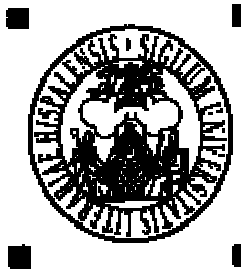


UNIVERSIDAD
de SEVILLA

Proyecto Fin de Carrera

Estudio e implementación de una Autoridad de Certificación

Autor: Gabriel Babiano Huete

Tutor: Teresa Ariza Gómez

“It is insufficient to protect ourselves with laws;
we need to protect ourselves with mathematics”

Bruce Schneier

A Rocío

Índice de contenido

Índice de ilustraciones

Índice de tablas

Estructura de la memoria

Este documento es la memoria del Proyecto Fin de Carrera de Ingeniero de Telecomunicación por la Universidad de Sevilla titulado *"Estudio e implementación de una Autoridad de Certificación"*. Su autor es Gabriel Babiano Huete, tutorado por Teresa Ariza Gómez perteneciente al Área de Ingeniería Telemática bajo el Departamento de Ingeniería de Sistemas y Automática de la Escuela Superior de Ingenieros.

Se encuentra dividido en nueve capítulos y tres anexos.

En el primer capítulo se hace una introducción al Proyecto: la motivación que llevó a su realización y sus objetivos.

En el segundo capítulo se estudia la criptología constituida por la criptografía y el criptoanálisis. Se profundiza en los distintos tipos de criptografía: simétrica y asimétrica, comparando sus ventajas e inconvenientes y cómo se complementan entre sí. Igualmente se estudian las funciones digestoras, las claves de sesión y algoritmos de distribución de claves.

En el tercer capítulo se realiza un estudio de la Infraestructura de Clave Pública (PKI) y sus componentes: los certificados (en particular sus estándares y codificación), las Autoridades de Certificación (CA) y Autoridades de Registro (RA) y cómo su uso mejora los niveles de confianza del PGP. En este capítulo también se realiza un estudio del funcionamiento de algunas CAs y en particular la más conocida, como es VeriSign.

El capítulo cuarto es el pliego de condiciones que contiene: la descripción, características, instalación y utilización del software requerido en el Proyecto. Se ha hecho especial hincapié en las herramientas openssl y keytool cuyo conocimiento ayuda a la comprensión de este Proyecto.

El capítulo quinto está dedicado a la aplicación implementada en este Proyecto que permite la encriptación y desencriptación en RSA utilizando las claves almacenadas en un keystore como es Mastin y su GUI XMastin.

El capítulo sexto está dedicado a la Autoridad de Certificación que se implementa en este Proyecto que firma Peticiones de Firma de Certificados (CSR) en formato PKCS#10 y entrega certificados bajo los estándares X.509 y PKCS#7 tanto en codificación DER como PEM.

El capitulo séptimo contiene el presupuesto del presente Proyecto en forma desglosada.

En el capítulo octavo se trazan posibles líneas de continuación del Proyecto, para las cuales, la lectura de esta memoria, serviría de ayuda.

En el capítulo noveno se exponen las conclusiones a las que se llegan tras la realización del Proyecto así como los logros alcanzados.

El anexo A contiene una descripción de los estándares PKCS establecidos por los Laboratorios de RSA.

El anexo B es el manual de usuario de las aplicaciones Mastin y XMastin que contiene una descripción, sus características, el software requerido, su instalación, uso y desinstalación. Además se introduce el uso de la herramienta keytool para gestionar la keystore utilizada.

El anexo C es el manual de administrador de la Autoridad de Certificación (CA) que contiene una descripción, sus características, el software requerido, su instalación, uso y desinstalación.

Se adjunta además una bibliografía de utilidad y una completa lista de acrónimos usados en esta memoria.

I. Introducción

I.1. Motivación

La extensión de la informática y de las redes de ámbito mundial ha provocado que se incrementen los peligros que sufre la información que circula y es almacenada en los sistemas informáticos. Son necesarias pues, herramientas que oculten esta información a miradas no deseadas y la muestren sin variación a quien esté autorizado. Estas herramientas surgen de algoritmos matemáticos desarrollados por la criptografía.

Como se verá más adelante, la criptografía tradicional simétrica (también conocida como de clave privada) puede garantizar dicha confidencialidad pero tiene lagunas importantes en la distribución de las claves. Estas deficiencias se suplen con la utilización de otro tipo de criptografía más reciente llamada asimétrica o de clave pública. Además de la confidencialidad, la criptografía asimétrica puede aportar nuevas funcionalidades como son la autenticación, firma digital y no repudiación.

Las Autoridades de Certificación (CA) son la base de la Infraestructura de Clave Pública (PKI) necesaria para poder realizar operaciones (sobre todo de carácter económico) en las que se requiera conocer la identidad de la parte contraria. Las herramientas con las que trabajan son los certificados digitales, que no son sino las claves públicas de los sujetos firmadas por dicha CA. Con los certificados, la posible "desconfianza" en la parte contraria se traslada a una posible, aunque menor "desconfianza" en la CA. La cuestión ahora es cómo de confiable es la propia CA.

En una CA comercial existen muchos productos (tipos de certificado), cada uno con un precio y seguro particular, aunque todos ellos se basen en la misma idea de firmado. Dicha colección de productos surge para diversificar el mercado, ya que por ejemplo no va a estar dispuesta a pagar una empresa por montar su servidor de comercio electrónico que un simple usuario por usar correo encriptado. De esta manera, podríamos decir que existen dos tipos de certificados básicos: los de los servidores SSL (para poder implementar HTTPS) y los personales (para, por ejemplo, correo encriptado S/MIME).

Con los certificados SSL es el propio servidor del usuario (o empresa) el que genera el par de claves y con sus datos crea la Petición de Firma de Certificado (CSR); ésta se remite a la CA y es labor de ésta la de cerciorarse de la validez de los datos, ya que peligra la confianza

de los usuarios que tiene depositada en ella, y devolver dicha CSR firmada en formato de certificado.

El tema de los certificados personales es similar, pero ahora el encargado de generar y almacenar los pares de claves para algoritmos asimétricos es una extensión del navegador - cliente de correo S/MIME más populares como son Microsoft Internet Explorer - Microsoft Outlook o Netscape Navigator - Netscape Communicator.

Con el fin de dificultar (e incluso imposibilitar) la migración de los usuarios, existen profundas discrepancias en la forma de realizar dichas tareas, por ejemplo, la generación Netscape la realiza con la etiqueta <KEYGEN> mientras que Microsoft se basa en un control ActiveX y cada uno almacena sus claves y certificados en bases de datos distintas.

Dichas discrepancias proporcionan una oportunidad a otras herramientas, como pudiera ser keytool de JavaSoft de realizar esas tareas, con la ventaja de poder funcionar en las principales plataformas (Win32, Linux, ...). Este Proyecto se aprovechará de esta característica de portabilidad que Java proporciona para desarrollar software independiente de la plataforma para la encriptación / desencriptación con el algoritmo asimétrico RSA.

El establecimiento de una PKI propia para determinadas comunidades (universidades, departamentos, grupos de correo, ...) necesita de una gran cantidad de certificados que expedidos de manera comercial (y renovados cada cierto tiempo) supondrían una pesada carga económica a dicha comunidad, tanta que incluso no sería factible dicha PKI. La CA que implementa este Proyecto, trata así de satisfacer estas necesidades con el menor coste posible.

I.2. Objetivos

- Este Proyecto surje con el ánimo de proporcionar a los usuarios un escenario integral de seguridad confiable basada en la criptografía de clave pública (o asimétrica) como complemento ideal de la criptografía simétrica. Es decir una **PKI propia** basada en una **Autoridad de Certificación**.
- Se busca la **máxima independencia** del navegador - cliente de correo S/MIME que se utilice.
- Se busca una solución con la **máxima portabilidad posible** entre plataformas.
- Con el fin de ampliar el abanico de usuarios, se necesitará un **interfaz amigable** y **fácil de usar**.
- Será primordial el **bajo coste** tanto en dinero como en tiempo.

Basándonos en dichos objetivos de carácter general que regirán en toda la realización del Proyecto, se pueden establecer una serie de objetivos parciales:

- Estudio de la criptología moderna: criptografía, criptoanálisis, criptografía simétrica y asimétrica, sus algoritmos, ventajas, inconvenientes, y cómo se complementan; funciones digestoras, algoritmos de distribución de claves, la PKI, las Autoridades de Certificación, los certificados, sus estándares y tipos de codificación, los estándares PKCS, etc.
- Desarrollo en Java de una aplicación en línea de comandos junto con su API (**Mastin v1.0**) que utilice las claves almacenadas en un keystore para encriptar con el algoritmo RSA según el estándar PKCS#1 en modo de funcionamiento Electronic Code Book (ECB) con relleno Optimal Asymmetric Encryption Padding (OAEP) definido en el PKCS#1 v2.
- Desarrollo en Java de una apliación (**XMastin v1.0**) que hace uso del API Mastin v1.0 con el fin de simplificar su uso y hacerlo más amigable al usuario.
- Desarrollo del **Manual de usuario de Mastin v1.0 y XMastin v1.0.**
- Implementación de una **Autoridad de Certificación (CA)** basada en OpenSSL a través del protocolo HTTPS con el objetivo de ser independiente del navegador.
- Desarrollo del **Manual de administrador de la Autoridad de Certificación (CA) v1.0**.

I.3. Fases del proyecto

Tarea	***Duración***
Estudio previo de la criptología actual: criptografía, criptoanálisis, criptografía simétrica y asimétrica, sus algoritmos, ventajas, inconvenientes, y cómo se complementan; funciones digestoras, algoritmos de distribución de claves, la PKI, las Autoridades de Certificación, los certificados, sus estándares y tipos de codificación, los estándares PKCS, etc	30 días
Definición, desarrollo y depuración de **Mastin v1.0** y **XMastin v1.0**	60 días
Desarrollo del **Manual de usuario de Mastin v1.0** y **XMastin v1.0**	5 días
Definición, desarrollo y depuración de la **Autoridad de Certificación (CA) v1.0**	40 días
Desarrollo del **Manual de administrador de la Autoridad de Certificación (CA) v1.0**	5 días
Redacción de la **memoria**	50 días

II. Criptología

La criptología está formada por dos técnicas complementarias: criptografía y criptoanálisis. A continuación se explica en qué consisten y cuales son sus bases.

II.1. Criptografía

La palabra "criptografía" proviene del griego *kryptos*, que significa esconder, y *gráphein*, escribir, es decir, escritura escondida. La criptografía ha sido usada a través de los años para mandar mensajes confidenciales cuyo propósito es que sólo las personas autorizadas puedan entender el mensaje.

La criptografía se fundamenta en dos operaciones esenciales: la encriptación y la desencriptación. La encriptación está basada, a su vez, en dos componentes: el algoritmo y la clave o claves.

El algoritmo es una transformación matemática que requiere de una componente alfanumérica llamada clave que toma un texto claro (*plaintext*), y lo cambia a unos datos cifrados en un criptograma (*ciphertext*). La operación inversa que obtiene el texto original a partir del cifrado se llama desencriptación y para ella también se necesita de una clave que puede ser idéntica a la que encripta (y estaríamos hablando de clave simétrica) o puede ser distinta (claves asimétricas).

Aunque en los inicios se utilizó criptografía basada en el secreto del algoritmo, al igual que una caja negra (*Ilustración 1*), pronto se vio que no era efectivo y se pasó a que la seguridad dependiese del secreto de la clave (*Ilustración 2*).

Ilustración 1: Criptografía basada en algoritmos secretos

Ilustración 2: Criptografía basada en clave secreta

Los sistemas actuales utilizan **algoritmo público y claves secretas**, debido a los siguientes motivos:

- Los algoritmos públicos se pueden fabricar en cadena, tanto chips de *hardware* como aplicaciones *software*. De esta manera el desarrollo es más barato.
- Los algoritmos públicos están más probados, ya que toda la comunidad científica puede trabajar sobre ellos buscando fallos o agujeros. Un algoritmo secreto puede tener agujeros detectables sin necesidad de conocer su funcionamiento completo, por lo tanto, un criptoanalista puede encontrar fallos aunque no conozca el secreto del algoritmo.
- Es más fácil y más seguro transmitir una clave que todo el funcionamiento de un algoritmo.

Así, un sistema de comunicaciones con criptografía utiliza un algoritmo público para encriptar y otro para desencriptar, pero son completamente inservibles para el criptoanalista sin el conocimiento de la clave (*Ilustración 3*).

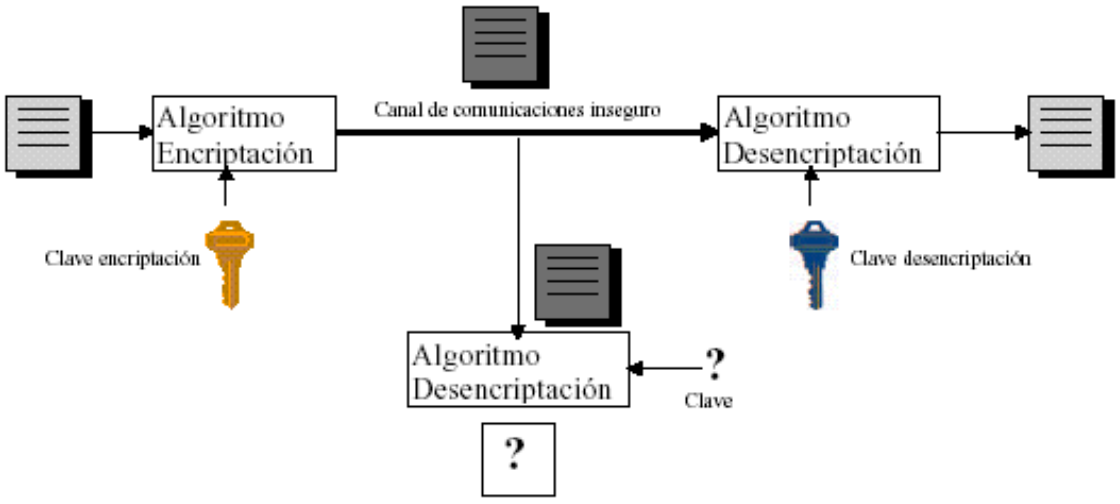


Ilustración 3: Esquema de una comunicación segura

El número de claves posibles que puede utilizar un algoritmo depende de la longitud de la clave con la que se trabaje. Así, un algoritmo que admite una clave de n bits, presenta 2 a la n combinaciones. Por ejemplo, para n=40 bits, que hoy en día se consideraría como una clave débil, daría lugar a un espacio de claves de 2^{40}=1099511627776 que, aunque para la capacidad de procesamiento del ser humano es elevada, no lo es en absoluto para las modernas computadoras y lo será menos en el transcurrir de los tiempos.

II.2. Criptoanálisis

El criptoanálisis trata de descubrir el texto en claro o la clave en una comunicación cifrada y normalmente no depende del conocimiento del algoritmo sino que se basa sobre todo en técnicas matemáticas. La dificultad del análisis depende de la información disponible, así el criptoanalista puede tener acceso a:

- Sólo el criptograma
- El criptograma y además su texto en claro
- Un texto en claro escogido por el criptoanalista y además su criptograma
- Un texto en claro escogido de entre los de diferentes criptogramas junto con éste
- Un criptograma elegido y su correspondiente texto en claro
- Un criptograma escogido de entre los de diferentes textos claros junto con éste

Lógicamente, aumenta la dificultad cuanto menos información se tiene, y si se encontrase la clave, se comprometería toda la seguridad del sistema.

En el criptoanálisis científico se utilizan las siguientes definiciones:

- Definimos como **distancia unívoca** la mínima cantidad de mensaje para poder descifrar la clave (un sistema ideal tiene una distancia unívoca infinito).
- **Romper un sistema** es conseguir un método práctico para descifrar la clave de un sistema criptográfico.

Podemos decir que tenemos varios niveles de seguridad en cuanto a la seguridad se refiere:

- **Sistema incondicionalmente seguro:** aquel cuyo criptograma generado es menor que la distancia unívoca.
- **Sistema probablemente seguro:** aquel que no se ha probado cómo romperlo (por el momento).
- **Sistema condicionalmente seguro:** cuando los analistas potenciales no disponen de medios para romperlo (en la actualidad).

No existen los sistemas completamente seguros, siempre se pueden violar probando todas las claves posibles. Por lo tanto, en criptografía se buscan sistemas que cumplan una de siguientes condiciones:

- El precio para romperlo es más caro que el valor de la información.
- El tiempo necesario para romperlo es más largo que el tiempo de vida de la información.

II.2.1. Ataques contra algoritmos simétricos

II.2.1.1. Sistema de búsqueda de llaves (fuerza bruta)

Se realiza un barrido del espacio de claves, es decir, se prueban todas las claves posibles. Es el método más utilizado pero el menos científico. La elección de las claves se puede hacer siguiendo una lógica (como combinaciones de nombres propios, geográficos, etc.) o aleatoriamente. En el caso de no utilizar una lógica se calcula una probabilidad de acierto del 50% del espacio de claves.

Por ejemplo, en un sistema DES (con una longitud de clave de 56 bits) con una velocidad de comprobación de claves de 2^{55} claves por segundo (cuya tecnología la podrían tener los servicios de información), tardaríamos 1 segundo de media en encontrar la clave ó 2 segundos para probar todas las claves posibles.

II.2.1.2. Criptoanálisis

II.2.1.2.1. Criptoanálisis diferencial

Es un análisis probabilístico a partir del impacto de pequeñas variaciones en el texto a cifrar. Es una de las mejores herramientas criptológicas contra modernos cifradores de bloques iterados como DES, Lucifer, FEAL ...

II.2.1.2.2. Criptoanálisis lineal

Análisis probabilístico a partir de la correlación entre la entrada y la salida contra cifradores de bloques iterados.

II.2.1.2.3. Criptoanálisis lineal-diferencial

Es una mezcla de los dos anteriores.

II.2.1.2.4. Ataques algebraicos

Son técnicas que basan su triunfo en que algunos cifradores de bloque muestran un alto grado de estructura matemática.

Decimos que un algoritmo tiene **estructura de grupo** si tras dos encriptados sucesivos con dos claves distintas, el resultado es equivalente al encriptado con otra clave. En particular el DES no tiene estructura de grupo, lo que permite que sea efectivo el TripleDES.

II.2.1.2.5. Differential Power Analysis (DPA) y Simple Power Analysis (SPA)

También conocido como "análisis diferencial de fallas". Son ataques provocados por "cambios físicos" en los sistemas criptográficos que pueden tener alguna relación con la clave empleada. Como ejemplo en la lucha del Gobierno de los EE.UU. contra este tipo de ataques

se encuentra el programa TEMPEST que trata de evitar la pérdida de información a través de emanaciones electromagnéticas.

II.2.1.3. Ataques basados en el sistema

Otra forma de romper un código es atacar el sistema criptográfico que utiliza el algoritmo criptográfico sin atacar el algoritmo en sí. Un ejemplo es la ruptura del sistema de encriptación de vídeo VC-1 utilizado para encriptar las primeras transmisiones de TV vía satélite.

II.2.1.4. Métodos estadísticos

Son mejores que los sistemas de prueba y ensayo pero sólo sirven para los algoritmos clásicos, actualmente en desuso.

Aprovechan la estadística de la lengua de la fuente. Por ejemplo, en un texto de lengua castellana, la estadística de las letras más comunes es:

Letra	*Porcentaje de aparición*
E	16,80%
A	12,00%
O	8,70%
L	8,00%
S	8,00%

Tabla 1: Estadística de letras más comunes en lengua castellana

Si el sistema sustituye las letras por otros símbolos, utilizando la frecuencia de aparición, es muy fácil detectar la correspondencia entre símbolo y letra.

Si se utilizan agrupaciones de letras el efecto es:

- Más facilidad para la detección de grupos de letras porque se ajustan más a las estadísticas. En español las agrupaciones d-e y q-u-e son muy frecuentes.
- Pero el proceso es más complicado. En español hay 26 letras en el alfabeto, si se agrupan en digramas (2 letras) el número de símbolos es $26^2 = 676$ símbolos.

Una solución fácil contra estos sistemas es comprimir los ficheros antes de la encriptación, así se cambia la estadística y, por lo tanto, se dificulta el análisis.

En resumen, entre los ataques más potentes a la criptografía simétrica se encuentran el criptoanálisis diferencial y lineal, sin embargo no han podido ser muy eficientes, por lo que, después de que un sistema criptográfico es publicado y se muestra inmune a estos dos tipos de ataques, la mayor preocupación es la longitud de las claves.

II.2.2. Ataques contra algoritmos asimétricos

II.2.2.1. Ataques de factorización

Intentan derivar una llave secreta a partir de su llave pública correspondiente. En el caso de RSA, este ataque puede instrumentarse mediante la factorización de un número asociado a la clave pública. Hoy en día la fortaleza del popular algoritmo RSA depende de la dificultad de factorizar números grandes. Dificultad que podría rebajarse con una gran capacidad de cómputo.

II.2.2.2. Ataques algorítmicos

Otra forma de atacar un sistema es encontrar una falla en el problema matemático en el que se basa.

II.2.2.3. Timming Attacks

Mediante una cuidadosa medida de la cantidad de tiempo requerida para realizar operaciones con las claves, los atacantes pueden encontrar exponentes Diffie-Hellman, factores de las claves RSA y romper otros criptosistemas. Contra un sistema vulnerable, el ataque no tiene prácticamente coste y a menudo, sólo se requiere el conocimiento del criptograma.

II.3. Clasificación de la criptografía

Las técnicas de criptografía moderna se pueden clasificar en dos según el tipo de clave utilizado:

- Criptografía simétrica también conocida como de clave secreta.
- Criptografía asimétrica también conocida como de clave pública.

II.3.1. Criptografía simétrica

Es el sistema de criptografía más simple y antiguo.

Toda la seguridad está basada en la privacidad de esta clave secreta, llamada simétrica porque es la misma para encriptar y desencriptar. El emisor del mensaje genera una clave y después la transmite mediante un canal seguro a todos los usuarios autorizados a recibir mensajes**. La distribución de claves es el gran problema de los sistemas simétricos**, ya

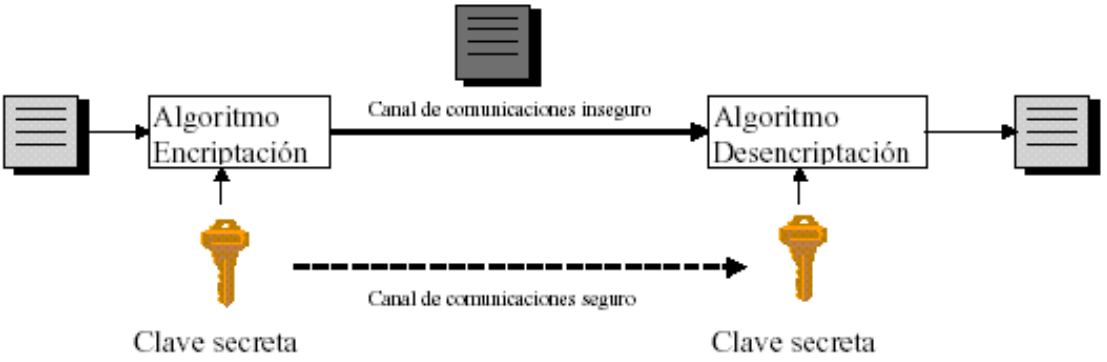


Ilustración 4: Criptografía simétrica

que necesita de un canal de comunicaciones seguro. Hoy en día se resuelve mediante sistemas asimétricos montados únicamente para transmitir claves simétricas como es el caso del algoritmo de Diffie y Helmann .

