una Autoridad de Certificación adicional consiste en acudir a los archivos de OpenSSL. Esto implicaría poder editar el directorio `/etc`. Queda claro que es inaceptable en esta situación, puesto que una simple aplicación web no debería requerir modificar archivos del sistema, y lo convertiría en un serio escollo para usarse en distintos alojamientos web. Ya que `openssl` acepta el recibir como parámetro un directorio con un certificado adicional, nuestro plugin deberá hacer uso de este dato.

Capítulo 7

Pruebas

La etapa de pruebas ha quedado bastante reducida al respecto de la planificación inicial, ya que no se ha considerado necesaria, y en las pruebas manuales los resultados han sido muy satisfactorios.

Durante el desarrollo de la prueba de concepto, aprovechando que era necesario salvar al disco duro el mensaje y la firma del mismo en distintos archivos, se diseñó de forma que se salvaran con distintos nombres en cada invocación. De esta manera, durante las pruebas se crearon mensajes distintos, con contenidos variados (codificaciones, formato, etc.).

De esta forma se aseguró que los cambios hechos durante el desarrollo no producían regresiones, y si en versiones anteriores el código validaba una firma, en las subsiguientes eso no debería cambiar los resultados.

Esto permitió encontrar un detalle especialmente delicado en la verificación. En las primeras pruebas, se verificaba correctamente el mensaje tan solo de mensajes que no contenían ningún salto de línea. Sin importar la codificación, o el contenido del mismo, la verificación era correcta tan solo en mensajes sin más de una línea.

Así se encontró el detalle de que los contenidos deben de ser normalizados para que OpenSSL no rechace validar los mensajes. La normalización es tan sencilla como eliminar los retornos de carro (`CR` o `\r`).

En la siguiente tabla se presentan los resultados de algunas de las pruebas realizadas. Nótese que se hicieron también pruebas usando el formato S/MIME como formato de entrada para la verificación, pero se descartó en ver que su complejidad añadida resultaba en algunas dificultades para componer un mensaje válido que fuera aceptado por OpenSSL, sin ser un formato que aportara nada adicional, puesto que no pretendíamos firmar múltiples campos ni valores binarios. Además, la verificación de la firma tan solo resultó correcta en los casos en que se suministraba, además del mensaje en S/MIME que unía firma y mensaje, un segundo archivo con el mismo mensa-

je en formato suelto, lo que resultaba muy impráctico comparado con otros procedimientos.

Los otros formatos usados fueron la firma suelta («detached») en el formato entregado directamente por el navegador (PEM), que corresponde al formato binario DER codificado en base64. El formato DER no nos aporta ninguna ventaja significativa (un teórico menor tamaño al no requerir la codificación especial en 7 bits), por lo que no fue usado más que para las pruebas.

Tipo de archivo	PEM	DER	S/MIME y mensaje	S/MIME
Texto ASCII una línea	OK	OK	OK	KO
Texto UTF-8 una línea	OK	OK	OK	KO
Texto ASCII multilínea	OK	OK	OK	KO
Texto UTF-8 multilínea	OK	OK	OK	KO
Texto con formato	OK	OK	OK	KO

Capítulo 8

Conclusiones

8.1. Objetivos conseguidos

Todos los objetivos fundamentales se pueden dar por conseguidos. Desde el punto de vista del alumno, se han adquirido algunos conocimientos adicionales de criptografía, así como de las especificaciones principales, y los estándares de facto usados en Internet para los propósitos de firma (aparte del modelo de Pretty Good Privacy o Gnu Privacy Guard, que se basan en su lugar en el modelo de red de confianza, en lugar de autoridades centrales de certificación).

Se ha experimentado con OpenSSL, tanto en la generación de una Autoridad de Certificación, como el crear certificados con ella. Se ha examinado también GNU TLS como alternativa a OpenSSL debido a su superior documentación, pero GNU TLS está más orientada al cifrado de HTTP, y no tiene el soporte necesario de verificar mensajes S/MIME, o con la firma PKCS#7 separada del mensaje. También se llegó a descubrir y evaluar las herramientas en la línea de instrucciones de NSS, la biblioteca de criptografía de Mozilla.

WordPress ha sido otro tema en el que se ha investigado e indagado. Su API para crear añadidos no es especialmente directa, y es necesario pasar por ciertos aros para lograr algunas acciones. Se ha investigado acerca de la misma, y en algunos casos sobre documentación más específica para lograr ciertas tareas, ya que la documentación oficial presenta algunas carencias.

Finalmente, se ha integrado la funcionalidad de firma de Mozilla en un sitio web. Primero se probó sobre una pequeña prueba de concepto, donde se hacía el firmado sobre un único formulario, y se enviaba al servidor para ser verificada. Posteriormente se integró en su ubicación definitiva en WordPress.

8.2. Objetivos no conseguidos

Toda la funcionalidad esencial se ha implementado, así que todos los objetivos principales se han cumplido. Sin embargo, respecto de los objetivos intermedios, y los pasos especificados en la planificación, existen ciertas divergencias. Por ejemplo, el API JavaScript expuesta por el navegador que ha sido usada, es muy reducida, y se ha limitado al método `signText`. Esto ha supuesto que no ha sido necesario crear una biblioteca, o un API propio en JavaScript que envuelva a la ya existente, puesto que no ha sido necesario crear rutinas especialmente sofisticadas como para que sea útil su reutilización. Todo el código JavaScript utilizado es de una gran sencillez.