Estos sistemas **sólo permiten confidencialidad y no autenticación ni firma digital**. Para mantener la confidencialidad delante de un criptoanalista, el algoritmo debe cumplir las siguientes condiciones:

- Conocido el criptograma no se puede descifrar el texto ni adivinar la clave.
- Conocido el texto y el criptograma es más caro (en tiempo y/o dinero) descifrar la clave que el valor de la información.

Para la segunda condición siempre existe la posibilidad de "prueba y ensayo" para encontrar la clave, es decir, probar todas las claves posibles hasta encontrar la que descifra el criptograma. La seguridad respecto a este tipo de ataque depende de la longitud de la clave y lógicamente, cuanto mayor es la longitud de clave, mayor es la seguridad que proporciona.

Existe una clasificación de este tipo de criptografía en tres familias, la criptografía simétrica de **bloques** (*block cipher*), la criptografía simétrica de **flujo** (*stream cipher*) y la criptografia simétrica de **resumen** (*hash functions*). Aunque con ligeras modificaciones, un sistema de criptografía simétrica de bloques puede modificarse para convertirse en alguna de las otras dos formas, sin embargo es importante verlas por separado dado que se usan en diferentes aplicaciones.

En los algoritmos simétricos que encriptan bloques de texto, el tamaño de los bloques puede ser constante o variable según el tipo de algoritmo. Dependiendo de la naturaleza de la aplicación, tienen cuatro formas básicas de funcionamiento:

- **Electronic CodeBook (ECB).** Para mensajes cortos de menos de 64 bits. Se encriptan los bloques de texto por separado (*Ilustración* 5).

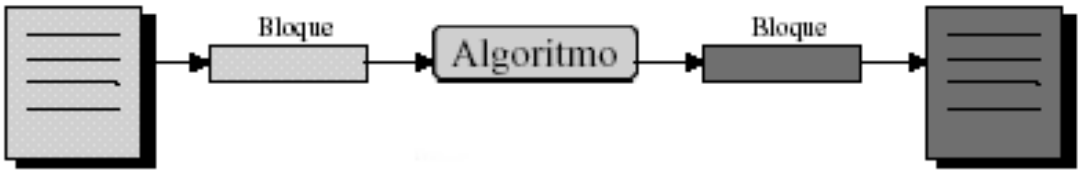


Ilustración 5: Modo ECB

- **Cipher Block Chainning (CBC).** Para mensajes largos. Los bloques de criptograma se relacionan entre ellos mediante funciones OR-EXCLUSIVA (*Ilustración* 6).

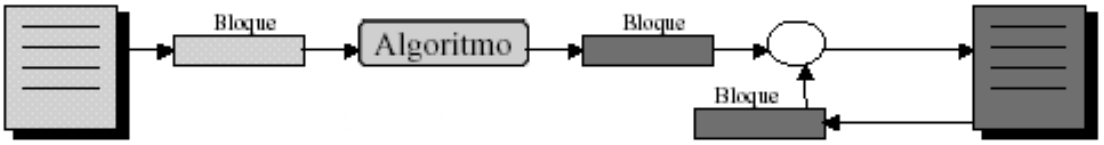


Ilustración 6: Modo CBC

- **Cipher FeedBack (CFB).** Se realiza una OR-EXCLUSIVA entre caracteres o bits aislados del texto y las salidas del algoritmo. El algoritmo utiliza como entrada los criptogramas (*Ilustración* 7).

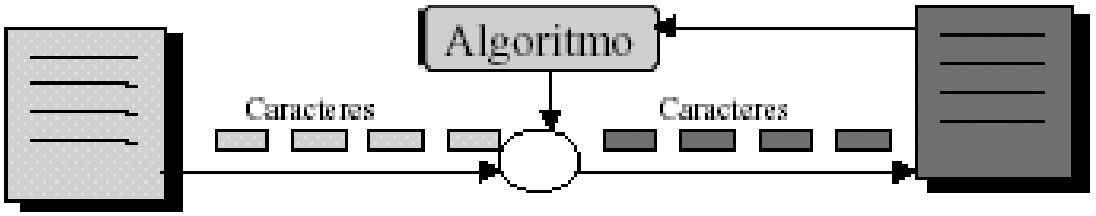


Ilustración 7: Modo CFB

- ***Output FeedBack (OFB).*** Al igual que el CFB, realiza una OR-EXCLUSIVA entre caracteres o bits aislados del texto y las salidas del algoritmo. Pero éste utiliza como entradas sus propias salidas, por lo tanto no depende del texto y evita la propagación de errores. Es además un generador de números aleatorios (*Ilustración* 8).

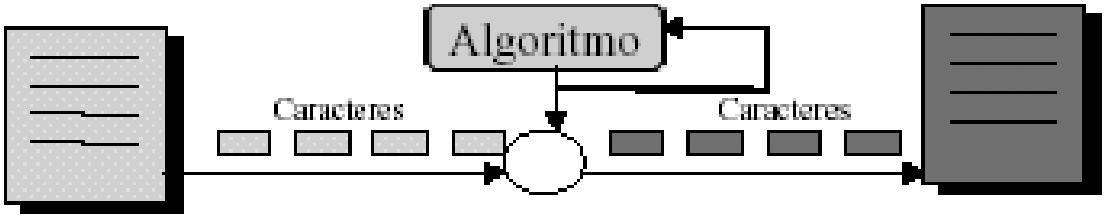


Ilustración 8: Modo OFB

Los algoritmos simétricos son más sencillos que los asimétricos, por ese motivo los procesos son más simples y rápidos. A continuación se enumeran y comentan los más utilizados.

II.3.1.1. DES o DEA (FIPS 46-2)

En 1971 IBM inventó un algoritmo de encriptación simétrico basado en la aplicación de teorías existentes sobre criptografía tales como el cifrador de ***Feistel*** o cifradores de

producto. Se llamó *Lucifer* y funcionaba con claves simétricas de 128 bits. Fue vendido en exclusividad a la empresa de seguros Lloyd's.

En 1973 el National Bureau of Standards (NBS) de los EE.UU. convocó un concurso para elegir un estándar de encriptación para la seguridad de los documentos oficiales. Este concurso fue ganado en 1977 por los inventores del *Lucifer* con una versión mejorada, este algoritmo se denominó Data Encryption Standard (DES).

En 1981 American National Standards Institute (ANSI) aprobó DES (X3.92) y lo llamaron Data Encription Algorithm (DEA). Desde entonces el estándar se difundió ampliamente en todo tipo de dispositivos. En 1994 DES dejó de ser prohibido a nivel de software por lo que aumentaron sus implementaciones

El National Institute of Standards Technology (NIST), sucesor del NBS, reprobó el DES desde 1994 hasta diciembre de 1998. Pero ya en 1997 decidió no prolongar la prórroga y no volver a utilizarlo; por lo tanto, se convocó un concurso para buscar un nuevo sistema. Este sistema se denominaría Advanced Encryption Standard (AES) y el algoritmo utilizado Advanced Encryption Algorithm (AEA).

El algoritmo DES toma como entrada un bloque de 64 bits del mensaje y éste se somete a 16 iteraciones con un bloque de **clave de 56 bits**. En la práctica el bloque de la clave tiene 64 bits, ya que a cada conjunto de 7 bits se le agrega un bit que puede ser usado como paridad. En total, una clave de 56 bits más 8 de paridad.

El sistema de desencriptación es muy similar, así se facilita la implementación de los dos tanto en hardware como en software.

II.3.1.1.1. Ventajas del DES

- Es **el más extendido** en el mundo, por lo tanto, es el que más máquinas utilizan (por ejemplo UNIX), más barato, más probado, etc.
- Es muy **rápido y fácil de implementar**.
- Ha sido durate casi 30 años considerado irrompible con un sistema práctico, pero ya no se considera así en la actualidad como veremos a continuación.

II.3.1.1.2. Inconvenientes del DES

- El criptoanálisis diferencial es capaz de romper el DES en 2^{47}=140737488355328 iteraciones si se conocen textos y criptogramas elegidos, es decir, si se tiene acceso al encriptador. Hoy en día no es un criptoanálisis práctico.
- Pero el principal inconveniente es que **la clave es corta** para la capacidad de cómputo actual en un ataque de prueba y ensayo mediante máquinas trabajando en paralelo a través de una red como Internet. Por este motivo, ya no es el estándar de seguridad de los EE.UU..
- El sistema **no permite longitud de clave variable**. Los algoritmos que permiten elegir la longitud de clave son mucho más prácticos. Las ventajas son las siguientes:

• El usuario del sistema puede resolver el compromiso entre velocidad de proceso y seguridad eligiendo la longitud de la clave sin cambiar el algoritmo.

• El sistema se adapta a los avances en velocidad de proceso. Cuando aumenta la velocidad de cálculo de las máquinas se utiliza una clave más larga, que mantiene la eficiencia del usuario y la dificultad de análisis con las nuevas máquinas.

• Permite crear limitaciones legales por longitud de clave a los EE.UU., así el algoritmo es libre pero la utilización está limitada por el tamaño de clave usada.

II.3.1.2. DESX

Consiste en una sencilla modificación al algoritmo DES, construida alrededor de dos pasos de blanqueo que al parecer mejora mucho la seguridad del algoritmo, haciendo casi imposible la búsqueda de la llave.

II.3.1.3. *TripleDES* (TDES) o DESede

Surge para evitar el problema de la clave corta del DES y continuar utilizando el algoritmo y los dispositivos ya creados.

Gracias a que el DES no tiene estructura de grupo:

- no existe una clave K3 tal que $E_{K2}(E_{K1}(M))=E_{K3}(M)$ (ataque "meet in the middle")
- no existe una clave K4 tal que $E_{K3}(D_{K2}(E_{K1}(M)))=E_{K4}(M)$

Surgió un sistema basado en tres iteraciones del algoritmo, llamado *TripleDES* (TDES) compatible con el DES simple que puede utilizarse de dos maneras:

II.3.1.3.1. Claves de 128 bits

Aunque en realidad es una clave de **112 bits** más 16 bits de paridad, como resultado de juntar dos claves DES de 56 bits y 8 bits de paridad.

Como podemos ver en la *Ilustración* 9, para obtener un criptograma cifrado con el TDES, hemos de pasar el texto en claro por un DES (al que aplicamos 64 de los 128 bits de clave), luego por un DES inverso tambien llamado ANTIDES (donde aplicamos los 64 bits restantes) y posteriormente por un nuevo DES con los primeros 64 bits de clave.

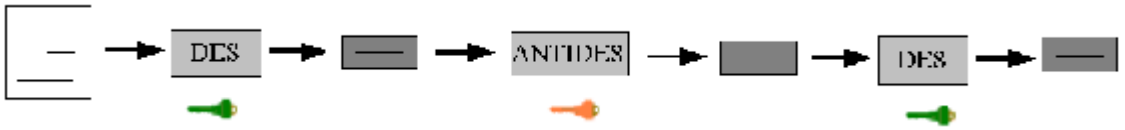


Ilustración 9: TripleDES con clave de 112 bits

II.3.1.3.2. Claves de 168 bits

El proceso es similar al del TripleDES de 128 bits pero utilizando para cada parte una clave DES de 56 bits distinta lo que supone un total de 168 bits (*Ilustración* 10)

Ilustración 10: TripleDES con clave de 168 bits

Como se ha comentado, este algoritmo es compatible con el DES simple sin más que aplicar la misma clave de 56 bits en cada una de las tres partes del proceso; de tal manera que primero cifraríamos con el DES, posteriormente descifraríamos lo anterior ya que usamos la misma clave en el ANTIDES, y terminaríamos cifrando con el DES con la misma clave. Resultado: un cifrado con DES simple.

II.3.1.4. Advanced Encryption Standard (AES) / Rijndael

Como ya he comentado, el NIST convocó en 1997 un concurso para buscar el sustituto al DES debido a la corta longitud de clave. El sistema ganador se denominaría Advanced Encryption Standard (AES) y el algoritmo utilizado Advanced Encryption Algorithm (AEA).

Las propuestas fueron presentadas antes de junio de 1998 y se realizó una primera ronda para eliminar candidatos. En agosto de 1998 se publicó la lista de los 15 algoritmos candidatos (de algunos de ellos hablaremos posteriormente):

Nombre del algoritmo	Creadores del algoritmo
CAST-256	*Entrust Technologies, Inc.*
CRYPTON	*Future Systems, Inc.*
DEAL	*Richard Outerbridge, Lars Knudsen*
DFC	*CNRS - Centre National pour la Recherche Scientifique – Ecole Normale Superieure*
E2	*NTT - Nippon Telegraph and Telephone Corporation*
FROG	*TecApro Internacional S.A.*
HPC	*Rich Schroeppel*
LOKI97	*Lawrie Brown, Josef Pieprzyk, Jennifer Seberry*
MAGENTA	*Deutsche Telekom AG*
MARS	*IBM*
RC6	*RSA Laboratories*
RIJNDAEL	*Joan Daemen, Vincent Rijmen*
SAFER+	*Cylink Corporation*

Nombre del algoritmo	Creadores del algoritmo
SERPENT	*Ross Anderson, Eli Biham, Lars Knudsen*
TWOFISH	*Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson*

Tabla 2: 15 algoritmos candidatos al AES

En agosto de 1999 el NIST publicó la lista de los 5 algoritmos que pasaron la primera ronda:

Nombre del algoritmo	Creadores del algoritmo
MARS	*IBM*
RC6	*RSA Laboratories*
RIJNDAEL	*Joan Daemen, Vincent Rijmen*
SERPENT	*Ross Anderson, Eli Biham, Lars Knudsen*
TWOFISH	*Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson*

Tabla 3: 5 algoritmos finalistas al AES

El 2 de octubre de 2000 se conoció finalmente al ganador que resultó ser el algoritmo **Rijndael** creado por los belgas Vincent Rijmen y Joan Daemen (sucesor de otro cifrador de bloques llamado Square). Hay que hacer notar sin embargo, que AES es el Rijndael pero restringido a **128, 192 ó 256 bits de longitud de clave** y 128 bits de tamaño de bloque. Rijndael permite fácilmente la extensión de la longitud de clave en pasos de 32 bits.

Una de las grandes ventajas, que fue además uno de los principales requisitos para optar somo sucesor del DES, es que puede ser usado tanto como **cifrador de bloques** (*block cipher*), **cifrador de flujo** (*stream cipher*), **función de resumen** (*hash function*) y como **generador de números pseudoaleatorios**.

II.3.1.5. International Data Encryption Algorithm (IDEA)

Publicado en 1990 por *James L. Massey* y *Xuejia Lai* del *Swiss Federal Institute of Technology* inventaron un algoritmo nuevo denominado IDEA a partir del PES y del IPES posteriormente.

Este algoritmo está **libre de restricciones y permisos nacionales** y es de **libre distribución por Internet.** Ésto ha hecho que sea un algoritmo muy popular, sobre todo fuera de los EE.UU., utilizándose en sistemas como: UNIX en Europa, **PGP** para correo electrónico, etc.

Trabaja con **bloques de texto de 64 bits** y una **clave de 128 bits** y está basado en un cifrador Feistel de 8 rondas. Puede funcionar con los 4 modos: ECB, CBC, CFB y OFB. Siempre opera con números de 16 bits utilizando operaciones como OR-EXCLUSIVA, suma de enteros o multiplicación de enteros.

El algoritmo de desencriptación es muy similar. Por estos motivos es **sencillo de programar y rápido**.

Una segunda versión de este algoritmo lo protegía de ataques por análisis diferencial por lo que la única manera de romperlo es por fuerza bruta

II.3.1.6. Ron´s Codes o Rivest´s Ciphers

Fueron inventados por Ron L. Rivest y son propiedad de *RSA Security Inc* (www.rsasecurity.com). De ellos podemos destacar RC2, RC4, RC5 y RC6:

II.3.1.6.1. RC2 (rfc 2268)

Fue secreto comercial de RSA, pero fue divulgado anónimamente en Usenet en 1996.

Encripta **bloques de 64 bits** y junto con la clave de longitud variable (**entre 1 y 2048 bits**), generan una tabla de 128 bits que se usan para cifrar. Además de dichos bits, a la clave se le adjunta otra cadena (de 40 a 88 bits) llamada *salt* que dificulta la tarea a los criptoanalistas que precomputan tablas de búsqueda de posibles encriptaciones. Es de dos a tres veces más rápido que el DES.

Junto con el RC4, posee un estatus especial que concede el Gobierno de los EE.UU. que simplifica las tareas de exportación.

II.3.1.6.2. RC4

Es un secreto comercial de *RSA Security Inc*. pero el código es de dominio público (publicado en Usenet en 1994). Es un cifrador de flujo consistente en hacer un XOR al mensaje con una tabla de 256 bytes que se supone aleatoria y que se desprende de la clave de longitud variable (de 1 a 2048 bits)(*Ilustración 11*).

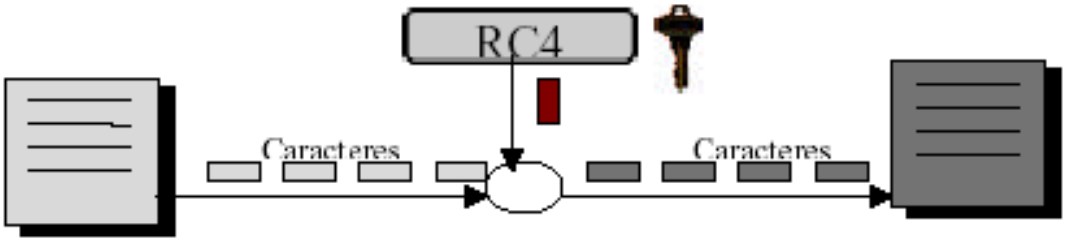


Ilustración 11: Cifrado con RC4

Junto con el RC2, posee un estatus especial que concede el Gobierno de los EE.UU. que simplifica las tareas de exportación.

II.3.1.6.3. RC5 (rfc 2040)

Publicado en 1994. Está basado en el RC4, pero el RC5 usa otra operacion, llamada dependencia de datos, que aplica shifts a los datos para obtener así el mensaje cifrado. Es, a diferencia del RC4, un cifrador de bloques y tiene una arquitectura orientada a **palabra de tamaño variable** w de **16, 32 ó 64 bits**. El número de rondas (r) y la **longitud de la clave** (b) también son variables (**de 0 a 2048 bits)** al igual que la mayoría de nuevos algoritmos. Esto permite adaptarse a las necesidades de velocidad/seguridad de cada aplicación.

Podemos concretar el algoritmo de la siguiente manera: RC5–w, RC5–w/r, y RC5–w/r/b. Una elección típica es el RC5-32/12/16. Se recomienda usar 12 ciclos con el RC5-32 y 16 con el RC5-64. El tamaño de los bloques de texto en claro y del texto cifrado es de 2w.

Tiene una descripción muy compacta y es adecuada tanto para hardware como para software.

La empresa ***Netscape*** **utiliza la versión RC5 para su sistema de seguridad SSL**, por ese motivo se ha extendido ampliamente.

II.3.1.6.4. RC6

El cifrador de bloque RC6 fue diseñado por Ron Rivest en colaboración con *RSA Laboratories* (www.rsasecurity.com/rsalabs/rc6) como una mejora del RC5, que al igual que él, hace uso esencialmente de rotaciones dependientes de datos.

RSA Laboratories envió el RC6 al NIST como candidato a sucesor del DES en el concurso AES que finalmente ganó Rijndael .

Es un algoritmo simple y elegante, muy rápido en plataformas de 32 bits y con rápida preparación de la clave. Desafortunadamente tiene un estrecho margen de seguridad.

II.3.1.7. FEAL

El Fast Data Encipherment Algorithm (FEAL) es una famila de algoritmos presentados por *Shimizu* y *Miyaguchi* que ha jugado un papel crítico en el desarrollo y refinamiento de varias técnicas criptoanalíticas, incluyendo el análisis lineal y diferencial.

FEAL-N (cifrador de N rondas similar al DES) encripta bloques de 64 bits con una **clave de 64 bits**.

FEAL fue diseñado en aras a la **velocidad y simplicidad** especialmente para microprocesadores de 8 bits por lo que usa operaciones orientadas a byte (8 bits) y evita permutaciones de bit y búsquedas en tablas.

FEAL-4 se posicionó como una alternativa al DES pero se descubrió mucho menos seguro de lo que se esperaba. Lo mismo sucedió con el FEAL-8. Ya el FEAL -16 y el FEAL-32 ofrecen unos niveles de seguridad comparable al DES pero su throughput decrece conforme aumenta el número de rondas. Además, mientras la velocidad de las implementaciones DES puede ser mejorado haciendo crecer el tamaño de las tablas de búsqueda, parece ser más difícil para el FEAL. En resumen, **esta familia de algoritmos se considera insegura y no se recomienda su utilización**.

II.3.1.8. BlowFish

BlowFish es un algoritmo de cifrado por bloques creado por *Bruce Schneier* (www.counterpane.com) que no está patentado, es gratuito y no requiere licencia. Fue diseñado para ser rápido, compacto, simple, seguro y robusto (a diferencia de DES, su seguridad no disminuye por simples errores de programación). Las operaciones XOR y sumas que realiza son de 32 bits de longitud de palabra lo que lo convierte en más veloz que el DES.

La longitud de la clave de Blowfish es variable, **múltiplo de 64 bits** y puede llegar a tener una longitud de **hasta 448 bits**.

El algoritmo de cifrado por bloques Blowfish, que cifra datos en **bloques de 64-bits** al mismo tiempo, es dividido en dos partes: claves de expansión y cifrado de datos.

- Las claves de expansión convierten una clave de más de 448 bits en varias subclaves que totalizan 4168 bytes. Blowfish utiliza un gran número de subclaves que deben ser preprocesadas antes de cualquier proceso de cifrado o descifrado.
- El cifrado de datos consiste en una función simple que permite **16 iteraciones**. Cada iteración llamada *Around@* consiste en la permutación de una clave dependiente y una sustitución de una clave y datos dependiente.

II.3.1.9. TwoFish

TwoFish es un algoritmo de cifrado por bloques, creado por *Bruce Schneier* (www.counterpane.com) que encripta **bloques de 128 bits** con una **clave de 128, 192 ó 256 bits** de longitud. Al igual que BlowFish, no está patentado, es gratuito y no requiere licencia.

II.3.1.10. Stealth

El cifrado Stealth es el nombre genérico para una familia de algoritmos simétricos de criptografía desarrollado por *IQ international*. Todas las variantes de la encriptación Stealth incluyen las siguientes características:

- Alta velocidad de proceso.
- Muy seguro. La técnica típica de la criptografía Stealth utiliza $2^{50.000}$ combinaciones teóricas, limitadas únicamente por la longitud de la clave. Todos los datos son cifrados de manera aleatoria y totalmente incomprensible. Pequeños cambios de clave suponen datos cifrados diferentes.
- Operación que llega al byte. Hasta un simple byte puede ser cifrado o descifrado.
- Acceso aleatorio. Cualquier volumen de datos puede ser descifrado/recifrado desde cualquier byte-offset accediendo de modo aleatorio completo.
- No existe expansion de datos. Los datos cifrados tienen idénticos bytes de orden y longitud.
- Integridad de errores. A diferencia de otros algoritmos, los errores no se propagan y las corrupciones en datos cifrados sólo afectarán a los bytes correspondientes de descifrado.
- No existen claves débiles. A diferencia de otras fórmulas algorítmicas no existen claves de cifrado débiles inherentes.

- El cifrado Stealth está disponible en un número muy amplio de fórmulas y gran variedad de longitud de claves. Existen diversas variantes del cifrado Stealth con variedad de plataformas como son los estándares 1040/1060.

II.3.1.11. MARS

Fue diseñado por IBM (www.ibm.com).

Posee un buen margen de seguridad. Es innovador por su estructura heterogénea.

Como la mayoría de los algoritmos modernos, tiene un **tamaño variable de claves**, y su rendimiento depende de la disponibilidad de funciones para multiplicar y desplazar.

Resulta muy rápido en plataformas 32 bits aunque es complejo (necesita mucha ROM).

II.3.1.12. SAFER

El Secure And Fast Encryption Routine con **clave** (Key) **de 64 bits** (SAFER K-64) es un cifrado de bloque iterado con **bloques de 64 bits** diseñado por Massey en 1993 para *Cylink Corporation.*

Consiste en idénticas rondas seguidas de una transformación a la salida. La recomendación inicial de 6 rondas fue seguida de otra recomendación para modificar el tratamiento de la clave (Strengthened Key) dando lugar al **SAFER SK-64**, que debería ser usado en lugar del SAFER K-64 y usar 8 rondas.

SAFER consta de simples operaciones de byte, y así, recomendado para procesadores de pequeño tamaño de palabra como chipcards.

En conrtraste con cifradores del estilo del Feistel, en SAFER, las operaciones usadas para encriptar difieren de las de desencriptar.

SHARK es un cifrador diseñado por Rijmen (creador de Rijndael) que se puede considerar similar al SAFER pero con el que nos debemos mantener cautelosos.

II.3.1.13. Serpent

Fue diseñado por *Ross Anderson* (Reino unido), *Eli Biham* (Israel) y *Lars Knudsen* (Noruega) con un gran márgen de seguridad. Aunque tiene un escaso gasto de memoria RAM y ROM, es **muy lento**.

II.3.1.14. LOKI'91

Nueva versión del LOKI'89 diseñado por *Brown*, *Pieprzyk*, y *Seberry*, fue propuesto como sucesor del DES con una longitud de **clave mayor de 64 bits**, un tamaño de **bloque de 64** bits y **16 ciclos**.

Difiere del DES principalmente en el tratamiento de la clave y la función f.

Se han observado **debilidades ante análisis diferenciales**.

II.3.1.15. Khufu y Khafre

Son cifradores del estilo del DES que fueron propuestos como una alternativa al DES en software rápido. Usa **bloques de 64 bits** y un **número variable de ciclos** (típcamente 16, 24 ó 32). **Las claves** de Khufu pueden llegar **hasta 512 bits** y las de Khafre son múltiplos de 64 bits (típicamente 64 ó 128).

Bajo los mejores ataques conocidos, el Khufu de 16 ciclos y el Khafre de 24, son mucho más difíciles de descifrar que el DES.

II.3.1.16. SKIPJACK

Es un cifrador de bloque **clasificado** cuya especificación es mantenida por la National Security Agency (NSA) de los EE.UU. aunque su especificación está disponible en el FIPS 185. Tiene una **longitud de clave de 80 bits,** un tamaño de **bloque de 64 bits** y lleva a cabo **32 iteraciones**.

II.3.1.17. CAST-128 (rfc 2144), CAST-256 (rfc 2612)

CAST es una familia de cifradores de estilo DES con **tamaño de clave y número de ciclos variables** aunque con **tamaño de bloque de 128 bits**: CAST-128 tiene una longitud de clave de 128 bits y CAST-256 de 256 bits.

II.3.1.18. 3-WAY

Es un cifrador de bloque con **tamaño de clave y de bloque de 96 bits**. Fue diseñado por *Daemen* en aras a la velocidad tanto en hardware como en software y con capacidad para resistir ataques diferenciales y lineales.

II.3.1.19. Comparativa

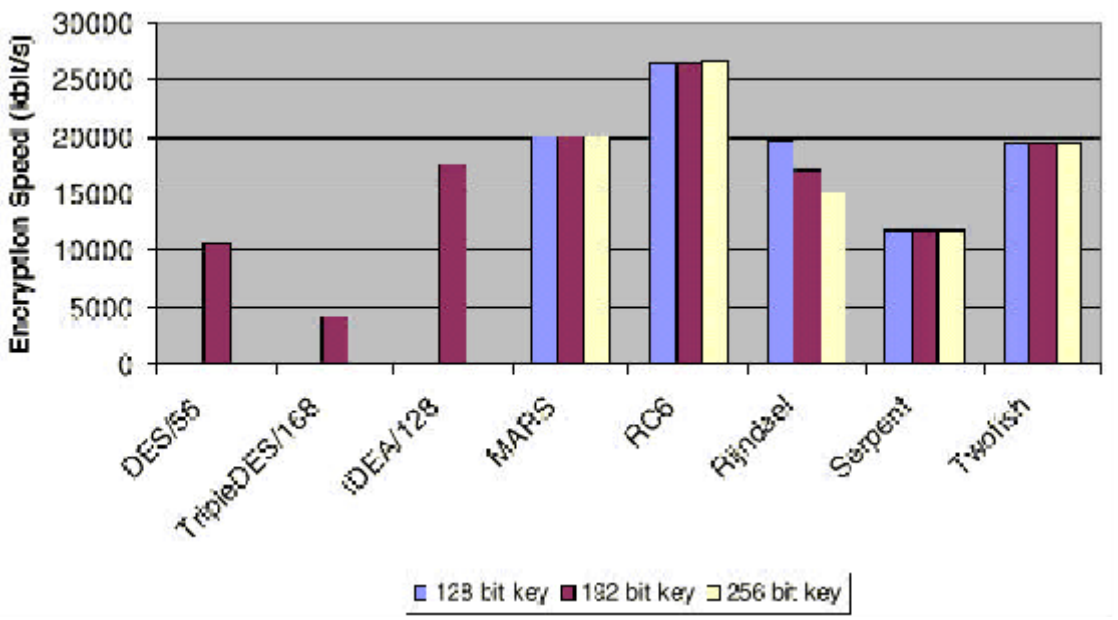


Ilustración 12: Comparativa de velocidad entre distintos algoritmos

II.3.2. Funciones de resumen, digestoras o hash

II.3.2.1. Definición

El vocablo inglés *hash* tiene dos acepciones principales: "picadillo" y "lío", y eso es a grandes rasgos lo que realiza: comprime un texto en un bloque de longitud fija pero de una forma muy particular que veremos a continuación. Estas funciones también se conocen como "digestoras" por una mala traducción de *digest* cuya correcta traducción es la de "resumen" y son funciones **de una dirección** (dispersión).

La función de resumen es muy **útil en autenticación y firma digital** (*Ilustración 11*) por los siguientes motivos:

- **No tener que encriptar todo el texto** ya que este proceso es lento con los algoritmos asimétricos. El resumen sirve para comprobar si la clave pública del emisor es auténtica y no es necesario encriptar todo el texto si no se requiere confidencialidad.
- **Para poder comprobar automáticamente la autenticidad**: si se encripta todo el texto, al desencriptar sólo se puede comprobar la autenticidad comprobando si el resultado es inteligible. Evidentemente este proceso debe realizarse de forma manual. Utilizando un resumen del texto, se puede comprobar si es auténtico comparando el resumen realizado en el receptor con el desencriptado.

- **Para comprobar la integridad del texto**, ya que si ha sido dañado durante la transmisión o en recepción no coincidirá el resumen del texto recibido con la desencriptación.

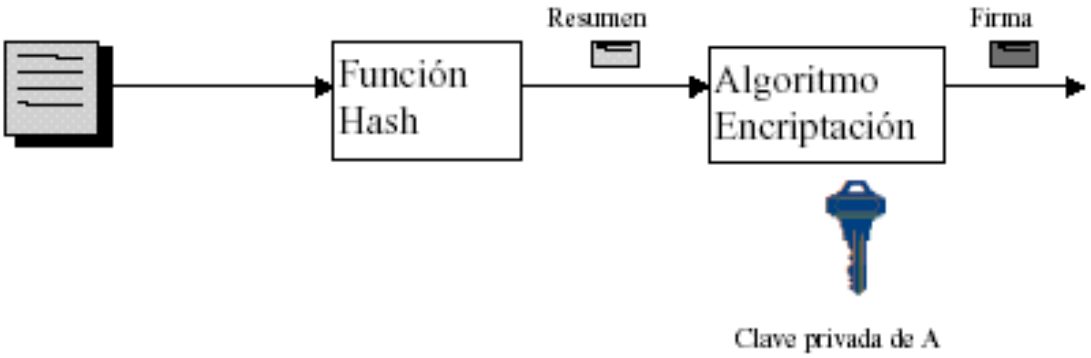


Ilustración 13: La función hash como parte del proceso de firmado.

- Las funciones hash **no encriptan**, sólo comprimen textos en un bloque de longitud fija. Son diferentes de las funciones clásicas de compresión de textos, como ZIP, Huffman, V-42, etc. ya que estas funciones son reversibles e intentan eliminar la redundancia de los textos manteniendo la información. Las funciones hash por el contrario, deben cumplir las siguientes condiciones:

- Transformar un texto de longitud variable en un bloque de **longitud fija**.

- Ser **irreversibles**, es decir, no se puede recuperar el texto desde el resumen.

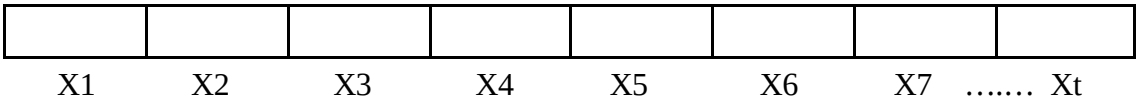
- **Resistencia a la preimagen:** significa que dada cualquier imagen y, es computacionalmente imposible encontrar un mensaje x tal que h(x)=y. Otra forma como se conoce esta propiedad es que h sea de un solo sentido.

- **Resistencia a una 2ª preimagen:** significa que dado x, es computacionalmente imposible encontrar una x' tal que h(x)=h(x'). Otra forma de conocer esta propiedad es que h sea **resistente a una colisión suave.**

- **Resistencia a colisión fuerte**: significa que es computacionalmente imposible encontrar dos diferentes mensajes x, x' tal que h(x)=h(x').

Su funcionamiento a grandes rasgos es el siguiente:

1. Tomamos como entrada una cadena de longitud arbitraria. Luego se divide este mensaje en pedazos iguales, por ejemplo de 160 bits. Como en general el mensaje original no será un múltiplo de 160, entonces se le agrega un relleno (por ejemplo de ceros (*zero padding*)) al último elemento para completar un número entero de trozos de 160 bits. Entonces el mensaje toma la forma X = X1, X2, X3,…,Xt donde cada Xi tiene igual longitud (160 bits por ejemplo).



2. Se asigna un Valor Inicial (VI) a H0:

```
H0 = VI
```

3. Se obtiene H1 como resultado de combinar H0 con X1 usando una función de compresión f:

```
H1 = f(H0,X1)
```

4. Se obtiene H2, combinando H1 y X2 con f:

```
H2 = f(H1,X2)
```

5. Se hace lo mismo para obtener H3:

```
H3 = f(H2,X3)
```

6. Hasta llegar a Ht:

```
Ht = f(Ht-1, Xt)
```

7. Entonces el valor hash será:

```
h(M) = Ht
```

La *Ilustración 14* puede presentarnos el funcionamiento de estas funciones de resumen o hash:

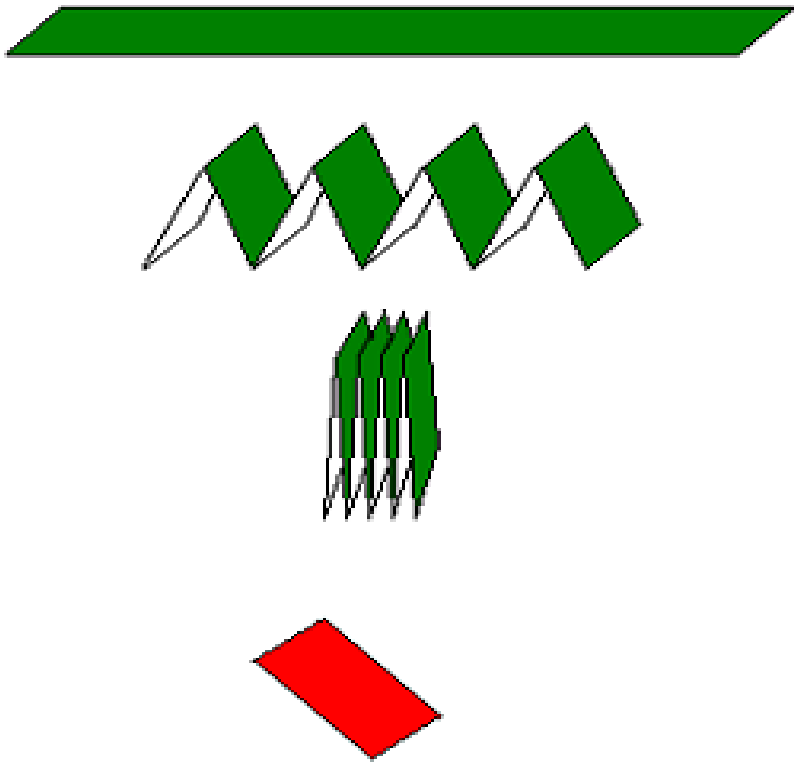


Ilustración 14: Abstracción del funcionamiento de las funciones de resumen.

Las funciones hash pueden operar como Modification Detection Codes (MDC) o como Hash Message Authentication Codes (HMAC).

Los MDC sirven para resolver el problema de la integridad de la información: al mensaje se le aplica un MDC (una función hash) y se manda el resultado junto con el propio mensaje; al recibirlo, el receptor aplica la función hash al mensaje y comprueba que sea igual al hash que se le adjunta (*Ilustración 15*).

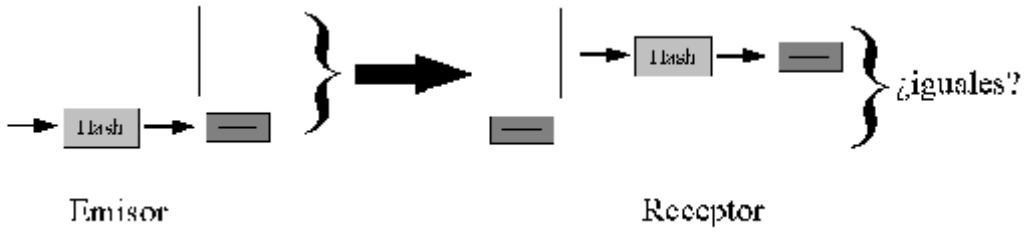


Ilustración 15: Ejemplo de MDC

Los HMAC (rfc 2104) **sirven para autenticar el origen de los mensajes además de la integridad**. Es decir, se combina el mensaje M con una clave privada K y se les aplica un hash h(M,K), si al llegar a su destino h(M, K) se comprueba la integridad de la clave privada K, entonces se demuestra que el origen es sólo el que tiene la misma clave K, probando así la autenticidad del origen del mensaje.

Las funciones hash proporcionan una excelente forma de esparcir el azar (entropía) de una entrada entre todos los bits de entrada de una función. Para generar un número "aleatorio", simplemente se puede tomar un gran número de fuentes de datos que aparentemente cambian a través del tiempo, como reloj de tiempo real, bitácoras y entradas de usuario y se pasa toda esa información a través de una función hash. Si hay más bits de entropía en el bloque de entrada que bits de salida de la función, se puede suponer que todos los bits de la salida son independientes y aleatorios, siempre y cuando la función de resumen sea segura.

Mediante un compendio de mensajes, es posible crear llaves de encriptación para códigos de encriptación simétrica. La llave de encriptación se produce calculando el resumen de una frase introducida. Por ejemplo, PGP utiliza esta técnica.

Las funciones hash más conocidas son las que se crean a partir de un block cipher como MD5, SHA-1 o RIPEMD-160. Actualmente se ha podido encontrar **debilidades en las funciones hash que tienen como salida una cadena de 128 bits**, por lo que se ha recomendado usar salidas de 160 bits. Así mismo se han encontrado ataques a MD5 y SHA-0 (antecesora de SHA-1), esto ha dado lugar que se dirija la atención sobre la función hash RIPEMD-160.

El ataque más popular por fuerza bruta a una función hash es conocido como "birthday attack" y se basa en la siguiente paradoja: si hay 23 personas en un local existe una probabilidad de al menos ½, de que existan dos personas con el mismo cumpleaños. Esto es debido a que en algunas funciones , cuando a una entrada aleatoria, devuelve una de entre k posibles salidas igualmente probables, se espera obtener la misma salida después de alrededor de la $1.2k^{1/2}$. Para el ejemplo anterior, sustituir k por 365 (http://www.ciphersbyritter.com/NEWS4/BIRTHDAY.HTM)

II.3.2.2. Message Digest #2 (MD2) (rfc 1319)

Desarrollada por *Rivest* en 1992, genera resúmenes de **128 bits**. Es **la más segura** de las funciones de compendio de *Rivest*, pero es la que lleva más tiempo calcular.

II.3.2.3. Message Digest #4 (MD5) (rfc 1186 y 1320)

Desarrollada por *Rivest* en 1992 como alternativa a MD2, aunque posteriormente se demostró que es **inseguro**: se pueden encontrar dos mensajes que produzcan el mismo resumen. Genera resúmenes de **128 bits**.

II.3.2.4. Message Digest #5 (MD5) (rfc 1321)

Desarrollada por *Rivest* en 1992 a partir de MD4, genera resúmenes de **128 bits** y es **de libre utilización**. En el verano de 1996, se le descubrieron algunas **fallas** que permiten calcular algunos tipos de colisiones.

II.3.2.5. Secure Hash Algorithm #0 (SHA-0 o SHA)

Deasarrollado por la agencia americana NSA para usarse en el DSS del NIST. Al igual que MD5, está basado en MD4 y fue diseñado para superar algunas de las debilidades potenciales de MD4. Produce unos compendios de **160 bits**. Poco después de la publicación del SHA, NIST declaró que **no era adecuado** emplearlo sin un pequeño cambio que dio lugar al SHA-1.

II.3.2.6. Secure Hash Algorithm #1 (SHA-1)

También desarrollado por la NSA en 1994 a partir de un pequeño cambio del SHA. Funciona con mensajes de hasta 2^{64} bits y genera un boletín de **160 bits**. SHA-1 está documentado en FIPS 180-1 (http://csrc.nist.gov/publications/fips/fips180-1/fipds180-1.pdf) o en ISO/IEC 10118-3.

II.3.2.7. Secure Hash Algorithm #2 (SHA-2)

Produce boletines de **256, 384 ó 512 bits**. El SHA-2 constituye el **Secure Hash Standard** (SHS) (http://csrc.nist.gov/cryptval/shs.html) del NIST. SHA-2 está documentado en FIPS 180-2

II.3.2.8. RIPEND-128, RIPEND-160

Son funciones de resumen europeas con tamaño de resumen de **128 y 160 bits** respectivamente. Existen extensiones de este algoritmo a un número mayor de bits.

Además de estas funciones, también es posible utilizar como funciones de resumen sistemas de encriptación simétrica de bloque, por ejemplo DES. Para utilizar una función de encriptación como función de resumen, basta con ejecutar la función de encriptación en modo de retroalimentación de código. Como llave se utiliza una llave elegida al azar y específica a la aplicación. Se encripta todo el archivo y el último bloque de datos es el resumen.

II.3.2.9. Utilización de claves

Para aplicaciones **únicamente de autenticación e integridad, no firma,** se puede añadir una clave simétrica a la generación del resumen. De esta manera **no es necesario encriptar**, ésta clave ya demuestra que el usuario es auténtico y el resumen propiamente demuestra la integridad del texto. El problema es utilizar una clave simétrica y, por lo tanto, se debe transmitir por un canal seguro. El sistema utilizado actualmente es el de **claves de sesión encriptadas mediante la clave privada** del emisor (ver capítulo de claves de sesión).

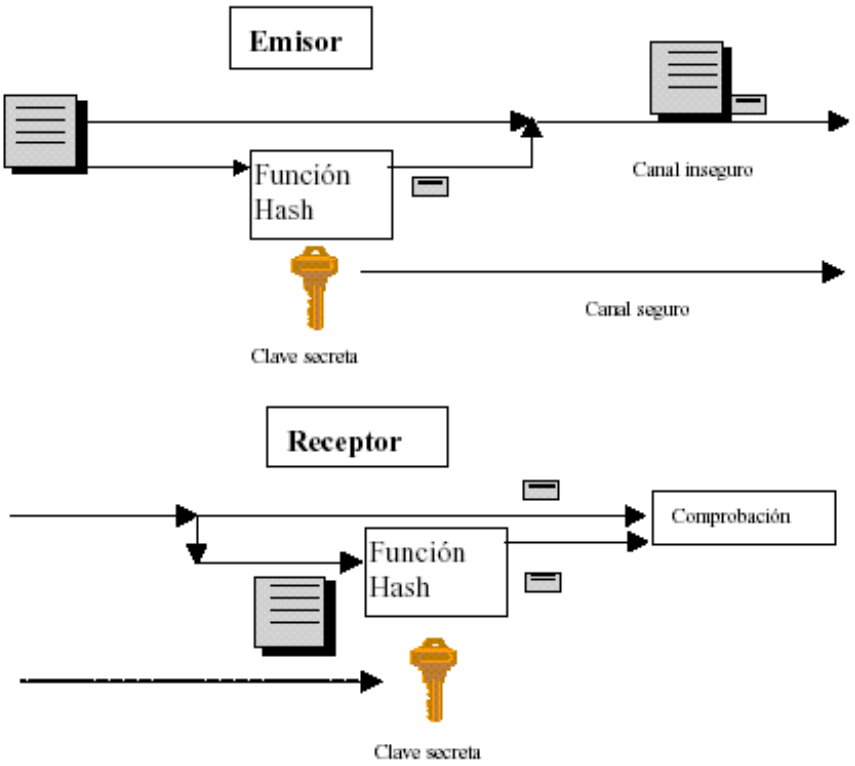


Ilustración 16: Utilización de claves

II.3.3. Criptografía asimétrica o de clave pública

En 1976 *Whitfield Diffie* y *Martin Hellman* publicaron el artículo "*New directions in cryptography*" en el que proponían un nuevo tipo de criptografía basado en utilizar claves distintas para encriptar y desencriptar, una de ellas se hace pública y la otra permanece privada para cada usuario. Así todos los usuarios de la red tienen acceso a las claves públicas, pero únicamente a su propia clave privada. Estas ideas supusieron una revolución en la criptología: se podía utilizar para confidencialidad (al igual que los sistemas simétricos),

junto con autenticación y firma digital. Por ende, **solucionamos el problema de la distribución de claves simétricas**.

Para cada tipo de servicio se encripta de manera diferente:

- **Confidencialidad**. El emisor encripta el texto con la clave pública del receptor y el receptor lo desencripta con su propia clave privada. Así cualquier persona puede enviar un mensaje encriptado, pero sólo el receptor, que tiene la clave privada, puede descifrar el contenido (*Ilustración 17*).