El sistema de pruebas ha sido también muy básico, y quizás no tan exhaustivo como se hubiera planteado en algún documento de planificación. Sin embargo, la prueba de concepto que involucraba a cliente y servidor, se ha diseñado de tal forma que se pueda utilizar para hacer pruebas con distintos documentos de entrada en la parte más sensible de la aplicación: la verificación de la firma.

Dicha prueba, permite al usuario introducir un texto de entrada, y asignarle un nombre al archivo que se guardará con el texto introducido, y el texto firmado. Esto ha permitido guardar un pequeño catálogo de documentos con valores variados, y así hacer las pruebas sobre todos ellos de manera sistemática, y así verificar que posteriores cambios en el código no producían regresiones.

8.3. Posibles ampliaciones

El objetivo central del proyecto es bastante acotado, así que las ampliaciones son posibles, pero su temática es más bien reducida. Lo más obvio a mejorar en el proyecto es la usabilidad y la presentación. Un desarrollador con algo más de experiencia el diseño podría mejorar la presentación de los mensajes en el código HTML y CSS de la web, ya que ahora el aspecto es algo sencillo y austero.

Lo siguiente sería proporcionar más y mejor información acerca del contenido firmado y de la Autoridad de Certificación que expidió el certificado firmante. Esto sería especialmente útil para redes de confianza específicas para un grupo o comunidad. Por ejemplo, dentro de una comunidad de usuarios, o dentro de una institución o empresa, podría considerarse como válido el uso de un determinado certificado, pero fuera podría ser desconocido, o inválido.

El caso más obvio sería el de las instituciones estatales, como la Fábrica

Nacional de Moneda y Timbre, que pueden expedir certificados plenamente válidos, y con todos los requerimientos de oficialidad, pero que puede ser desconocido fuera de España. En este caso queda claro que en algunas circunstancias sería de utilidad mostrar el la Autoridad de Certificación.

Otro aspecto posible, sería explorar la posibilidad de que el sitio web, además de verificar (quizás opcionalmente) la firma del lado del servidor, ofrezca al navegador información suficiente para que haga la verificación de la firma también en el lado del cliente. Desgraciadamente esta clase de funcionalidad requeriría en gran medida de que el navegador soporte a su vez el validar contenidos arbitrarios del documento mediante alguna clase de API en JavaScript, pero esto está muy lejos de ser posible. Ya el API del objeto «crypto» es muy limitada, insuficientemente documentada, y ni siquiera es estándar, ya que tan solo es proporcionada por FireFox o SeaMonkey. Cabría estudiar la posibilidad de si es posible lograrlo mediante un plugin del navegador que pudiera estar enlazado con bibliotecas de criptografía, y con cierto acceso al sistema de archivos.

Bibliografía

[1] Matt Mullenweg et al.; *WordPress*. http://wordpress.org/

[2] Matt Mullenweg et al.;; *WordPress Plugin API*. http://codex.wordpress.org/Plugin_API

[3] Adam R. Brown; *WordPress hooks database*. http://adambrown.info/p/wp_hooks

[4] Beau Lebens; *Hooking Into comments*. http://dentedreality.com.au

[5] Brad Williams et. al.; *Professional WordPress Plugin development*. http://www.wrox.com/WileyCDA/WroxTitle/Professional-WordPress-Plugin-Development.productCd-0470916222.html

[6] Netscape; *Signing Text from JavaScript*. http://docs.oracle.com/cd/E19957-01/816-6152-10/contents.htm

[7] Netscape; *Netscape Form Signing*. http://web.archive.org/web/20040821155856/developer.netscape.com/tech/security/index.html?content=formsign/formsign.html

[8] Mozilla; *Using the Mozilla crypto object from JavaScript*. https://developer.mozilla.org/en/JavaScript_crypto

[9] Nikos Mavrogiannopoulos, Simon Josefsson; *GNU TLS: Transport Layer Security Library for the GNU system*.

[10] Paul Heinlein; *OpenSSL Command-Line HOWTO*. http://www.madboa.com/geek/openssl/

[11] OpenSSL; *PKCS#7 manpage*. http://www.openssl.org/docs/apps/pkcs7.html

[12] IETF; *RFC 3851*. http://www.ietf.org/rfc/rfc3851.txt

[13] Greg Frascadore; *Creating Detached PKCS-7 Signatures Using OpenSSL*. http://web.mac.com/nissplus/IslandOfApples/Creating%20Detached%20PKCS7%20Signatures%20Using%20OpenSSL.html

[14] Sun Microsystems; *Network Security Services*. http://docs.oracle.com/cd/E19850-01/816-6400-10/pkcsutil.html

[15] Mozilla; *Overview of NSS Open Source Crypto Libraries*. http://www.mozilla.org/projects/security/pki/nss/overview.html

[16] Mozilla; *NSS: Tools to Ship*. https://wiki.mozilla.org/NSS:ToolsToShip

[17] Universitat Jaume I; *CryptoApplet*. http://proyectostic.uji.es/pr/cryptoapplet/

[18] Juan Antonio Martínez Castaño; *Certificados FNMT en Linux Mini-HOWTO*. http://oasis.dit.upm.es/~jantonio/firmadigital/