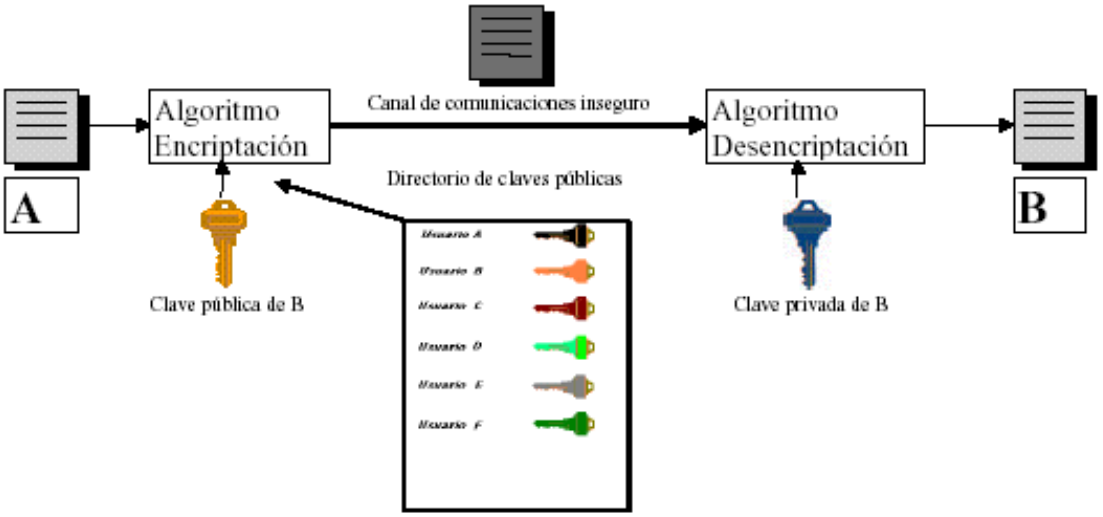


Ilustración 17: Confidencialidad en sistemas de clave asimétrica

- **Autenticación**. Se encripta el mensaje o un resúmen de éste mediante la clave privada del emisor y cualquier persona puede comprobar su procedencia utilizando la clave pública del emisor. El mensaje es auténtico porque sólo el emisor verdadero puede encriptar con su clave privada (*Ilustración 18*).

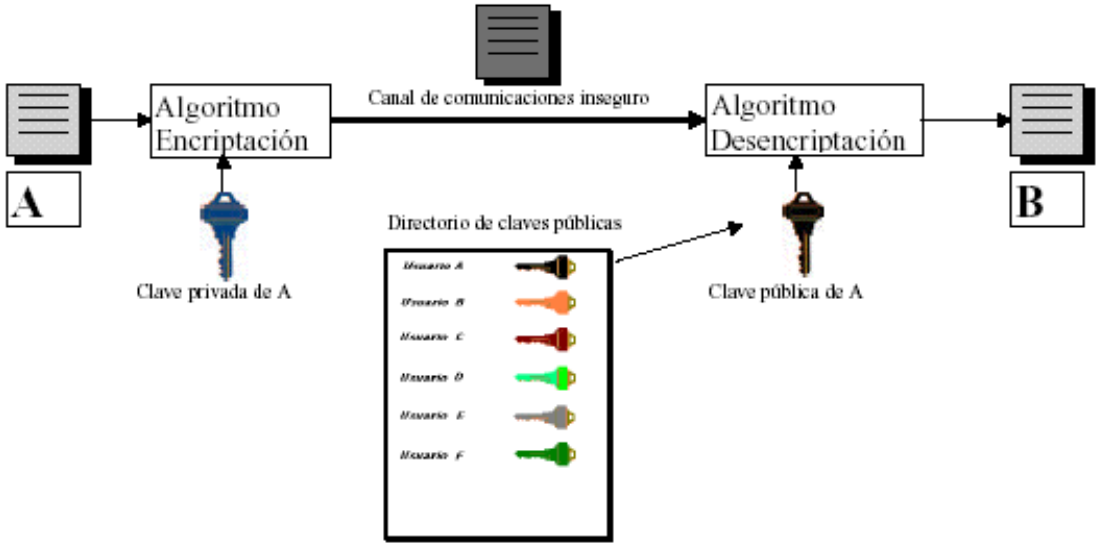


Ilustración 18: Autenticación en sistemas de clave asimétrica

- **Firma digital**. Igual que la autenticación pero siempre se encripta el resumen del mensaje, cuyo criptograma es la firma del emisor. Así, el emisor no puede negar la

procedencia ya que se ha encriptado con su clave privada. Por otro lado, el receptor no puede modificar el contenido porque el resumen sería diferente y no coincidiría con la desencriptación de la firma. Pero el receptor sí puede comprobar que el resumen coincide con la firma desencriptada para comprobar su autenticidad. La firma digital lleva implícita la autenticación (*Ilustración 19*).

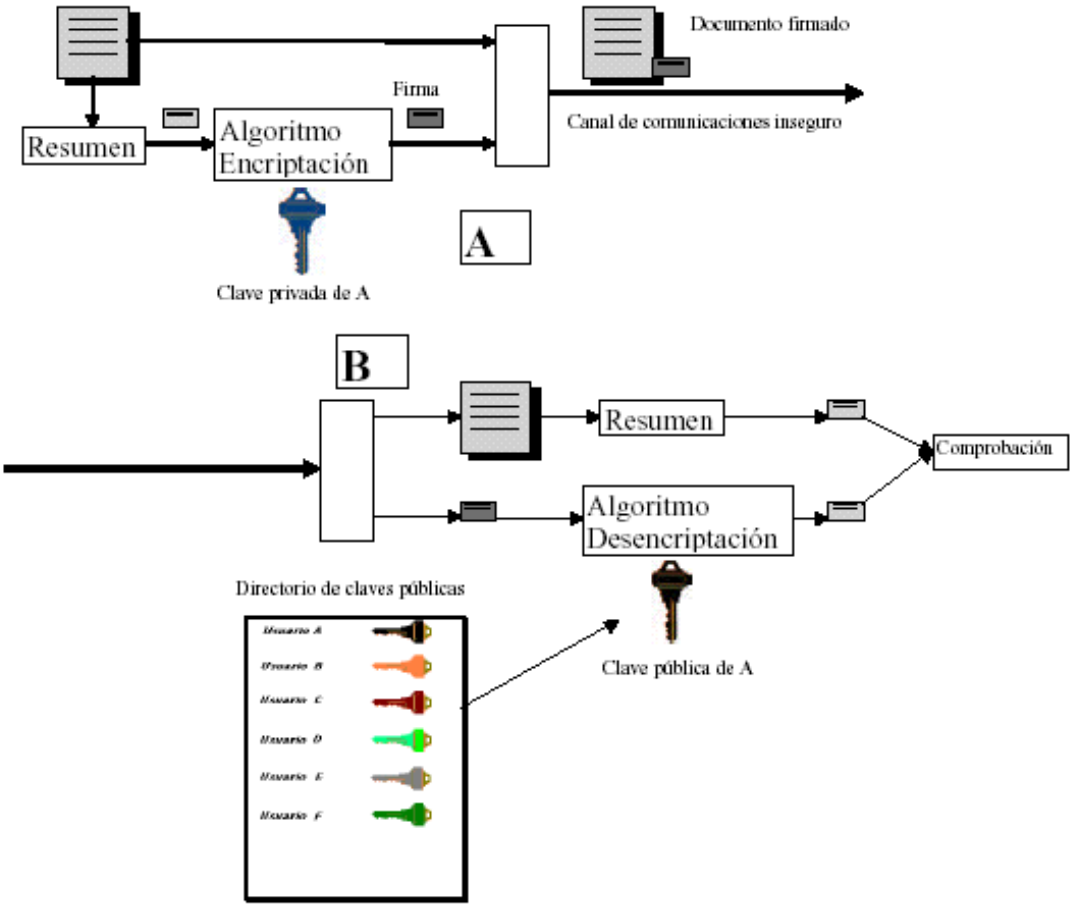


Ilustración 19: Firmado digital en sistemas de clave asimétrica

Los algoritmos asimétricos están basados en funciones matemáticas fáciles de resolver pero muy complicadas de realizar la inversa, por ejemplo, la potencia, el logaritmo y el módulo.Estas funciones son útiles para criptografía si la inversa es fácil de calcular conociendo un número concreto que es la clave, del par de claves que no se utilizó para encriptar.

Así, **la clave privada y pública están relacionadas matemáticamente**, pero esta relación debe ser suficientemente compleja para que el criptoanalista no la pueda encontrar. Debido a esto, las claves privadas y públicas no las elige el usuario sino que las calcula un algoritmo y normalmente son demasiado largas para poderlas memorizar.

Un algoritmo de clave pública debe cumplir:

- **Conocido el criptograma no se puede descifrar el texto ni adivinar la clave.**
- **Conocido el texto en claro y el criptograma es más caro** (en tiempo y/o dinero) **descifrar la clave que el valor de la información.**

- **Conocida la clave pública y el texto no se puede generar un criptograma encriptado con clave privada**.

En estos sistemas, también se utiliza el criptoanálisis de búsqueda de claves y se puede aplicar las mismas suposiciones que en algoritmos simétricos. Además de este método, también hay algoritmos matemáticos para obtener la clave privada desde la pública pero, si el algoritmo es bueno, éstos son más caros que el valor de la información. Para complicar estos sistemas de criptoanálisis, se utilizan claves muy largas (para mayor información, consultar el capítulo de criptoanálisis en algoritmos asimétricos).

La ventaja es que **implementan servicios de autenticación y firma**, y además **resuelven el problema de la distribución de claves**: la clave pública puede ser visible por cualquiera y la privada no se transmite nunca.

El inconveniente de estos sistemas es la **dificultad de implementación y la lentitud de proceso.**

En la actualidad, la criptografía asimétrica o de clave pública se divide en tres familias según el problema matemático en el cual basan su seguridad. La primera familia es la que basa su seguridad en el **Problema de Factorización Entera** (PFE), los sistemas que pertenecen a esta familia son, el sistema RSA y el de Rabin-Williams (RW). La segunda familia es la que basa su seguridad en el **Problema del Logaritmo Discreto** (PLD), a esta familia pertenece el sistema de Diffie-Hellman (DH) de intercambio de claves, ElGamal y el DSA. La tercera familia es la que basa su seguridad en el **Problema del Logaritmo Discreto Elíptico** (PLDE), en este caso hay varios esquemas tanto de intercambio de claves como de firma digital que existen como el DHE (Diffie Hellman Elíptico), DSAE, (Nyberg-Rueppel) NRE, (Menezes, Qu, Vanstone) MQV.

II.3.3.1. RSA (Rivest-Shamir-Adleman)

Es el más popular y utilizado de los algoritmos asimétricos. Fue inventado en 1978 por *Ronald ("Ron") Rivest*, *Adi Shamir* y *Len Adleman* cuyas iniciales dan nombre al algoritmo. Patentaron el algoritmo y cuando alcanzó popularidad fundaron una empresa, *RSA Data Security Inc.*

En el algoritmo RSA existen **dos claves: pública y privada**. La clave pública consta a su vez del llamado módulo (n) y del exponente público (e) que por lo general se toma igual a $65553=010001|_{16}$. La clave privada consta de dos números primos (p y q) que multiplicados nos dan el módulo y un exponente privado (d). El cálculo de estas claves se realiza en secreto en la máquina que poseerá la clave privada. El proceso es el siguiente:

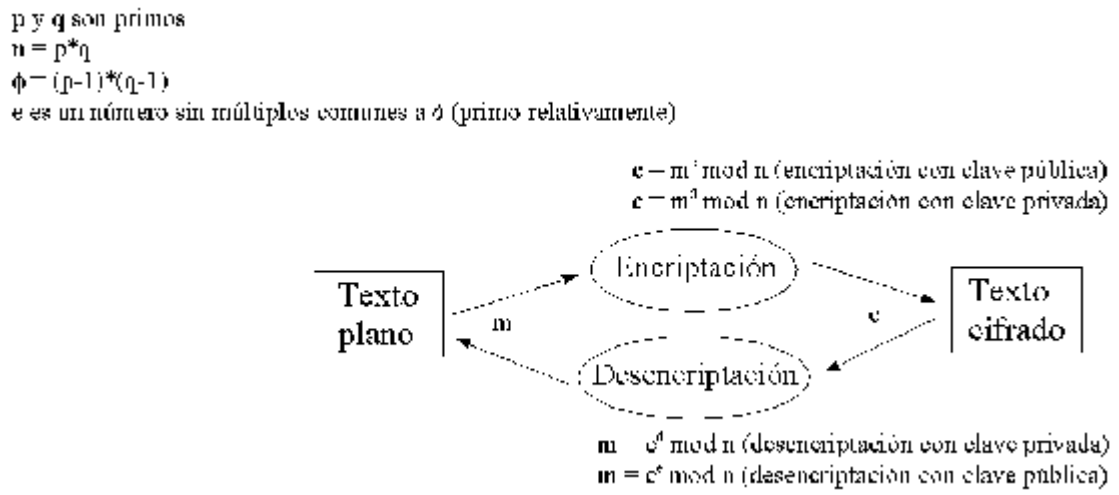


Ilustración 20: Algoritmo RSA

Lógicamente, si encriptamos con la clave pública, debemos desencriptar con la clave privada y viceversa.

La seguridad de RSA se basa en el Problema de Factorización Entera (PFE): aún no se ha descubierto ninguna forma analítica de descomponer números grandes (como por ejemplo n) en factores primos (p y q). Si usásemos el método de "prueba y error", tendríamos que recorrer muchas claves hasta calcular el exponente privado (d).

RSA está muy extendido como algoritmo asimétrico y puede ofrecer sus servicios exclusivos: distribución de claves, autenticación y firma digital, que sólo se pueden implementar con estos tipos de sistemas además de confidencialidad, aunque para esto último, se suelen utilizar los algoritmos de clave simétrica ya que estos son mucho más rápidos.

En RSA existen una serie de aspectos importantes a tener en cuenta:

1. **La longitud de las claves**. En la actualidad es recomendable usar claves a partir de 2048 bits (616 dígitos). La longitud de las claves tiene que ver directamente con la seguridad del sistema. Si el número n pudiese ser factorizado, entonces sin mucha dificultad se podría calcular a d a partir de e, p, y q por lo tanto descifrar cualquier mensaje.

2. **La aleatoriedad de las claves.** Se evitan muchos ataques si las claves son elegidas de forma aleatoria.

3. **Método de relleno.** El método que actualmente es usado para aplicaciones en el esquema de cifrado es el ***OAEP*** ya que resiste a los ataques que actualmente se conocen y viene recogido en el estándar PKCS#1 v.2 (particularmente es el método que usaremos en la aplicación Mastin). En el caso de esquemas de firma digital, el método de relleno recomendable es ***PSS,*** descrito en PKCS#1 v2.1

4. **Elección de parámetros.** La elección adecuada de los parámetros que se usan aumenta la seguridad del sistema así como su fácil y rápida implementación. Como la de elegir al exponente público $e = 65537 = 010001|_{16}$. Los números primos p y q además de ser generados aleatoriamente deben de tener la misma longitud y no estar "situados cerca".

II.3.3.2. Criptosistema ElGamal

ElGamal es un criptosistema de clave pública basado en el Problema del Logaritmo Discreto (PLD). Fue desarrollado por *Taher ElGamal* en 1984. No está patentado y es **gratuito**. Consiste de algoritmos tanto de encriptación como de firma. El algoritmo de encriptación es similar al protocolo de acuerdo de claves de Diffie-Hellman.

En la *Ilustración* podemos ver su funcionamiento, donde la clave pública se compone de g, p e y, y la clave privada es x:

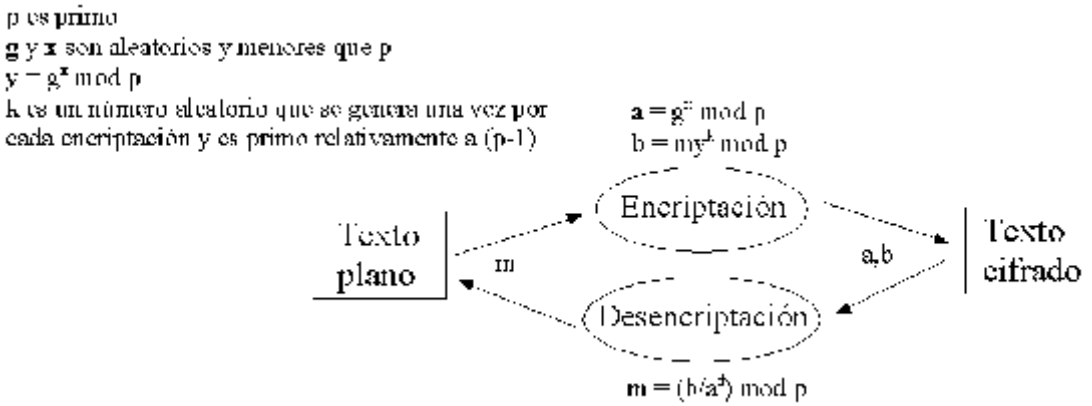


Ilustración 21: Algoritmo de ElGamal

El algoritmo de firma de ElGamal es similar al algoritmo de encriptación en tanto que la calve pública y privada tiene la misma forma, sin embargo, la encriptación no es lo mismo que la verificación de firmas, ni la desencriptación lo mismo que la creación de firmas. DSA está basado en parte por el algoritmo de firma de ElGamal.

La seguridad mostrada por ElGamal es similar a la de RSA a igualdad de longitud de clave pero presenta las siguientes desventajas:

1. Necesidad de **aleatoriedad**
2. **Menor velocidad** (especialmente para firmar)
3. **Expansión del mensaje** por un factor de dos, aunque esta expansión es insignificante si se usa sólo para el intercambio de claves.

II.3.3.3. DSS (Digital Signature Standard)

El DSS es un sistema de firma digital adoptado como estándar por el NIST y se encuentra documentado en FIPS 186-1 (http://csrc.nist.gov/fips/fips1861.pdf).

Utiliza la función Hash **SHA** y el algoritmo asimétrico **DSA** (**Digital Signature Algorithm**) que se utiliza para firma digital. Al igual que ocurre con el algoritmo de clave pública de ElGamal, su seguridad depende de la dificultad de calcular logaritmos separados. Utiliza más parámetros que el RSA, que son:

- ***KG*** (claves públicas de grupo): son comunes y públicas para un grupo de usuarios.
- ***KU*** (clave pública): se genera una por usuario a partir de las KG y es pública.

- ***KP*** (clave privada): es privada de cada usuario, se genera a partir de las anteriores.
- ***k*** (número aleatorio): se genera uno para cada firma.
- **s** y **r**: son dos palabras de 160 bits que forman la firma de un texto.

El número k permite que el mismo texto del mismo usuario no genere siempre la misma firma.

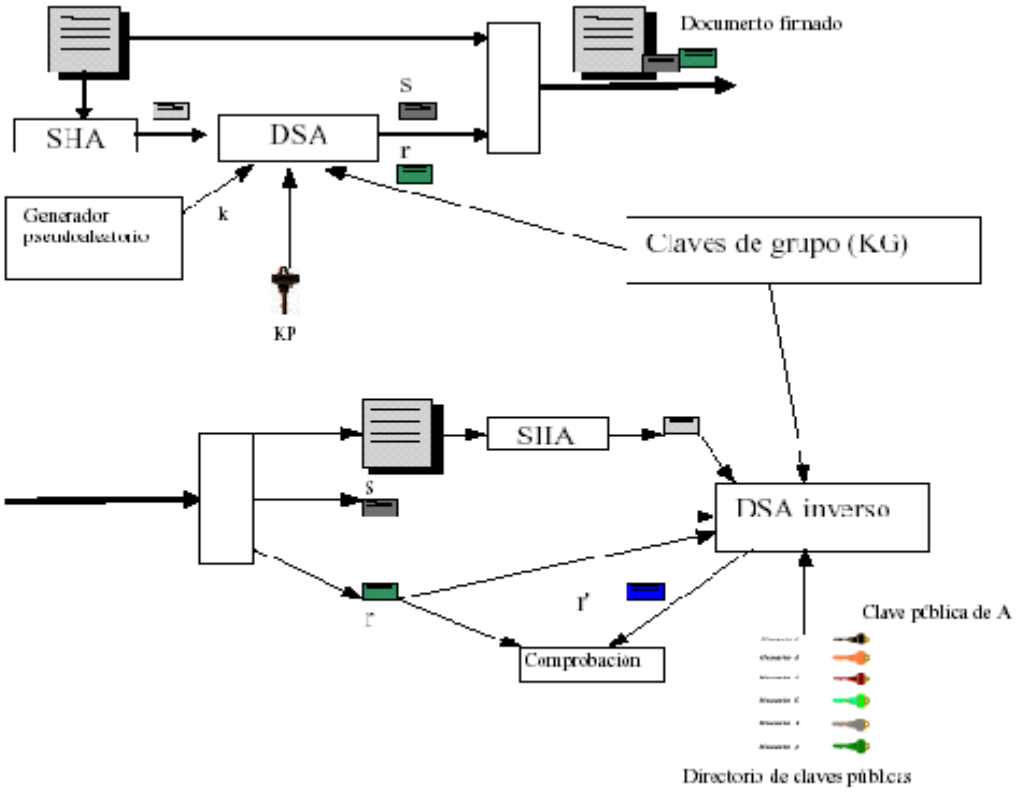


Ilustración 22: Esquema de firmado DSS

II.3.3.4. Criptografía de Curva Elíptica (CCE)

Las curvas elípticas son construcciones matemáticas que pueden ser definidos sobre cualquier campo (por ejemplo real, racional o complejo). Sin embargo, las usadas en criptografía son definidas principalmente en campos finitos. Su seguridad radica en el Problema del Logaritmo Discreto Elíptico (PLDE), a diferencia del RSA que se basa en el Problema de Factorización Entera (PFE).

La principal ventaja de este tipo de criptografía frente al RSA es la longitud de la clave secreta. Se puede demostrar que mientras en RSA se tiene que usar una clave de 1024 para ofrecer una considerable seguridad, los CCE solo usan 163 bits para ofrecer la misma seguridad, así también las claves RSA de 2048 bits son equivalentes en seguridad a 210 bits de CCE. Esto se debe a que para resolver el PLDE el único algoritmo conocido toma tiempo de ejecución exponencial, mientras que el algoritmo que resuelve PFE toma un tiempo menor.

Otra buena característica de CCE es que es posible construir una aritmética que optimice la rapidez y construir un circuito especial para esa aritmética (lo que se conoce como Base

Normal Optima). Lo que permite que los CCE sean idóneos para ser implementados en Smart Cards, teléfonos móviles, fax, PDAs, PCs, etcétera.

Los CCE son el mejor candidato para reemplazar a las aplicaciones que tienen implementado RSA, estas definen también esquemas de firma digital, intercambio de claves simétricas y otros.

II.3.3.5. Algoritmo de Diffie-Hellman

El algoritmo Diffie-Hellman fue el primer algoritmo asimétrico. Se describía en el famoso articulo "*New directions in Cryptography*" publicado en noviembre de 1976, se utilizaba para ilustrar un ejemplo de la criptografía que *Diffie* y *Hellman* acababan de descubrir la criptografía de clave pública.

Solamente se puede utilizar para **intercambiar claves simétricas**, pero ésta es una de las principales funciones de los algoritmos asimétricos, así está muy extendido en sistemas de Internet con confidencialidad de clave simétrica (VPNs, SSL, etc...). La seguridad del algoritmo depende de la dificultad del cálculo de un **logaritmo discreto**. Esta función es la inversa de la potencia discreta, o sea, de calcular una potencia y aplicar una función mod.

Potencia discreta: $Y = X_a \bmod q$

Logaritmo discreto: $X = Ind_{a,q}(Y)$

La generación de claves públicas es el siguiente:

- Se busca un número **grande y primo** llamado **q**.
- Se busca α **raiz primitiva de q**. Para ser raiz primitiva debe cumplir que: $\alpha \bmod q$, $\alpha_2 \bmod q$, $\alpha_3 \bmod q$, ... , $\alpha_{q-1} \bmod q$ son números diferentes.
- α y q son claves públicas.

Para compartir una clave simétrica se realiza el proceso indicado en la *Ilustración 23*:

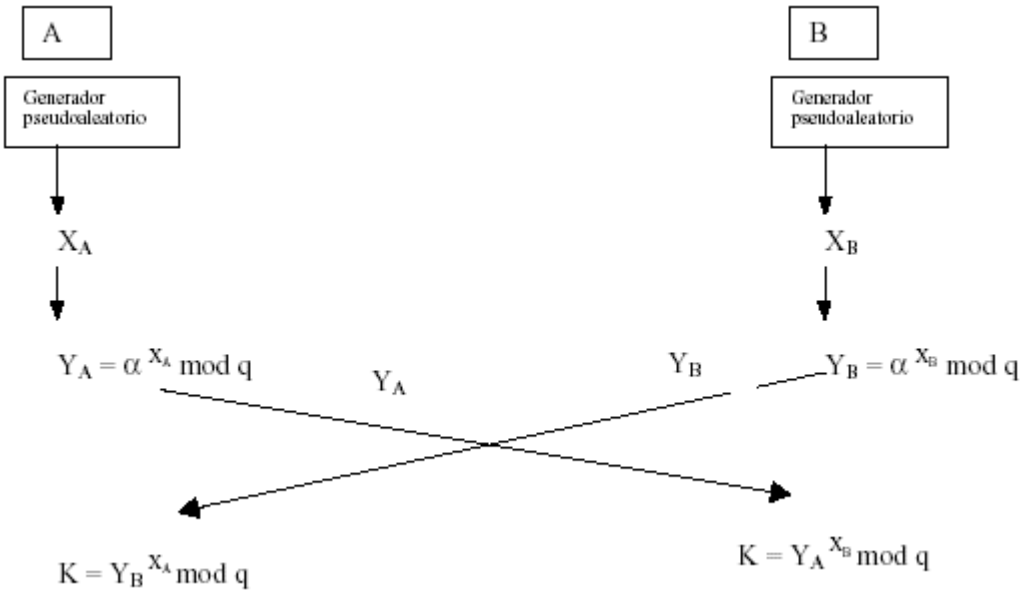


Ilustración 23: Algoritmo de intercambio de claves de Diffie-Hellman

Las K calculadas por los dos usuarios son iguales por la propiedad distributiva de la multiplicación, así:

$K = Y_B\ X_A \bmod q = (\alpha\ X_B \bmod q)\ X_A \bmod q = \alpha\ X_B\ X_A \bmod q = \alpha\ X_A\ X_B \bmod q = (\alpha\ X_A \bmod q)\ X_B \bmod q = Y_A X_B \bmod q = K$

Los criptoanalistas sólo disponen de las Y_i, q y α. Por lo tanto necesitan conocer alguna de las dos X_i, para esto deben realizar el logaritmo discreto $Ind_{\alpha,q}(Y_i)$ y esta operación no tiene una solución analítica para números grandes.

En un sistema con múltiples usuarios que quieren compartir claves simétricas uno a uno, se publican todas las Y_i en un directorio accesible. Cuando se quiere enviar un mensaje encriptado con otro usuario se realiza el proceso:

1. El emisor coge del directorio la Y_R del receptor.
2. El emisor calcula la clave K con su número secreto X_E.
3. Se envía el mensaje encriptado con K.
4. El receptor, para calcular K, utiliza su número secreto X_R y coge del directorio la Y_E del emisor.

II.3.4. Claves de sesión

Utilizar siempre la misma clave para muchas transmisiones tiene dos problemas:

- Cuanto más criptograma de la misma clave se tiene más fácil es romper un sistema.

- Si un criptoanalista descubre la clave podrá descifrar todas las transmisiones sin que los implicados sean conscientes.

Por lo tanto **es aconsejable cambiar de clave a menudo**. Se llaman claves de sesión a las claves utilizadas durante una única sesión.

En los **sistemas asimétricos no representa ningún problema cambiar de clave**, porque la distribución de claves públicas es abierta. Además, en autenticación se encripta un resumen, por lo tanto el criptograma nunca es muy grande y, si no realizan confidencialidad, no necesitan cambiar tan a menudo.

En los **sistemas simétricos la distribución de claves es un problema**, pero si estas se deben cambiar frecuentemente, el problema es mucho mayor. Antes de la invención de los sistemas asimétricos se utilizaban **centros distribuidores (KDC) y jerarquías de claves**. Pero este problema se ha simplificado con los sistemas asimétricos.

La mayoría de sistemas actuales utilizan técnicas mixtas de clave simétrica para confidencialidad y clave asimétrica para autenticación o firma y para distribuir las claves de sesión simétricas. Así el proceso puede ser el siguiente (*Ilustración 24*):

1. El ordenador emisor genera una clave de sesión aleatoria.
2. Se encripta el mensaje con la clave de sesión.
3. Se envía el mensaje encriptado y la clave de sesión encriptada con la clave pública del receptor.
4. El receptor desencripta la clave de sesión y, con ella, desencripta el mensaje.

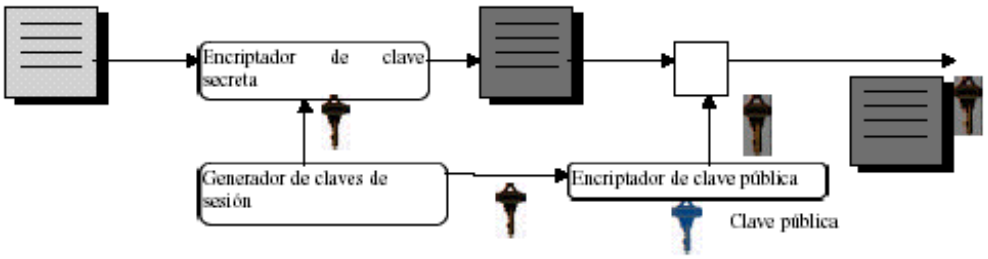


Ilustración 24: Claves de sesión

Si la comunicación se realiza entre dos usuarios que **no tienen las claves públicas** del otro, como en el caso de comunicación de **usuario anónimo** con una **Web pública**, se envía una clave pública en claro al comenzar la conexión ya que no hay peligro si la ven personas externas. Ejemplos de este sistema son los protocolos de Internet: SSL, SET, HTTPS,.... Así el proceso es el siguiente:

1. El cliente se conecta a un servidor de Internet seguro.
2. El servidor de Internet envía su clave pública al cliente.
3. El cliente encripta una clave de sesión aleatoria con la clave pública del servidor y la envía.
4. Todas las comunicaciones se realizan encriptadas con la clave de sesión.

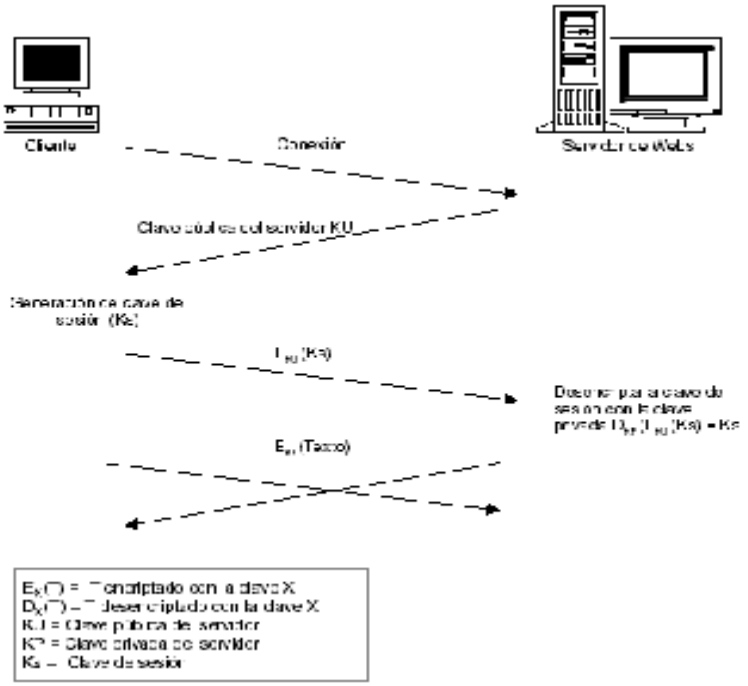


Ilustración 25: Claves de sesión si no se conocen las claves públicas

Otra forma de distribuir claves de sesión sin necesidad de utilizar el RSA es el algoritmo de distribución de claves de Diffie-Hellman.

II.3.5. Clave simétrica vs. clave asimétrica

Los sistemas de encriptación basados en clave simétrica o en clave asimétrica tienen distintas ventajas e inconvenientes. Resaltaremos y resumiremos las características mencionadas con anterioridad.

II.3.5.1. Ventajas de la criptografía de clave simétrica

1. Los cifradores pueden ser diseñados para tener una alta tasa de salida de datos. Esta tasa es muy elevada si utilizamos hardware específico.
2. Las claves son mucho más cortas.
3. Más facilidad de implementación.
4. Los cifradores pueden ser empleados como primitivas para construir diferentes dispositivos criptográficos como generadores de números pseudoaleatorios.
5. La criptografía de clave simétrica cuenta con una dilatada historia aunque realmente no obtuvo todo su potencial hasta el desarrollo de la computadora. Además los criptoanalístas la han estudiado con mayor profundidad.

II.3.5.2. Desventajas de la criptografía de clave simétrica

1. Las claves deben permanecer secretas en ambas partes de la comunicación.
2. En una red con gran cantidad de usuarios, se deben manejar una gran cantidad de claves. Además, para gestionar las claves de forma efectiva, requieren el uso de "terceras partes confiables" (Trusted Third Parts (TTP)).
3. La clave debería cambiar frecuentemente, e incluso, en cada sesión.

II.3.5.3. Ventajas de la criptografía de clave asimétrica

1. Sólo la clave privada debe mantenerse secreta aunque la autenticidad de las claves públicas debe ser garantizada, con la utilización de una infraestructura de clave pública, por ejemplo.
2. Dependiedo del modo de uso, un par de claves pública / privada pueden permanecer sin cambiarse durante largos periodos de tiempo. Por ejemplo si sólo usamos este tipo de encriptación para el intercambio de claves simétricas.
3. En una red con gran cantidad de usuarios, no es necesario almacenar ninguna clave aparte de nuestra clave privada. La clave pública del receptor puede ser obtenida a partir de, por ejemplo, una autoridad de certificación.

II.3.5.4. Desventajas de la criptografía de clave asimétrica

1. Las tasas de salida de datos para la mayoría de los algoritmos de clave asimétrica son varios órdenes de magnitud menores que para los algoritmos de claves simétricos.
2. El tamaño de las claves es normalmente mucho mayor que aquel requerido para la encriptación en clave simétrica.
3. Mayor dificultad de implementación.
4. La seguridad de los algoritmos de clave asimétrica radica en la presumible dificultad de un conjunto de problemas "numéricos".
5. La criptografía asimétrica no tiene la "experiencia" con que cuenta la criptografía simétrica, ya que no fue descubierta hasta mediados de los setenta.

II.3.5.5. Conclusión

No se puede hablar de que un tipo de encriptación es superior al otro. Ambos tipos se complementan y benefician simbióticamente: **usaremos la encriptación con clave simétrica para encriptar el mensaje y la encriptación con clave pública para el intercambio de claves**. Con ello conseguiremos las siguientes características:

1. Podremos cambiar de clave simétrica cuantas veces deseemos, incluso más de una vez por sesión, para aumentar la seguridad del esquema.
2. El intercambio de claves se realizará de forma totalmente segura siempre que contemos con una infraestructura de clave pública que garantice la autenticidad de la parte contraria.
3. La encriptación se realizará con el algoritmo simétrico de tal manera que se gana en velocidad.
4. No necesitaremos tener una base de contraseñas secretas de todas las partes contrarias.
5. Podremos mantener el mismo par de claves asimétricas durante largos periodos de tiempo (aunque nada nos impide cambiarlas) ya que se utilizan para encriptar mensajes muy cortos que son las claves simétricas.
6. Debido a que usamos criptografía asimétrica, podemos garantizar autenticación y firma digital.

III. PKI

Una Public Key Infraestructure (PKI) es una infraestructura de tipo jerárquico en la que la confianza en la identidad de una entidad se basa en la confianza en la entidad superior que la certifica. Por ello, se establecen una serie de mecanismos para la generación, distribución y revocación de las claves públicas de algoritmos asimétricos mediante los certificados digitales a través de las autoridades de certificación.

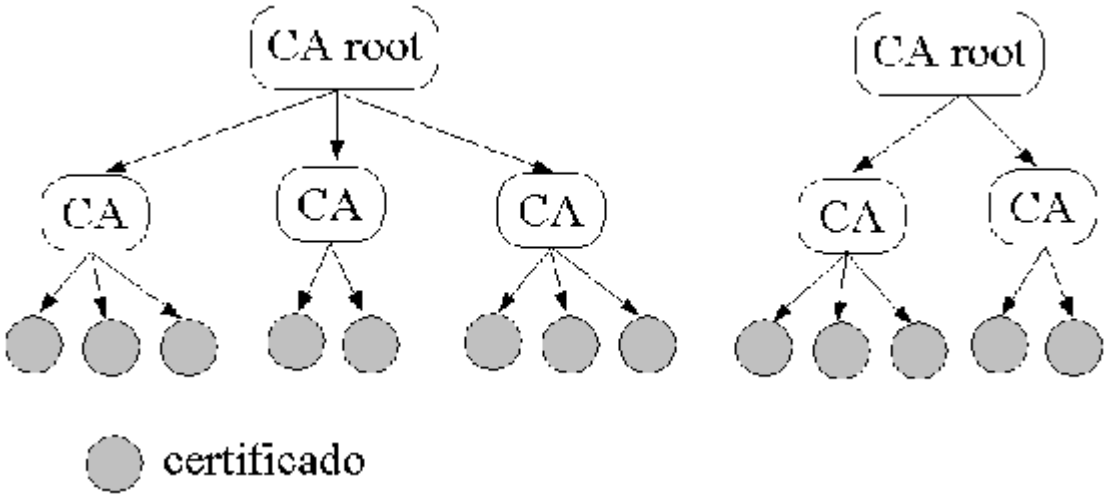


Ilustración 26: PKI

La PKI está compuesta por:

• Certificados de clave pública

• La Autoridad de Certificación (CA)

• La Autoridad de Registro (RA)

• El servicio de directorio

III.1. Certificados de clave pública

Como se ha comentado anteriormente, para solucionar el problema de la transmisión de las claves simétricas, se pueden encriptar con algoritmos de clave asimétrica. Esto es, se coge la clave simétrica y se encripta con la clave pública del destinatario de la encriptación simétrica. Se transmite y el destinatario lo desencripta con su clave privada, obteniendo de esta forma la clave de una manera segura. Ahora surge otro problema que es: ¿cómo se sabe que dicha clave pública pertenece realmente al destinatario que pretendemos y no a un intruso que suplante la personalidad del otro?

La solución son los certificados de clave pública. En los certificados de clave pública, por lo general, podemos encontrar los siguientes datos:

- El nombre del usuario.
- La clave pública de ese usuario.
- Datos e informaciones generales.
- La firma de una tercera persona de confianza.

Podemos definir un certificado (también conocido como certificado de clave pública) como una declaración firmada digitalmente por el emisor (issuer) diciendo que la clave pública (y alguna otra información) de un sujeto (subject) tiene un valor determinado.

Así la firma de esta tercera persona asegura que la clave pública pertenece al usuario. Toda la confianza se basa en la autenticidad de la firma y, por lo tanto, de la clave pública de la tercera persona. Aceptar o rechazar una clave pública depende de la firma que la avala en el certificado.

Con los certificados, el problema de la suplantación de personalidad se ha trasladado de la recepción de claves públicas a la confianza en las claves de terceras personas. Para resolver este problema los métodos más utilizados son:

- Niveles de confianza del PGP.
- Autoridades de certificación.

III.1.1. Ciclo de vida

Se podría decir que el ciclo de vida de un certificado empieza en la generación de un par de claves (pública y privada) para el algoritmo asimétrico, en este caso RSA (PKCS#1). Para obtener un certificado de una CA es necesario mandarle la Petición de Firma de Certificado (CSR) según el estándar PKCS#10 y en formato PEM que contiene entre otras cosas la clave pública del sujeto y sus datos en el DistinguishedName (DN). La CA (o en su caso la RA) tiene la obligación de cerciorarse de la veracidad de los datos contenidos en la CSR que va a firmar, puesto que la confianza depositada en él depende de ello. Una vez hecho, la CA remite de alguna forma el certificado en sí en alguno de los estándares (X.509 o PKCS#7) y formatos (DER o PEM) disponibles.

En caso de que la clave privada se vea comprometida o por cualquier otra cuestión, el usuario podrá pedir a la CA la revocación de su certificado y a partir de ahí el certificado se encontrará revocado y aparecerá en la próxima actualización de las Listas de Certificados Revocados (CRL).

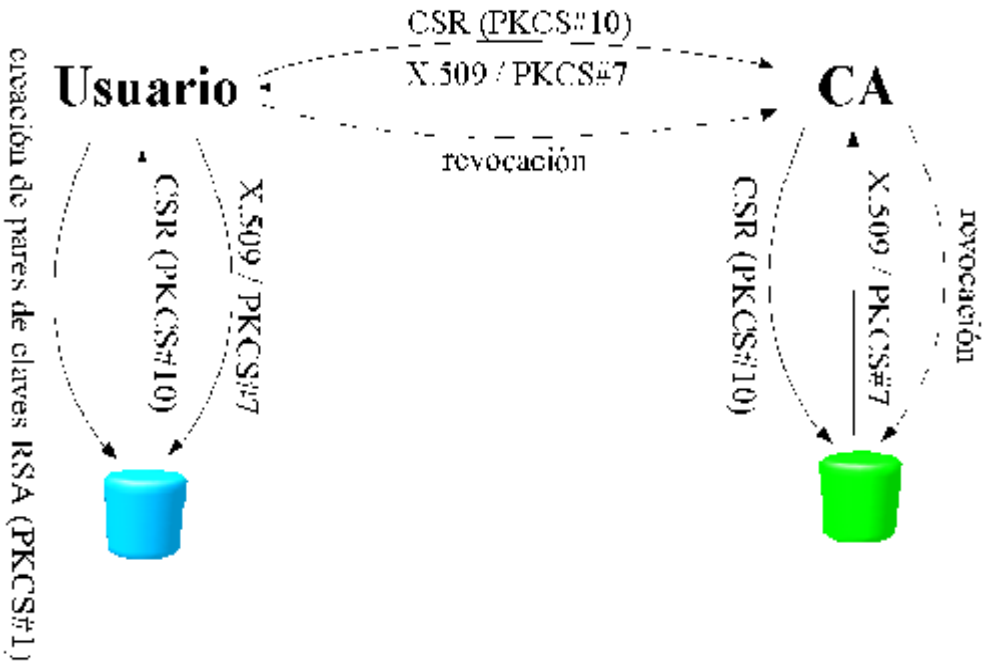


Ilustración 27: Ciclo de vida de un certificado

Normalmente, en los CA comerciales, existe un periodo de tiempo entre la revocación y la inclusión de dicho certificado en su CRL en la cual su uso sería "incorrecto" y cualquier problema estaría cubierto por un seguro que paga la CA.

En la CA que implementa este Proyecto, la aparición en la CRL es inmediata a la revocación, no dando lugar a periodo temporal en que encontrándose revocado, la CA dé por válido un certificado firmado por ella y por lo tanto no sería necesario el uso de un seguro.

III.1.2. ITU-T X.509

El protocolo X.509 de la ITU-T es el sistema de certificados de clave pública más utilizado. Su origen es el directorio X.500 desarrollado para dar servicio al correo electrónico X.400. Actualmente se utiliza en los protocolos seguros y en los sistemas de correo Internet más conocidos, excepto el PGP. Permite trabajar con CA y anidar certificados para crear estructuras jerárquicas.

El contenido de los certificados X.509 es el siguiente:

Campo	*Descripción*
Versión (Version)	La versión de protocolo X.509.
Número de serie (SerialNumber)	Identificador único del certificado, asignado por el CA. Este número se utiliza a la hora de construir la Lista de Certificados Revocados (CRL).
Algoritmo de la firma del certificado (Signature)	Identifica el algoritmo usado por la CA para firmar el certificado.
Autoridad de certificación (Issuer)	X.500 DistinguishedName (DN) de la CA. El uso de este certificado implica la confianza en la entidad que firma este certificado. En algunos casos, las CA raiz firman su propio certificado.
Fechas de inicio y final (Validity)	El certificado sólo tiene validez entre estas dos fechas. Es conveniente no permitir un período de validez largo y así obligar a renovar certificados y claves con asiduidad.
Usuario (Subject)	X.500 DistinguishedName (DN) de la entidad cuya clave pública identifica el certificado.
Clave pública (SubjectPublicKeyInfo)	La clave pública del usuario junto al nombre del algoritmo, permite múltiples longitudes.
Identificador único de la CA (IssuerUniqueID)	Las CA tienen un número de identificación único en el mundo.
Identificador único del usuario (SubjectUniqueID)	Los usuarios tienen un identificador único en la CA para todos sus certificados.
Extensiones (Extensions)	Posibles extensiones de la información. A partir de X.509 v3
Firma de la CA (Signature)	El CA firma con su clave privada todos los campos anteriores.

Tabla 4: Contenido de los certificados X.509

Versión	*Comentario*
X.509 v1	Ha estado disponible desde 1988, está ampliamente usada y es la más genérica.
X.509 v2	Introdujo el concepto de los identificadores únicos para el sujeto y el emisor para manejar la posibilidad de reusar dichos nombres todo el tiempo. La mayoría de la documentación desaconseja la reutilización de los nombres. por lo que esta versión no se suele usar.

Versión	*Comentario*
X.509 v3	Es la más reciente (1996) e introduce las extensiones, donde cada cual puede definir una extensión e incluirla en el certificado. Algunas extensiones comunes que se utilizan hoy en día son: KeyUsage (limita el uso de claves para un uso particular) y AlternativeNames (que permiten que se asocien otras identidades con su clave pública como nombres DNS, direcciones de email e IP).

Tabla 5: Versiones de los certificados X.509

Se pueden codificar tanto en **codificación DER como PEM**.

III.1.2.1. X.500 DistinguishedNames (DN)

Se usan para identificar entidades como el nombre del sujeto o del emisor de los certificados X.509. Está compuesto por varias partes y no hay ninguna norma que diga cuales son las que se deben usar o no. Por ejemplo, keytool soporta las siguientes:

Parte	*Descripción*
CommonName (CN)	Nombre común de la entidad. En el caso de certificados SSL, debe ser el nombre de la máquina a la que se accede, por ejemplo 192.168.0.3, us.es o www.us.es (siendo distintos para el navegador)
OrganizationUnit (OU)	Nombre del departamento o división de la organización
OrganizationName (O)	Nombre de la organización
LocalityName (L)	Nombre de la localidad
StateName (S)	Nombre del estado (EE.UU.) o pais
Country Code (C)	Código de dos letras del pais

Tabla 6: Partes de un X.500 DistinguishedName (DN) que soporta keytool

Un ejemplo de DN es la cadena:

```
“CN=192.168.0.3, OU=Departamento de Telematica, O=Universidad de Sevilla,
L=Sevilla, S=Spain, C=ES”
```

III.1.3. PKCS#7

Estándar desarrollado por *RSA Laboratories* que define una sintaxis general para mensajes que incluyen mejoras critptográficas tales como firmas digitales y encriptación. Surgió como ampliación del Privacy-Enhanced Mail (PEM) y ha sido la base de la especificación de correo electrónico seguro S/MIME. Sin embargo, sus aplicaciones no se limitan al correo, PKCS#7 se ha convertido en la base de la seguridad de mensajes en sistemas tan diversos como la especificación Secure Electronic Transaction (SET) para pagos con la tarjeta de crédito o el estándar de intercambio de información personal PKCS #12.

Se pueden codificar tanto en **codificación DER como PEM**.

Ver Anexo A para mayor información acerca de los estándares PKCS.

III.1.4. Codificación

III.1.4.1. DER

Las Distinguished Encoding Rules (DER) son un formato **binario** de codificación de cualquier objeto ASN.1 (Abstract Sintax Notation 1). Sin embargo tienen el problema de que no se pueden transportar por correo electrónico, de ahí que surgiera el RFC 1241.

III.1.4.2. RFC 1421

Tambien conocido como "**codificación Base64**" o como **PEM**. Es el estandar de codificación **imprimible** de certificados en Internet en sustitución de la codificación binaria (DER). Facilita la exportación de certificados, CRL's o CSR's a otras aplicaciones por email o mediante algún otro mecanismo.

III.2. Niveles de confianza del PGP

El sistema sería ideal si todos los certificados llegaran firmados por una persona a la que se ha comprobado la clave pública, pero no siempre es así. Además, una clave sin certificado sólo es de confianza si se transmite personalmente o mediante un medio de comunicación público, revistas, periódicos, webs que no son vías seguras.

En PGP se asigna dos niveles de confianza a cada clave pública de la base de datos. Estos son:

- **Confianza propia**. La confianza en clave pública del usuario calculada según el

  procedimiento por donde ha venido:

  - **Directamente**. Confianza máxima.
  - **Por certificado**. Depende de la firma de la tercera persona.

- **Confianza para firmar certificados**. Una clave pública puede tener una confianza propia muy alta porque ha llegada por un sistema seguro, pero puede ser que no se pueda confiar en las firmas de certificados de este usuario, porque firme sin confirmar su procedencia.

III.3. Autoridades de certificación (CA)

El sistema de PGP sirve para grupos pequeños de usuarios donde siempre hay un enlace entre ellos, aunque sea por una cadena de confianzas de muchas personas. Pero tiene dos inconvenientes:

- No es útil para los millones de usuarios de Internet ya que no pueden certificarse todos entre sí.
- No es útil para sistemas judiciales. Si se tiene que comprobar la procedencia de una firma y, por lo tanto, de la clave pública, con PGP se han de seguir largas cadenas de usuarios.

Para solucionar estos problemas se han creado las Autoridades de Certificación (CA).

Las tareas de la CA son, entre otras:

- La **generación** de certificados. La CA garantiza la identidad del propietario del certificado mediante la inclusión de su firma digital .
- La **revocación** de certificados en caso de que se haya producido algún percance, como la pérdida o compromiso de la clave privada.
- Dependiendo de lo que se indique en la política de certificación, es posible que además la CA genere los pares de clave para cada usuario. No ocurrirá en nuestro caso donde la generación de las claves así como el secreto de la clave privada corren a cuenta del usuario.
- Otras funciones como la publicación y **distribución de certificados** y **listas de revocado**s.

III.3.1. Autoridad de Registro (RA)

La RA es una entidad intermedia entre la CA y el usuario, cuya principal función consiste en reducir la carga que puede suponer para la CA comprobar la identidad de todos los usuarios, así como las peticiones de certificación. Dependiendo de la entidad e importancia de la CA, la RA puede existir o no, y no solo una sino varias que colaborarían con la CA en estas tareas.

En nuestro caso, no serán necesarias ya que, en caso de que sea un servicio público, tan sólo certificaremos la dirección de correo electrónico para la certificación personal y nos eximiremos de mayores responsabilidades en la certificación de servidores. Si por el contrario, el servicio fuese para una comunidad restringida de usuarios, sería el propio personal quien certificaría el sujeto de las peticiones.

III.3.2. El servicio de directorio

Este componente de la PKI debe almacenar los certificados de todos los usuarios para que cualquiera que lo desee pueda acceder y descargar el certificado de otro usuario. Además debe servir para almacenar las listas de revocados, así como cualquier otra estructura de información en la que se incluya el número de serie de los certificados revocados, y que también se firma digitalmente por parte de la CA para garantizar su integridad y autenticidad.

Por lo general, se suele optar por una estructura de directorios basada en **LDAP**, por su amplia utilización y porque cada vez son más las aplicaciones y APIs que permiten hacer uso de este tipo de directorios. En nuestro caso se va a disponer de una **base de datos de tipo texto** que hará las veces de directorio debido a que no certificaremos el contenido de subject en el certificado.

III.3.3. Ejemplos de CA's

En este apartado se estudiarán ejemplos de CA's. Primeramente se estudiará a la CA más popular, perteneciente a la empresa VeriSign y posteriormente se verán otras CA's comerciales mencionando características comunes observadas.

III.3.3.1. VeriSign

III.3.3.1.1. Descripción

Empresa surgida de RSA. Se accede a su CA a través de la página web http://www.verisign.com.

Si pulsamos en *DIGITAL CERTIFICATES & SSL*, llegamos a la página web http://www.verisign.com/products/site donde podremos ver la lista de productos de VeriSign en materia de certificados digitales y SSL. Se puede además consultar la lista de precios (noviembre 2002):

Producto	*Validez*	*Localización*	*Precio*	*Adicional*	*Renovación*
Secure Site	1 año	EE.UU.	$349	$249	$249
	2 años	EE.UU.	$598	$498	$498
	1 año	Internacional	$449	$349	$349
	2 años	Internacional	$798	$698	$698
Secure Site Pro	1 año	EE.UU.	$895	$895	$895
	2 años	EE.UU.	$1595	$1595	$1595
	1 año	Internacional	$895	$895	$895
	2 años	Internacional	$1595	N/A	$1595

Producto	*Validez*	*Localización*	*Precio*	*Adicional*	*Renovación*
Commerce Site	1 año 2 años 1 año 2 años	EE.UU. EE.UU. Internacional Internacional	$895 $1744 N/A N/A	$895 $1644 N/A N/A	$895 $1644 N/A N/A
Commerce Site Pro	1 año 2 años 1 año 2 años	EE.UU. EE.UU. Internacional Internacional	$1395 $2695 N/A N/A	$1495 $2695 N/A N/A	$1495 $2695 N/A N/A

Tabla 7: Productos y precios de Verisign (noviembre 2002)

Estos productos se diferencian, entre otras características, por la cuantía económica de la garantía que ofrecen.

Además de dichos productos, se puede obtener un ID SSL gratuito de prueba, pulsando "*Free Trial SSL ID*":

En el proceso de alistamiento (*enrollment*), además de enviar diversa información, se ha de adjuntar (en un campo de texto) la propia Petición de Firma de Certificado (CSR) X.509 en formato PEM del servidor SSL.

La lista de servidores SSL compatibles con VeriSign (tanto de 40 como de 128 bits), se puede comprobar en http://www.verisign.com/products/site/compatibility.html. En otra página se nos guía acerca de cómo obtener el CSR en los servidores SSL más populares (http://www.verisign.com/support/csr).

Una vez se halla rellenado los sucesivos formularios, en no más de una hora llegará un email a la dirección de correo que especificamos con el asunto: "*VeriSign Trial Server ID*" donde se especifica, además del número de petición, nuestro certificado X.509 en formato PEM:

```
YOUR ORDER NUMBER: 114735964

Dear VeriSign Customer:

Congratulations!  Your Trial SSL Server Digital ID, issued to:

CN:  192.168.0.1
O:   UNIVERSIDAD DE SEVILLA
OU:  DEPARTAMENTO DE TELAMATICA

can be installed by following the instructions below.

Before using your Trial SSL Server ID, install the Test CA
Root in each browser you plan to use as part of your test of
SSL.

*********************************************************
* To download the Test CA Root, go to:

http://www.verisign.com/server/trial/faq/index.html
```

```
and follow the instructions there.


* To install your Trial SSL Server ID, go to:

http://www.verisign.com/support/install/index.html#trial

and follow the instructions there.

**********************************************************

After testing your Trial SSL Server ID, you will need to purchase
a full service Server ID, available as part of VeriSign's trust
solutions.  Follow these easy steps to continue benefiting from
VeriSign SSL Server IDs:

* Step 1 - Visit:  http://www.verisign.com/products/site/

Here you can familiarize yourself with the full range
of available VeriSign Secure Site services, with packages
including:

- VeriSign Secure Site Pro and Commerce Site Pro
solutions with Global IDs that enable 128-bit
SSL encryption--the world's strongest--with all
Microsoft and Netscape browsers.

- VeriSign Commerce Site and Commerce Site Pro ID's
bundled with VeriSign's payment processing solution,
Payflow Pro.  These e-commerce solutions are available at:
http://www.verisign.com/products/site/commerce/

- Additional VeriSign e-commerce services, such as the
widely-recognized Secure Site Seal to post on your site
as a symbol of trust.  VeriSign also offers up to $250,000
of NetSure protection and Network Security auditing by Qualys.
Learn more about all of VeriSign's trust solutions at:
http://www.verisign.com/products/site/


* Step 2 - Order VeriSign SSL Server IDs at:

http://www.verisign.com/products/site/secure

**********************************************************

It's that easy!  If you have any questions about installing or using your
Trial SSL Server ID, please contact us at 650-426-3400.

Thank you for your interest in VeriSign products!


Customer Support Department
VeriSign, Inc.
The Value of Trust (sm)

E-mail: support@verisign.com
Web: http://www.verisign.com
```

```
For sales call (650)426-5112
Fax: (650)961-8870


P.S. During testing of your VeriSign Trial SSL ID, you can receive
additional technical information by downloading our new white paper
on implementing SSL and payment processing solutions at:
http://www.verisign.com/rsc/gd/pmt/ecomm-tech/



-----BEGIN CERTIFICATE-----
MIIDTDCCAvagAwIBAgIQEucIX9/JvhNHwfyZ+zvmeDANBgkqhkiG9w0BAQUFADCB
qTEWMBQGA1UEChMNVmVyaVNpZ24sIEluYzFHMEUGA1UECxM+d3d3LnZlcmlzaWdu
LmNvbS9yZXBvc2l0b3J5L1Rlc3RDUFMgSW5jb3JwLiBCeSBSZWYuIExpYWIuIExU
RC4xRjBEBgNVBAsTPUZvciBWZXJpU2lnbiBhdXRob3JpemVkIHRlc3Rpbmcgb25s
eS4gTm8gYXNzdXJhbmNlcyAoQylWUzE5OTcwHhcNMDIxMTExMDAwMDAwWhcNMDIx
MTI1MjM1OTU5WjCBizELMAkGA1UEBhMCRVMxDjAMBgNVBAgTBVNwYWluMRAwDgYD
VQQHFAdTZXZpbGxhMR8wHQYDVQQKFBZVbml2ZXJzaWRhZCBkZSBTZXZpbGxhMSMw
IQYDVQQLFBpEZXBhcnRhbWVudG8gZGUgVGVsYW1hdGljYTEUMBIGA1UEAxQLMTky
LjE2OC4wLjEwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAOGMiC+mxDSwBZQv
SWExB+AKSmAXd6L6T9W2zbKnTARgLaai49q5rIqCjMDtN1puh/EbVcIWebjlxzMc
t1sdEjezfLjwy/YalnSVjk1esG5C/rza0ofaBg9vvRk0HHNqcIJxVpK3MKuJxLey
3UuWOKZBqcvQfnuJBgHMDi3QvsZVAgMBAAGjgdEwgc4wCQYDVR0TBAIwADALBgNV
HQ8EBAMCBaAwQgYDVR0fBDswOTA3oDWgM4YxaHR0cDovL2NybC52ZXJpc2lnbi5j
b20vU2VjdXJlU2VydmVyVGVzdGluZ0NBLmNybDBRBgNVHSAESjBIMEYGCmCGSAGG
+EUBBxUwODA2BggrBgEFBQcCARYqaHR0cDovL3d3dy52ZXJpc2lnbi5jb20vcmVw
b3NpdG9yeS9UZXN0Q1BTMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjAN
BgkqhkiG9w0BAQUFAANBAETh7EJkKW6+zUers0eXa3eY1VurYmUiXDGJBVZvE5Jg
pYMmx2etWaO8tXoi5mly0gfqLmttpS5eqArJRg4epxw=
-----END CERTIFICATE-----
```

III.3.3.1.2. CPS

A continuación se introduce un ejemplo de Certification Practice Statement (CPS) que regula el comportamiento de la CA:

```
VERISIGN TEST CERTIFICATION AUTHORITY
CERTIFICATION PRACTICE STATEMENT

YOU MUST READ THIS VERISIGN TEST CERTIFICATION AUTHORITY PRACTICE STATEMENT
("TEST CPS") CAREFULLY. BY CLICKING "ACCEPT" BELOW AND/OR REQUESTING, USING,
OR RELYING UPON A TEST CERTIFICATE OR THE TEST CA ROOT CERTIFICATE (AS THESE
TERMS ARE DEFINED BELOW), YOU AGREE TO BE BOUND BY THE TERMS OF THIS TEST CPS,
AND TO BECOME A PARTY TO THIS TEST CPS. IF YOU DO NOT AGREE TO THE TERMS OF
THIS TEST CPS, YOU MUST NOT REQUEST, USE, OR RELY UPON A TEST CERTIFICATE, THE
TEST CA ROOT CERTIFICATE, OR ANY COMPONENT THEREOF.

This VeriSign Test Certification Authority Certification Practice Statement
Agreement ("Test CPS") is entered into between VeriSign, Inc. ("VeriSign"),
and the entity listed in the above enrollment form ("Customer").

Background
This Test CPS sets forth certain terms and conditions under which VeriSign's
Test Certification Authority ("Test CA") issues and manages Customer's test
Server Digital IDSM certificate ("Test Certificate") and under which the
certificate for the test CA ("Test CA Root Certificate") and Test Certificate
```

are used. A person or entity (including but not limited to Customer) that is the subject of, and has been issued a Test Certificate, and is capable of using, and is authorized to use, the private key that corresponds to the public key listed in the Test Certificate is a subscriber ("Subscriber"). The Test Certificate enables Customer to simulate the operation of a web site that provides certain security services when a Server Digital IDSM certificate is properly used. VeriSign makes the Test CA Root Certificate available on its web site for authorized test purposes only in accordance with this Test CPS or other applicable agreement. An employee or authorized representative of the Customer downloading and/or using the Test CA Root Certificate, can simulate the process by which a browser authenticates and establishes a secure channel to a web site that properly uses a Server Digital IDSM certificate.

In consideration of the mutual promises in this Test CPS, and intending to be legally bound, Customer and VeriSign agree as follows:

1. USE LIMITED TO AUTHORIZED TESTING PURPOSES. CUSTOMER SHALL ONLY USE THE TEST CERTIFICATE AND THE TEST CA ROOT CERTIFICATE FOR TEST PURPOSES. CUSTOMER SHALL NOT USE OR RELY UPON THE TEST CERTIFICATE, THE TEST CA ROOT CERTIFICATE, AND ALL COMPONENTS THEREOF FOR ANY OTHER PURPOSES, INCLUDING COMMERCIAL TRANSACTIONS, AUTHENTICATING THE IDENTITY OF A SUBSCRIBER OR THE TEST CA, OR IN CONNECTION WITH ENSURING THE CONFIDENTIALITY OF ANY INFORMATION. CUSTOMER SHALL NOT REQUEST OR USE THE TEST CERTIFICATE OR THE TEST CA ROOT CERTIFICATE FOR ANY PURPOSE OTHER THAN FOR AUTHORIZED TECHNICAL TESTING. CUSTOMER ACKNOWLEDGES THAT NEITHER THE IDENTITY NOR AUTHORITY OF SUBSCRIBERS HAS BEEN AUTHENTICATED OR APPROVED BY THE TEST CA OR VERISIGN.

2. Issuance and Use. Upon VeriSign's receipt and approval of a complete and approved application for a Test Certificate, VeriSign shall issue the Test Certificate to Customer. VeriSign shall also provide Customer with access to the Test CA Root Certificate. The Test Certificate and the Test CA Root Certificate shall be used exclusively for authorized testing purposes in simulating the process by which a browser authenticates and establishes a secure channel to a web site that properly uses a Server Digital IDSM certificate.

3. Fees. If Customer selected the extended Test Certificate option, Customer shall pay VeriSign the corresponding fee listed in the enrollment form. Customer shall pay any and all applicable sales taxes.

4. No Suspension or Revocation Services. Customer acknowledges that VeriSign shall not be required to suspend or revoke the Test Certificate at the request of Customer but Customer shall be responsible for requesting revocation of Customer's Test Certificate prior to requesting a refund for such Test Certificate.

5. Permission to Publish Information. VeriSign shall be entitled to publish and otherwise to disclose publicly any Test Certificate or any portion of the content of a certificate, in connection with VeriSign's dissemination of Test Certificate information within and outside of VeriSign's Test CA hierarchy.

6. DISCLAIMER OF WARRANTY. VERISIGN ISSUES TEST CERTIFICATES "AS IS." VERISIGN DISCLAIMS ANY WARRANTIES WHATSOEVER WITH RESPECT TO THE SERVICES PROVIDED BY VERISIGN HEREUNDER, INCLUDING WITHOUT LIMITATION ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. VERISIGN MAKES NO REPRESENTATION OR WARRANTY TO ANY PERSON THAT ANY SUBSCRIBER TO WHICH IT HAS ISSUED A TEST CERTIFICATE IS IN FACT THE PERSON OR ORGANIZATION IT CLAIMS TO BE IN INFORMATION SUPPLIED TO VERISIGN, OR THAT ANY PERSON OR

ORGANIZATION IS IN FACT THE PERSON OR ORGANIZATION LISTED IN A TEST CERTIFICATE OR THE TEST CA ROOT CERTIFICATE. VERISIGN MAKES NO ASSURANCES OF THE ACCURACY, AUTHENTICITY, INTEGRITY, OR RELIABILITY OF INFORMATION CONTAINED IN TEST CERTIFICATES OR THE TEST CA ROOT CERTIFICATE, OR OF THE RESULTS OF CRYPTOGRAPHIC METHODS IMPLEMENTED IN CONNECTION WITH SUCH CERTIFICATES. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY VERISIGN OR ITS EMPLOYEES OR REPRESENTATIVES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF VERISIGN'S OBLIGATIONS. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE DISCLAIMERS IN THIS TEST CPS MAY NOT APPLY TO YOU.

7. LIMITATION OF LIABILITY. IN NO EVENT SHALL VERISIGN'S LIABILITY TO ANY AND ALL PERSONS FOR ANY AND ALL CLAIMS, LOSSES, OR DAMAGES RELATING TO, IN WHOLE OR IN PART, THIS TEST CPS, THE PRODUCTS AND/OR SERVICES, OR OTHERWISE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, EXCEED FIFTY U.S. DOLLARS ($ 50.00). UNDER NO CIRCUMSTANCES WHATSOEVER SHALL VERISIGN BE LIABLE FOR SPECIAL, INDIRECT, RELIANCE, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS, LOST DATA, OR LOSS RESULTING FROM BUSINESS INTERRUPTION, EVEN IF VERISIGN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

8. Indemnity. Customer shall indemnify VeriSign and its directors, officers, agents, employees, contractors, parents, affiliates, or subsidiaries (collectively, the "Indemnified Parties") and hold the Indemnified Parties harmless from and against any losses, costs, damages, and fees (including attorneys' fees) incurred by the Indemnified Parties in connection with: (a) any breach by Customer, or its employees or authorized representatives, of any warranty or obligation under this Test CPS or (b) any unauthorized representation or misrepresentation of fact by Customer to any third party with respect to any product or service provided under this Test CPS ("Indemnity Conditions"). Upon appropriate notice, Customer shall defend, at its expense, any claim brought against one or more of the Indemnified Parties based on or arising out of one or more of the Indemnity Conditions.

9. Certificate Expiration. The Test Certificate, and the parties obligations under this Test CPS, shall expire either fourteen (14) days (for free Test Certificates) or sixty (60) days (for paid for Test Certificates) from the date of issuance of the Test Certificate. This Test CPS may be terminated (a) by either party immediately upon the institution by or against the other party of insolvency, receivership, or bankruptcy proceedings, upon any assignment for the benefit of the other party's creditors, or upon the other party's dissolution or ceasing to do business or (b) by VeriSign immediately and without prior notice in the event of a breach of any of the duties, obligations, terms, or provisions of this Test CPS (a "Breach") by RA if the Breach may compromise the security of VeriSign's services.

10. After Expiration or Revocation. Upon expiration or revocation for any reason, Customer shall not, for any reason, use or rely upon the Test Certificate and all components thereof. Such expiration or revocation shall not affect Sections 5, 6, 7, 8, 10 and 11 of this Test CPS and all subsections of the foregoing, which shall continue in full force and affect to the extent necessary to permit the complete fulfillment thereof. Customer shall delete the Test CA Root Certificate and the Test Certificate from Customer's browsers upon expiration or revocation of the Test Certificate or upon completion of testing.

11. Miscellaneous Provisions.

11.1 Governing Law; Venue; Waiver of Jury Trial. This Test CPS shall be governed by the laws of the State of California other than its conflicts of

laws principles. The parties irrevocably consent to the exclusive jurisdiction of, and venue in, the following courts for the resolution of any dispute arising hereunder: the Superior or Municipal Court of Santa Clara County, California and the United States District Court for the Northern District of California at San Jose (collectively, "California Courts"). Any suit by or between the parties relating to this Test CPS shall be brought in and resolved by one of the California Courts. The parties hereby waive any right to jury trial with respect to any suit brought in connection with this Test CPS.

11.2 Successors and Assigns. Except as otherwise provided herein, this Test CPS shall be binding upon, and inure to the benefit of the respective successors, executors, heirs, representatives, administrators, and assigns of the Parties. Neither this Test CPS nor the Test Certificate shall be assignable by Customer. Any such attempted assignment or delegation shall be void and of no effect.

11.3 Severability. If any provision of this Test CPS, or the application thereof, shall for any reason and to any extent, be invalid or unenforceable, the remainder of this Test CPS and application of such provision to other persons or circumstances shall not be affected thereby and shall be interpreted so as best to reasonably effect the intent of the parties. IT IS EXPRESSLY UNDERSTOOD THAT EACH AND EVERY PROVISION OF THIS TEST CPS THAT PROVIDES FOR ANY LIMITATION, DISCLAIMER OR EXCLUSION OF LIABILITY, WARRANTIES, OR DAMAGES IS INTENDED BY THE PARTIES TO BE SEVERABLE AND INDEPENDENT OF ANY OTHER PROVISION AND TO BE ENFORCED AS SUCH.

11.4 Entire Agreement. This Test CPS constitutes the entire understanding and agreement of the Parties hereto with respect to the subject matter hereof and supersedes all prior and contemporaneous oral and written agreements or understandings among the parties.

11.5 Notices. Whenever either party desires or is required to give any notice, demand, or request to VeriSign with respect to this Test CPS, each such communication shall be in writing and shall be effective only if it is delivered by a courier service that confirms delivery in writing or is mailed, certified or registered mail, postage prepaid, return receipt requested, addressed to the representative of the Customer at the address below or to VeriSign at: Practices, VeriSign, Inc., 150 Charleston Road, Mountain View, CA 94043, voice: 650-961-7500 and fax: 650-961-7300. Customer shall immediately advise VeriSign of any legal notice served on Customer that might affect VeriSign. Such communications shall be effective when they are received.

11.6 Trademarks and Trade Names. By reason of this Test CPS or the performance hereof, Customer and VeriSign shall acquire no rights of any kind in any trademark, brand name, logo, or product designation of the other Party and shall not make any use of the same for any reason except as otherwise authorized in writing by the Party that owns all rights to such trademarks, trade names, logos, or product designation.

11.7 Independent Relationship. Customer, and Customer's employees, consultants, contractors, and agents are not agents, employees, joint ventures, or joint venturers of VeriSign, and they have no authority to bind VeriSign by contract or otherwise to any obligation.

III.3.3.1.3. Conclusión

Cabe decir como bug , que la página web que accede al servicio de búsqueda de la CA, no funciona, dejando inutilizable además el servicio de revocación de la CA.

No se pueden elegir formatos de certificados, limitándose a X.509 PEM

III.3.3.2. Otras CA's

En la página web http://www.whichssl.com perteneciente a la empresa COMODO se puede encontrar una comparativa entre distintas CA's que ofrecen entre sus productos la obtención de certificados SSL para sitios web y comercio electrónico. De ellas podemos mencionar:

COMODO InstantSSL	http://www.instantssl.com	También perteneciente a COMODO.
VeriSign	http://www.verisign.com	Ya se ha comentado.
thawte	http://www.thawte.com	**Perteneciente a VeriSign.**
Entrust	http://www.entrust.com	
BALTIMORE	http://www.baltimore.com	
GeoTrust	http://www.geotrust.com	
freeSSL.com	http://www.freessl.com	Perteneciente a GeoTrust
ipsCA	http://www.ipsca.com	Empresa **española** con gran cantidad de productos además de los certificados.
Real Casa de la Moneda FÁBRICA NACIONAL DE MONEDA Y TIMBRE	http://www.fnmt.es	Utiliza a las Delegaciones de Hacienda en España como RA's.

La mayoría de estas CA's disponen de diversos productos comerciales. Además, suelen disponer de dos productos gratuitos (y por ello limitados):

- Certificado personal para el email.
- Certificado para SSL en prueba durante algún tiempo.

En la obtención de un **certificado personal para email**, es el propio navegador el encargado de generar las claves y **no se permite introducir el CSR**. Este hecho nos limitará el uso de la encriptación al navegador que se haya utilizado.

En la obtención de un **certificado SSL** de prueba, por el contrario sí podremos introducir el CSR. Tiene el inconveniente en que sólo es válido durante algún tiempo limitado, además **no suele permitirse la revocación**.

Normalmente **no se utiliza HTTPS** y se han tenido algunos **problemas** con la obtención de la CRL. Además, la **elección de formatos** suele ser **limitada**.

Al ser un servicio gratuito, el certificado que nos entregan está firmado por otro que no se encuentra preinstalado en los navegadores (por los que han tenido que pagar a los fabricantes).

IV. Pliego de condiciones

IV.1. SuSE Linux 8.0 Proffesional

IV.1.1. Descripción

SuSE Linux (www.suse.de/es) es una distribución del sistema operativo Linux (Unix no propietario) de tipo comercial con una instalación muy sencilla de la mayor parte de los paquetes que vamos a necesitar.

IV.1.2. Características

Se ha elegido esta distribución Linux por los siguientes motivos:

- Distribuye un sistema operativo Linux , recomendado (e incluso necesario) para poder trabajar con la mayoría de los paquetes que se utilizarán (OpenSSL, Apache...)
- Su instalación es muy sencilla en comparación con otras distribuciones Linux (sobre todo con las más antiguas).
- Los paquetes están suficientemente actualizados (primavera de 2002) por lo que además disminuye notablemente la necesidad de instalar por nuestra cuenta controladores para hardware y las propias aplicaciones con el consiguiente ahorro de tiempo y dinero.
- Se puede encontrar con bastante facilidad en comercios del ramo.
- Ha sido elegida mejor distribución Linux por PC Actual (noviembre 2002)

IV.1.3. Versión

SuSE Linux 8.0 Proffesional

IV.1.4. Instalación

Para la instalación de los paquetes que describiremos a continuación, existen dos posibilidades: que no tengamos previamente instalado el sistema operativo en el PC o que sí lo hayamos instalado con anterioridad.

1. Para el caso en que no tengamos instalado el sistema operativo deberemos proceder a su instalación.

   Para la instalación introduciremos el CD número 1 con la precaución de tener activado el arranque desde CDROM en la BIOS de la placa base (consultar instrucciones de la placa base para resolver cualquier de duda). Encenderemos el PC con el CD introducido y seguiremos los pasos en la instalación. Dicha instalación se sale fuera de los propósitos de este Proyecto por lo que para cualquier aclaración me remito al manual que lleva el propio sistema operativo.

   Donde sí nos detendremos es en la ventana de la propia instalación titulada "*Comprobación de la Instalación*" donde deberemos pinchar bajo el apartado: "*Software*" en la frase "*Selección detallada de software"*. Cuando nos aparezca la siguiente ventana pincharemos en el botón denominado "*Paquetes individuales...*". Continúa en 3.

2. Si por el contrario ya tenemos instalado el sistema operativo, sólo nos resta instalar, o comprobar que los tenemos instalados en su defecto, los paquetes de los que vamos a hacer uso. Para ello habremos de entrar como "*root*" en el sistema y entrar en la aplicación de configuración "*yast2*". En el marco derecho de la aplicación pinchamos en

3. En el marco de "*Grupo*" seleccionamos el grupo de paquetes: "*Productividad/Trabajo en red/Seguridad*" y seleccionamos los siguientes paquetes que aparecen en el marco de la derecha:

	Nombre	***Tamaño***	***Descripción***	***Versión***
X	**openssl**	2.48MB	Capa de Zócalos Seguros y seguridad en la Capa de Transporte	0.9.6c-29
X	**openssl-doc**	1.81MB	Documentación y ejemplos de OpenSSL	0.9.6c-29

Para poder montar un servidor HTTP y HTTPS, deberemos seleccionar los paquetes siguientes del grupo "*Productividad/Trabajo en red/Web/Servidores*"

	Nombre	***Tamaño***	***Descripción***	***Versión***
X	**apache**	1.65MB	El servidor HTTP Apache	1.3.23-73
X	**apache-doc**	3.71MB	Apache documentation	1.3.23-73
X	**mod-ssl**	1.34MB	Criptografía potente para el servidor Web Apache 1.3	2.8.7-41

Para poder usar los servlets de Java, además de tener instalado apache, se deberá seleccionar, en el grupo "*Productividad/Trabajo en red/Web/Servidores/Frontends*":

	Nombre	Tamaño	Descripción	Versión
X	**jakarta-tomcat**	17.72MB	Servidor Java y Servlets Java	4.0.1-1227

Para poder enviar los correos con las claves a los usuarios, se seleccionará en el grupo "*Productividad/Trabajo en red/Correo electrónico/Clientes*":

	Nombre	Tamaño	Descripción	Versión
X	**mutt**	2.68MB	Programa de correo	1.3.27i-62

La descripción de los paquetes es la siguiente (extraída de la propia instalación):

Nombre:	**openssl**
Licencia :	1998-2001 The OpenSSL Project
Versión:	0.9.6c-29, Sat Mar 23 20:14:25 2002
Requerido por:	cron, shadow, pico, siga, susetour, susetour_en, kdeaddons3, kdeadmin3, kdeartwork3, kdebase3, kdebase3-SuSE, kdebase3-nsplugin, kdegames3, kdegraphics3, kdelibs3, kdemultimedia3, kdenetwork3, kdepim3, kdetoys3, kdeutils3, kinternet, koncd, susehelpcenter, autofs, cups-libs, fetchmail, mailx, mutt, openldap2-client, pine, samba-client, sendmail, smpppd, tcpdump, w3m, wget, cyrus-sasl, heimdal-lib, openssh, openssl, ifnteuro, isax, sax2, xf86, xf86tools, xbanner, postgresql-libs, python, kde2-compat, curl, vorbis-tools, apache-contrib, grip, noteedit, terminatorX, htmldoc, xdrawchem, seccheck, kwintv, quanta
Descripción:	El Proyecto OpenSSL es un esfuerzo colaborativo para desarrollar un conjunto de herramientas robusto, de nivel comercial, completo y de Código Abierto que implemente los protocolos Secure Sockets Layer - Capa de Zócalos Seguros (SSL v2/v3) y Transport Layer Security - Seguridad de Capa de Transporte (TLS v1) con una criptografía mundial fuerte. El Proyecto está gestionado por una comunidad de voluntarios alrededor del mundo que utiliza Internet para comunicarse, planear y desarrollar la caja de herramientas OpenSSL y la documentación relacionada. Derivación y Licencia OpenSSL se basa en la excelente librería SSLeay desarrollada por Eric A. Young y Tim J. Hudson. La caja de herramientas OpenSSL se entrega bajo una licencia del tipo Apache, lo que básicamente significa que es usted libre de tenerlo y utilizarlo para propósitos comerciales y no comerciales.
Autores::	Mark J. Cox, Ralf S. Engelschall , Dr. Stephen Henson , Ben Laurie , Bodo Moeller , Ulf Moeller , Holger Reif , Paul C. Sutton
Nombre:	**openssl-doc**
Licencia :	1998-2001 The OpenSSL Project
Versión:	0.9.6c-29, Sat Mar 23 20:14:25 2002
Requerido por:	openssl-doc
Descripción:	Este paquete contiene la documentación y ejemplos de OpenSSL.
Autores:	Mark J. Cox , Ralf S. Engelschall , Dr. Stephen , Ben Laurie , Bodo Moeller , Ulf Moeller , Holger Reif , Paul C. Sutton
Nombre:	**jakarta-tomcat**
Licencia :	The Apache Software License
Versión:	4.0.1-127, Tue Mar 26 16:43:20 2002
Requerido por:	jakarta-tomcat

Descripción:	La meta del Proyecto Jakarta es proporcionar soluciones de servidor de calidad comercial basadas en la Plataforma Java que se está desarrollando de una forma abierta y cooperativa. Tomcat: Implementación de las Páginas de Referencia del Servidor y los Servlets Java El producto insignia, Tomcat, es una implementación de clase mundial de las especificaciones Java Servlet 2.2 and JavaServer Pages 1.1. Esta implementación se utilizará en el servidor Web Apache así como en otros servidores Web y herramientas de desarrollo.
Autores:	Hans Bergsten, James Duncan, Pierpaolo Fumagalli, Craig McClanahan, Sam Ruby, Jon Stevens, Anil Vijendran, Brian Behlendorf, Kevin Burton, Danno Ferrin, Jason Hunter, Ramesh Mandava, Stefano Mazzocchi, Rajiv Mordani, Harish Prabhandham, Jean-Luc Rochat, James Todd
Nombre:	**apache**
Licencia :	1995-2001 The Apache Group
Versión::	1.3.23-73, Tue Mar 26 16:29:26 2002
Requerido por:	apache, apache-contrib

Descripción:	El servidor HTTP Apache. Muy pocos programas se han extendido tan rápidamente y han atraído tanto seguimiento en tan poco tiempo como este servidor HTTP. Según Netcraft, Apache se ejecuta sobre un 49.05% de todos los servidores conectados a Internet. Esto representa unos 1.182.142 sistemas. Reemplazó en sólo unos meses al servidor NCSA, que era hasta entonces el favorito imbatido. Esto es debido, sin duda, a que Apache se deriva de los fuentes del servidor NCSA y que su configuración es compatible con la del NCSA/1.3. El Proyecto Apache se estableció para proporcionar un fórum para el desarrollo de un servidor HTTP libremente disponible. La meta era desarrollar un servidor seguro, eficiente y fácilmente ampliable, siguiendo de cerca y al unísono el desarrollo del protocolo HTTP. A parte de un mejor rendimiento, la eliminación de errores conocidos, y la mejor cumplimentación de los estándares HTTP (HTTP/1.1), Apache proporciona las siguientes características útiles, que faltan con frecuencia en otros servidores: * archivos de contraseñas, utilizadas para proteger documentos contra accesos no autorizados, puede ser indexado de varias formas. Como consecuencia, la búsqueda de una entrada es más eficiente que para un archivo ordinario de texto. * los errores de peticiones del cliente pueden responderse de una forma flexible. Se pueden utilizar scripts CGI para analizar el error y generar respuestas más informativas de lo que proporcionan los códigos de estado HTTP, tanto al cliente como al administrador. * no hay restricciones sobre sinónimos ni redirecciones. Las directivas Alias y Redirect se pueden usar en número y profundidad arbitrarias. * Apache puede escoger la mejor representación de un recurso basándose en preferencias suministradas por el hojeador para el tipo de soporte, idiomas, conjunto de caracteres y codificación ("Negociación del Contenido"). * con Apache puede usted implementar diferentes servidores virtuales. Así, peticiones para direcciones IP diferentes (que se refieren al mismo ordenador) pueden ser diferenciadas y, dependiendo de la dirección IP, tratadas convenientemente. Otras ventajas adicionales son el concepto de módulo y la API (Application Programmers Interface) bien documentada. Las ampliaciones se desarrollan con facilidad y las nuevas características se integran fácilmente al servidor. Por ejemplo, existe una implementación de Secure Sockets Layer (SSL) de Netscape para Apache. Esto permite una comunicación segura entre el servidor y el cliente. Apache se instala en los directorios siguientes: /etc/httpd -> archivos de configuración /usr/sbin/httpd -> binario del servidor /var/log/httpd. * -> archivos de protocolos /var/run/httpd.pid -> PID del servidor /usr/local/httpd -> raíz del servidor /sbin/init.Rob l , RRob McCool , Robert S. Thau , Roy T. Fielding , Brian Behlendorf , Harald Hanche-Olsen , David Robinson , Wietse Venema , Keith Shafer , Kevin Steves obert S. Thau , Roy T. Fielding , Brian Behlendorf , Harald Hanche-Olsen , David Robinson , Wietse Venema , Keith Shafer , Kevin Steves d/apache -> script de Inicio/Fin
Autores:	Rob McCool , Robert S. Thau , Roy T. Fielding , Brian Behlendorf , Harald Hanche-Olsen , David Robinson , Wietse Venema , Keith Shafer , Kevin Steves
Nombre:	**apache-doc**
Licencia :	1995-2001 The Apache Group
Versión:	1.3.23-73, Tue Mar 26 16:29:26 2002
Requerido por:	apache-doc
Descripción:	Este paquete contiene la documentación HTML del servidor web Apache.
Autores:	Rob McCool , Robert S. Thau , Roy T. Fielding , Brian Behlendorf , Harald Hanche-Olsen , David Robinson , Wietse Venema , Keith Shafer , Kevin Steves
Nombre	**mod-ssl**

Licencia:	1995-2001 The Apache Group
Versión:	2.8.7-41, Tue Mar 26 16:29:26 2002
Requerido por:	mod_ssl
Descripción:	Este módulo proporciona criptografía potente para el servidor Web Apache 1.3 a través de los protocolos Secure Sockets Layer (SSL v2/v3) y Transport Layer Security (TLS v1), con la ayuda de la caja de herramientas OpenSSL de Open Source SSL/TLS, que se basa en SSLeay de Eric A. Young y Tim J. Hudson. El paquete mod_ssl fue creado en Abril de 1998 por Ralf S. Engelschall, y está derivado originalmente del software desarrollado por Ben Laurie para usarlo en el Proyecto de servidor HTTP Apache-SSL. El paquete mod_ssl se entrega bajo una licencia de tipo BSD, lo que básicamente significa que es usted libre de usarlo para propósitos comerciales y no comerciales.
Autores:	Ralf S. Engelschall
Nombre	**mutt**
Licencia:	GNU GPL
Versión:	1.3.27i-62, Mon Mar 25 21:33:27 2002
Requerido por:	mutt
Descripción:	Programa de correo de usuario muy potente. Soporta (entre otras cosas interesantes) resaltado, encadenado y PGP. No obstante, toma su tiempo habituarse a él.
Autores:	Michael Elkins,Thomas Roessler

Tabla 8: Descripción de los paquetes de SuSE Linux 8.0 Proffessional

IV.2. Java 2

IV.2.1. Descripción

Java (http://www.java.sun.com) es tanto un lenguanje de programación como una plataforma propiedad de Sun Microsystems (http://www.sun.com). Permite a los desarrolladores:

- Escribir software en una plataforma y ejecutarlo en otra (“write once, run everywhere”).
- Se pueden crear programas que se ejecuten en un navegador web (applets).
- Desarrollar aplicaciones del lado del servidor (servlets) reemplazando el uso de guiones (scripts) CGI.
- Escribir aplicaciones para otros dispositivos como teléfonos móviles, neveras, ...

IV.2.2. Características

Entre sus características cabe mencionar:

- Simplicidad ya que la administración de memoria y recolección de basura corren a cargo de la Máquina Virtual de Java (JVM). Java a diferencia de C y C++ no usa punteros estrictamente hablando sino referencias a los recursos de la JVM.
- Independiente de la arquitectura.
- Orientado a objetos, lo cual permite desarrollar y mantener software con menor coste.
- Portable.
- Distribuido ya que permite que las librerías se carguen desde Internet, por ejemplo.
- Interpretado. Se compila a bytecode y éste se interpreta
- Multitarea (o multihilo) que permite que los programs aprovechen al máximo los recursos del sistema, como la CPU.
- Robusto.
- Dinámico ya que las clases se cargan dinámicamente.
- Seguro ya que las clases se comprueban antes de cargarlas.

IV.2.3. Versión

Java 2 Standard Development Kit (J2SDK) v1.4.0

IV.2.4. Paquetes utilizados

IV.2.4.1. JCA y JCE

La Arquitectura Criptográfica de Java (JCA) permite a los desarrolladores incorporar funcionalidad tanto de bajo como de alto nivel a sus programas.

La Extensión Criptográfica de Java (JCE) extiende la JCA para permitir API´s para la encriptación, intercambio de claves y Códigos de Autenticación de Mensajes (MAC). La JCA se entregaba como un paquete opcional en versiones anteriores de Java 2 SDK, pero se ha integrado a partir de Java 2 SDK v1.4.

La JCA sigue los siguientes principios:

- independencia de la implementación e interoperabilidad

- independencia del algoritmo y extensibilidad mediante el uso de una arquitectura basada en proveedor (provider).

Estos dos principios son complementarios: se pueden usar Proveedores de Servicios Criptográficos (CSP) sin preocuparse de detalles de implementación o sus algoritmos (ver más adelante los distintos CSP estudiados).

La JCE API cubre:

- Encriptación simétrica de bloque, como DES, RC2 e IDEA
- Encriptación simétrica de flujo, como RC4
- Encriptación asimétrica, como RSA
- Encriptación Basada en Contraseña (PBE)
- Acuerdo de claves
- Códigos de Autenticación de Mensajes (MAC)

Java 2 SDK v1.4 viene con un proveedor estándar llamado "*SunJCE*" que viene con los siguientes servicios criptográficos:

- Una implementación de los algoritmos de encriptación DES (FIPS PUB 46-1), Triple DES, y Blowfish en los modos Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Propagating Cipher Block Chaining (PCBC).
- Generadores de claves para los algoritmos DES, Triple DES, Blowfish, HMAC-MD5, y HMAC-SHA1.
- Una implementación del algoritmo de encriptación MD5 con PBE DES-CBC definido en PKCS #5.
- Una implementación del algoritmo de acuerdo de claves Diffie-Hellman entre dos o más partes y su correspondiente generador de pares de claves y parámetros.
- Gestores de parámetros para los algoritmos Diffie-Hellman, DES, Triple DES, Blowfish, y PBE.
- Una implementación de los algoritmos de resúmen con clave HMAC-MD5 y HMAC-SHA1 definidos en RFC 2104.
- Una implementación del esquema de relleno descrito en PKCS #5.
- Una implementación del almacén de claves (keystore) propietario llamado "JCEKS".

Debido a leyes de exportación de EE.UU. y Canada, sólo se permite una encriptación fuerte (lo permiten los archivos que dictan la política de seguridad `/usr/java/j2sdk1.4.0_02/jre/lib/security/local_policy.jar` y `/usr/java/j2sdk1.4.0_02/jre/lib/security/US_export_policy.jar`). Es decir, un número limitado de bits de clave. Para tener permiso para utilizar una longitud de clave ilimitada debemos descargar los archivos de política necesarios que se pueden descargar de

http://java.sun.com/products/jce/index-14.html. Dichos archivos se encuentran comprimidos en el paquete `jce_policy-1_4_0.zip` que debemos de descomprimir. Posteriormente copiamos los archivos: `local_policy.jar` y `US_export_policy.jar` a `/usr/java/j2sdk1.4.0_02/jre/lib/security/` sobreescribiendo los antiguos que solamente permitían encriptación fuerte pero no ilimitada.

IV.2.4.2. Servlets

Un servidor HTTP se limita a mandar la información que solicitan los navegadores. Generalmente suelen ser archivos HTML pero en ocasiones, como ocurre con los formularios, el servidor hace llamadas a programas específicos. Estos programas tradicionalmente han sido guiones (scripts) escritos en lenguaje Perl. Estos scripts que corren en los servidores son los Common Gateway Interface (CGI), que son muy simples y están ampliamente soportados. Desafortunadamente, no son muy eficientes porque el servidor lanza una copia del script por cada respuesta que debe generar, lo cual si el sitio Web es muy popular, significa una gran sobrecarga para el servidor. Los fabricantes han desarrollado alternativas propietarias como ISAPI (Microsoft) o NSAPI (Netscape) para paliar esta pérdida de rendimiento. JavaSoft, por su parte, ofrece una alternativa basada en Java a los CGI con la introducción de los servlets.

IV.2.4.2.1. Servlets vs. CGI

Los serviets presentan varias ventajas con respecto a los scripts CGI:

- Un servlet no se ejecuta en un proceso separado, lo que evita tener que lanzar una nueva instancia cada vez que se solicita su intervención y les hace más eficientes. Una vez que se ha invocado, se queda en memoria, sin consumir más recursos del servidor, aunque se llame insistentemente. Un programa CGI necesita ser cargado y descargado en cada invocación.
- Los serviets, al estar escritos en Java, disfrutan de las características de seguridad inherentes a Java, como puede ser el manejo de memoria.
- El servlet se ejecuta en una sandbox, de la misma forma que un applet es ejecutado por un navegador en otra; de esta manera, aumenta la seguridad en el servidor.
- Los servlets están escritos en Java, por lo que tienen acceso a todos los paquetes y potencia de Java.
- Por la misma razón, heredan la portabilidad entre plataformas de Java.

IV.2.4.3. JFC

Las Java Foundation Clases es una colección de clases para desarrollar Interfaces Gráficas de Usuario (GUI) funcionales en distintas plataformas. Dentro de esta colección cabe destacar dos grupos de clases que utilicemos en XMastin:

- Abstrac Window Toolkit (AWT).

• Swing que extiende AWT añadiéndole más componentes.

IV.2.5. Instalación

Para obtener el J2SDK hemos de descargar los ejecutables de instalación para el sistema operativo (SO) que se utilice desde http://www.java.sun.com.

Para el caso de la CA vamos a instalarlo en un entorno Linux. En el caso de Mastin (incluyendo a XMastin) podemos hacerlo en cualquier SO que soporte Java.

Al ejecutar la instalación nos muestra las licencias de Sun Microsystems, Inc.:

```
gab:~ # ./j2sdk-1_4_0_02-linux-i586-rpm.bin
...
gab:~ # rpm -i j2sdk-1_4_0_02-fcs-linux-i586.rpm
```

IV.2.5.1. /usr/sbin/setDefaultJava

Para cambiar el entorno por defecto del enlace simbólico /usr/lib/java, ejecutaremos el script `/usr/sbin/setDefaultJava`. Debido a que dicho script estaba creado para versiones más antiguas de Java, debemos cambiarlo de las líneas:

```
     "SunJava2")
       if [ "$DEVEL" = "yes" ] ; then
         if [ -x /usr/lib/SunJava2/bin/javac ] ; then
JAVALINKTO=/usr/lib/SunJava2; fi
       else
         if [ -x /usr/lib/SunJava2/jre/bin/java ] ; then
JAVALINKTO=/usr/lib/SunJava2; fi
       fi
```

a las líneas:

```
     "SunJava2")
       if [ "$DEVEL" = "yes" ] ; then
         if [ -x /usr/java/j2sdk1.4.0_02/bin/javac ] ; then
JAVALINKTO=/usr/java/j2sdk1.4.0_02; fi
       else
         if [ -x /usr/java/j2sdk1.4.0_02/jre/bin/java ] ; then
JAVALINKTO=/usr/java/j2sdk1.4.0_02; fi
       fi
       ;;
```

Para ejecutarlo, escribiremos:

```
gab:~ # setDefaultJava --devel SunJava2
```

IV.2.5.2. /usr/bin/setJava

La mayoría de las aplicaciones java, usan un entorno que se encuentra en la variable de entorno PATH. Algunas veces, se utilizan otras variables de entorno como JAVA_HOME, JRE_HOME, ... Los valores por defecto apuntan a */usr/lib/java* por defecto. Si se desea desarrollar o ejecutar una aplicación en otra versión de JDK o JRE, podemos usar el script `/usr/bin/setJava` que cambiará las variables de entorno al paquete de java pedido.

Variables de entorno	*Descripción*
PATH	Ruta de búsqueda para comandos. Se incluye el path para java y javac
JAVA_BINDIR	Ruta a java, javac, ...
JAVA_HOME	El directorio que contiene los archivos del JDK y/o del JRE
JRE_HOME	Directorio que contiene los archivos del JRE
JDK_HOME	Directorio que contiene todos los archivos del JDK. Para versiones 1.1.x o superiores
SDK_HOME	Directorio que contiene todos los archivos del SDK. Para versiones 1.2.0 o superiores

Tabla 9: Variables de entorno de Java

Y debido a que dicho script resulta anticuado, razones debemos también actualizar el script */usr/bin/setJava* de las líneas:

```
    "SunJava2")
      if [ "$DEVEL_setJava" = "yes" ] ; then
        if [ -x /usr/lib/SunJava2/bin/javac ] ; then
SELECTED_JAVA_setJava="SunJava2-SDK" ; fi
      else
        if   [ -x /usr/lib/SunJava2/bin/javac ] ; then
SELECTED_JAVA_setJava="SunJava2-SDK"
        elif [ -x /usr/lib/SunJava2/jre/bin/java ] ; then
SELECTED_JAVA_setJava="SunJava2-JRE"
        fi
      fi
      ;;
```

a las líneas:

```
         "SunJava2")
      if [ "$DEVEL_setJava" = "yes" ] ; then
        if [ -x /usr/java/j2sdk1.4.0_02/bin/javac ] ; then
SELECTED_JAVA_setJava="SunJava2-SDK" ; fi
      else
        if   [ -x /usr/java/j2sdk1.4.0_02/bin/javac ] ; then
SELECTED_JAVA_setJava="SunJava2-SDK"
        elif [ -x /usr/java/j2sdk1.4.0_02/jre/bin/java ] ; then
SELECTED_JAVA_setJava="SunJava2-JRE"
        fi
      fi
      ;;
```

Y las líneas:

```
    "SunJava2-SDK")
      JAVA_NEW_PATH_setJava=/usr/lib/SunJava2/bin
      JAVA_BINDIR=/usr/lib/SunJava2/bin
      JAVA_HOME=/usr/lib/SunJava2
      JRE_HOME=/usr/lib/SunJava2/jre
      JDK_HOME=/usr/lib/SunJava2
      SDK_HOME=/usr/lib/SunJava2
      ;;
    "SunJava2-JRE")
      JAVA_NEW_PATH_setJava=/usr/lib/SunJava2/jre/bin
      JAVA_BINDIR=/usr/lib/SunJava2/jre/bin
      JAVA_HOME=/usr/lib/SunJava2/jre
      JRE_HOME=/usr/lib/SunJava2/jre
      unset JDK_HOME
      unset SDK_HOME
      ;;
```

A las líneas:

```
    "SunJava2-SDK")
      JAVA_NEW_PATH_setJava=/usr/java/j2sdk1.4.0_02/bin
      JAVA_BINDIR=/usr/java/j2sdk1.4.0_02/bin
      JAVA_HOME=/usr/java/j2sdk1.4.0_02
      JRE_HOME=/usr/java/j2sdk1.4.0_02/jre
      JDK_HOME=/usr/java/j2sdk1.4.0_02
      SDK_HOME=/usr/java/j2sdk1.4.0_02
      ;;
    "SunJava2-JRE")
      JAVA_NEW_PATH_setJava=/usr/java/j2sdk1.4.0_02/jre/bin
      JAVA_BINDIR=/usr/java/j2sdk1.4.0_02/jre/bin
      JAVA_HOME=/usr/java/j2sdk1.4.0_02/jre
      JRE_HOME=/usr/java/j2sdk1.4.0_02/jre
      unset JDK_HOME
      unset SDK_HOME
      ;;
```

Para ejecutar el script, escribiremos en la línea de comandos:

```
gab:~ # source setJava --devel SunJava2
```

IV.2.6. keytool

IV.2.6.1. Synopsis

```
keytool [comandos]
```

IV.2.6.2. Descripción

`keytool` es la herramienta de gestión de certificados y claves de Java. Gestiona una base de datos (keystore) de claves privadas y certificados X.509 con claves públicas autenticadas por entidades de confianza. El keystore se protege por una contraseña.

IV.2.6.2.1. Entradas

Existen dos tipos de entradas en un keystore:

- **keyEntry**: almacena entre otras cosas la clave privada por lo que se almacena de forma protegida con otra contraseña para prevenir accesos no autorizados.
- **trustedCertEntry**: contiene una clave pública certificada. Se denomina *trusted* (de confianza) porque el dueño del keystore confía que la clave pública del certificado pertenece realmente al sujeto (subject) dueño del certificado. El expendedor del certificado se responsabiliza de esto firmando el certificado.

IV.2.6.2.2. Alias

A todas las entradas del keystore se accede mediante un único alias. Los alias no distinguen mayúsculas de minúsculas.

El alias se especifica cuando se añade una entrada al keystore usando `-genkey` para generar un par de claves pública / privada o cuando se añade un certificado mediante el comando `-import` a la lista de certificados de confianza. Posteriores llamadas a `keytool` usarán el mismo alias para referirse a dicha entrada.

IV.2.6.2.3. Localización del KeyStore

Cada comando keytool tiene la opción `-keystore` para especificar el nombre y localización del archivo persistente que contiene el keystore. Este archivo se almacena por defecto en un fichero llamado `.keystore` situado en el directorio del usuario determinado por la propiedad del sistema *`user.home`*.

IV.2.6.2.4. Creación del KeyStore

Se crea un KeyStore cuando se usa cualquiera de los comandos `-genkey`, `-import` o *`-identitydb`* para añadir datos a un keystore que no existe aún.

Más específicamente, si no se especifica la opción `-keystore` y no existe aún un keystore, se creará uno llamado `.keystore` en el directorio determinado por `user.home`.

IV.2.6.2.5. Implementación del KeyStore

La clase `KeyStore` del paquete `java.security` proporciona un interfaz bien definido para acceder y modificar la información contenida en un keystore. Se pueden disponer de distintas implementaciones, cada una de un tipo determinado.

Existe una implementación propiedad de Sun Microsystems denominada JKS (Java KeyStore). Protege cada clave privada con una contraseña individual y también protege la integridad del keystore entero con otra contraseña (que puede ser igual a la anterior).

Por defecto se utiliza el tipo de keystore especificado explícitamente en la propiedad `keystore.type` en el archivo de propiedades de seguridad (normalmente es JKS). Dicho archivo se llama java.security y reside en `java.home\lib\security` donde `java.home` es el directorio del JRE.

IV.2.6.2.6. Algoritmos soportados y tamaños de claves

keytool permite a los usuarios especificar el algoritmo de creación de pares de claves o de firma de cualquier proveedor de servicios criptográficos registrado. Es decir, las opciones `-keyalg` y `-sigalg` deben ser soportadas por la implementación del proveedor.

IV.2.6.2.7. Generando pares de claves

Para generar un par de claves pública/privada, usaremos el comando `-genkey` de la forma:

```
gab:~ # keytool -genkey -keyalg RSA -keysize 2048
Enter keystore password:  miContrasena
What is your first and last name?
  [Unknown]:  Gabriel Babiano Huete
What is the name of your organizational unit?
  [Unknown]:  Departamento de Telematica
What is the name of your organization?
  [Unknown]:  Universidad de Sevilla
What is the name of your City or Locality?
  [Unknown]:  Sevilla
What is the name of your State or Province?
  [Unknown]:  Spain
What is the two-letter country code for this unit?
  [Unknown]:  ES
Is <CN=Gabriel Babiano Huete, OU=Departamento de Telematica, O=Universidad de
Sevilla, L=Sevilla, ST=Spain, C=ES> correct?
  [no]:  y

Enter key password for <mykey>
        (RETURN if same as keystore password):
```

Crea una pareja de claves pública/privada para el algoritmo RSA de tamaño 2048 bits en el alias por defecto `mykey` en el keystore por defecto `.keystore` situado el el directorio `home` del usuario.

IV.2.6.2.8. Generando una CSR

Para crear una Petición de Firmado de Certificado (CSR) usando el formato PKCS#10 se usará el comando `-certreq`. Con el siguiente ejemplo, generamos una CSR de la clave pública contenida en la entrada referenciada por el alias `mykey` y lo guardamos en el fichero `csr.pem`:

```
keytool -certreq -alias mykey -file csr.pem
```

Posteriormente, este CSR se podrá remitir a una CA, y ésta, nos remitirá dicha petición ya firmada en un certificado una vez que haya comprobado la identidad del solicitante.

IV.2.6.2.9. Importando certificados

Para importar certificados desde un archivo usaremos el comando `-import` de la siguiente manera:

```
keytool -import -alias cert -file certfile.cer
```

que importa el certificado del archivo `certfile.cer` y lo guarda el en keystore en la entrada identificado por el alias `cert`.

Se puede importar un certificado por dos posibles motivos:

- para añadirlo a la lista de certificados de confianza
- para importar un certificado respuesta de una CA com resultado de enviarle una CSR (obtenida mediante `-certreq`) a dicha CA.

Dependiendo del tipo de entrada que sea la apuntada por el alias:

- Si el alias apunta a una keyEntry, entonces keytool asume que se está importando un certificado respuesta. keytool comprueba si la clave pública del certificado coincide con la guardada bajo el alias y se sale si son diferentes.
- Si el alias no apunta a un keyEntry, entonces keytool asume que se está añadiendo una entrada de certificado de confianza. En este caso, el alias no debería existir en el keystore. Si no fuera así, keytool mostraría un error y no importaría el certificado. Si el alias no existiese en el keystore, keytool crearía una entrada de certificado confiado con el alias especificado y lo asociaría con el certificado importado.

keytool puede importar certificados X.509 v1, v2 y v3 y cadenas de certificados en formato PKCS#7 consistentes en certificados de ese tipo. Los datos para ser importados deben ser provistos ya sea en formato codificado binario o en formato codificado imprimible (también conocido como Base64) definido en el standard RFC 1421. En el último caso, la codificación debe estar limitada por una cadena que empiece por `-----BEGIN` al principio y `-----END` al final.

IV.2.6.2.10. Exportando certificados

Para exportar un certificado a un archivo se usa el comando `-export` de la forma:

```
keytool -export -alias cert -file certfile.cer
```

Este ejemplo exporta el certificado `cert` al archivo `certfile.cer`. Esto es, si el alias `cert` corresponde a un keyEntry, se exporta el certificado que autentica la clave pública de jane. Si por el contrario, `cert` es el alias de un trustedCertEntry, entonces se exporta el certificado confiado.

IV.2.6.2.11. Mostrando certificados

Para mostrar el contenido de un keystore, se usa el comando `-list`, de la forma:

```
keytool -list
```

o de la forma

```
keytool -list -alias mykey
```

para mostrar el contenido de la entrada del keystore referida por `mykey`

Para mostrar el contenido de un certificado guardado en un archivo, se usa el comando `-printcert` de la forma:

```
keytool -printcert -file certfile.cer
```

que muestra la información contenida en el archivo `certfile.cer`. Para esto último no se necesita un keystore.

IV.2.6.2.12. Borrando entradas

Para borrar entradas, ya sean keyEntry o trustedCertEntry, lo podemos hacer con el comando `-delete`. Con el siguiente ejemplo borramos la entrada asociada al alias `mykey`:

```
keytool -delete -alias mykey
```

IV.2.6.2.13. Opciones

Opciones que aparecen para la mayoría de los comandos:

Opcion	*Descripción*
-v	"Verbose". Se mostrará información detallada
-help	Se mostrará la ayuda
-storetype *storetype*	Especifica el tipo de keystore
-keystore *keystore*	Localización del keystore
-storepass *storepass*	Contraseña utilizada para proteger la integridad del keystore. storepass debe tener al menos 6 dígitos. Si no se especifica en la línea de comandos, se preguntará por ella.
-provider *provider*	Nombre de la clase del proveedor del servicio criptográfico

Tabla 10: Opciones para la mayoría de los comandos de keytool

IV.2.6.2.14. Opciones por defecto

```
-alias mykey
-keyalg DSA
-keysize 1024
```

```
-validity 90
-keystore el_archivo_de_nombre_.keystore_en_el_directorio_casa_del_usuario
-file stdin_si_se_lee_o_stdout_si_se_escribe
```

El algoritmo de firma (opción `-sigalg`) se deriva del algoritmo de la clave privada:

- si es DSA, entonces la firma es SHA1
- si es RSA, entonces la firma es MD5

Para más información acerca de keytool, se puede consultar la documentación suministrada por Java.

IV.3. The Bouncy Castle Crypto Package

IV.3.1. Descripción

Es propiedad de *Legion of Bouncy Castle* y se encuentra disponible en http://www.bouncycastle.org. Actualmente (noviembre de 2002) se encuentra en la versión 1.15 y en ella podemos encontrar:

- Una API criptográfica en Java que funciona con todos las versiones desde J2ME hasta JDK 1.4
- Un proveedor de JCE y JCA
- Una implementación del JCE 1.2.1
- Generadores de certificados X.509 v1 y v3 y archivos PKCS12
- Generadores de S/MIME y CMS
- Un archivo jar firmado para JDK 1.4 y el JCE de Sun
- Incorpora una documentación bastante completa

IV.3.2. Versión

The Bouncy Castle Crypto Package 1.15

IV.3.3. Instalación

Se instala automáticamente en la instalación de Mastin.

IV.3.4. Otros CPS's

Aunque se ha escogido como Proveedor de Servicio Criptográfico (CSP) a Bouncy Castle, existen otros proveedores, cada uno con sus ventajas e inconvenientes.

IV.3.4.1. IAIK-JCE

IV.3.4.1.1. Descripción

Está desarrollada por el Institute for Applied Information Processing and Communications (IAIK) de la Universidad de Tecnología de Graz (Austria). Puede ser descargada desde http://jce.iaik.tugraz.at.

La principal ventaja es que se goza del respaldo del instituto aunque tiene como inconveniente que la documentación se encuentra únicamente en línea (http://jce.iaik.tugraz.at/products/01_jce/documentation). Además no se dispone del código sino del archivo jar firmado.

Requiere una licencia de exportación individual para cada cliente si está situado fuera de la UE excepto en Australia, Canada, República Checa, Hungría, Japón, Nueva Zelanda, Noruega, Polonia, Suiza y EE.UU. En general, para aquellos en los que la autorización de exportación general EU001 es válida.

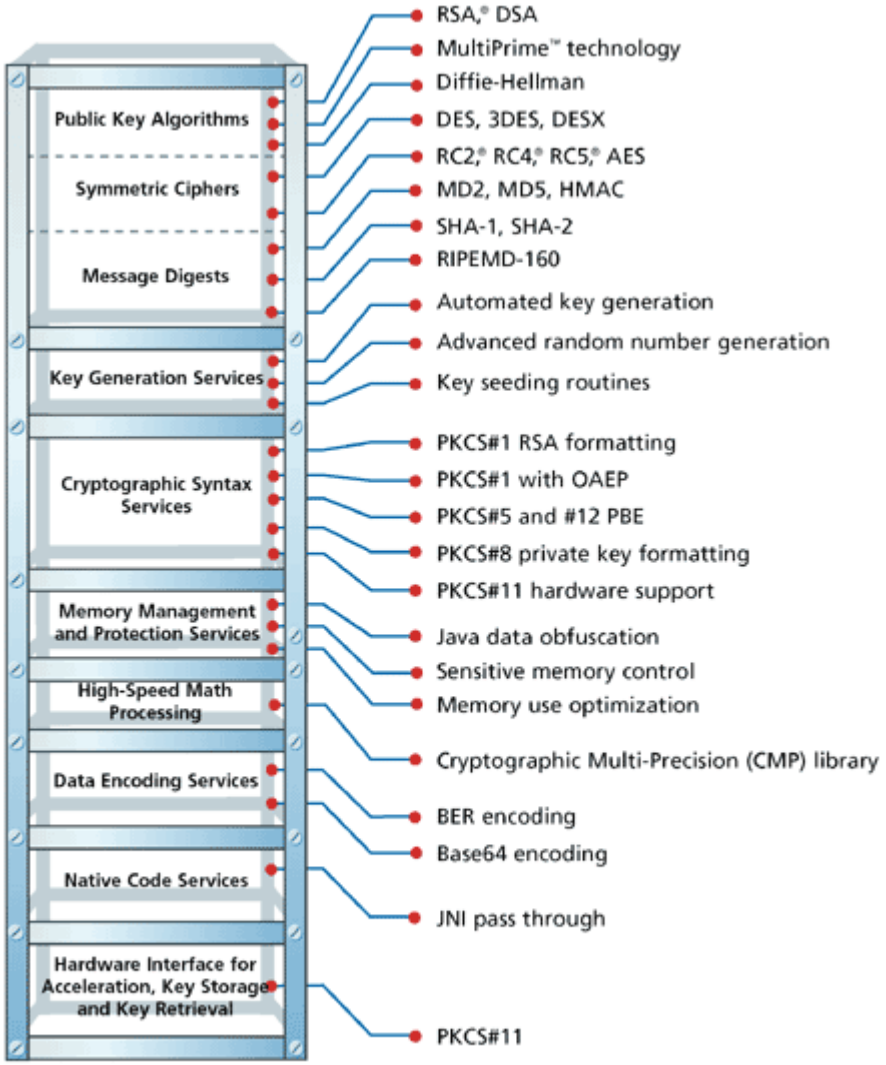
La licencia sería educacional (gratuita) aunque no se tendrían derechos comerciales en la aplicación Mastín y si se deseasen habría que pagar 500€ por una licencia de desarrollo (que incluye 10 licencias de ejecución y 1 licencia de servidor) y otros 100€ por 100 licencias de ejecución. La distribución de programas bajo licencia educacional se habría de efectuar sin distribuir el IAIK-JCE y quien desease ejecutar Mastin debería pagar licencia de ejecución. La obtención de dichas licencias encarecería notablemente la aplicación Mastin, lo cual se alejaría notablemente de nuestros objetivos.

IV.3.4.2. RSA BSAFE Crypto-J

IV.3.4.2.1. Descripción

CSP desarrollado por la empresa *RSA Security, Inc* (http://www.rsasecurity.com/products/bsafe/cryptoj.html), se trata de un **producto comercial** del que nos podemos bajar una versión de evaluación de forma gratuita. Es una adaptación 100% Java de otros productos de la misma empresa.

IV.3.4.2.2. Características



RSA BSAFE Crypto-J Functional Layers

Ilustración 28: Características de RSA BSAFE Crypto-J

IV.3.4.2.3. Conclusión

Al tratarse de un producto comercial, una licencia acarrea un coste económico que se aleja de nuestros objetivos, por lo cual no se escogerá este proveedor.

IV.3.4.3. JCSI

IV.3.4.3.1. Descripción

Java Crypto and Security Implementation (JCSI) es un **producto comercial** de la empresa *Wedgetail*

(http://www.wedgetail.com/jcsi) desde donde que se puede descargar una **versión de evaluación de 30 días**.

IV.3.4.3.2. Características

Las características del proveedor JCSI son las siguientes:

- KeyPairGenerator para RSA, DSA y Diffie-Hellman
- KeyAgreement para Diffie-Hellman (DH) y ESDH (Ephemeral-Static DH)
- Signature para SHAwithRSA, MD5withRSA, MD2withRSA, RIPEMD160withSHA, SHAwithDSA, RawRSA y RawDSA
- Cipher para RSA y RC4r
- KeyGenerator para RSA, AES, IDEA, DESede, Blowfish, RC4, RC2 y DES
- Cipher para AES, IDEA, DESede, Blowfish, RC2 y DES en los modos ECB, CBC y CFB con PKCS5Padding, NoPadding y Zeroes
- MAC para HMACwithSHA1, HMACwithMD5
- MessageDigest para SHA-1, SHA-256, SHA-384, SHA-512, MD5, MD2, MD4 y RIPEMD-160
- Cipher, KeyFactory y AlgorithmParameters para los algoritmos de encriptación basada en contraseña (PBE) PBEwithMD5andDES-CBC y los algoritmos PBE definidos para PKCS#12
- Soporte para AES.

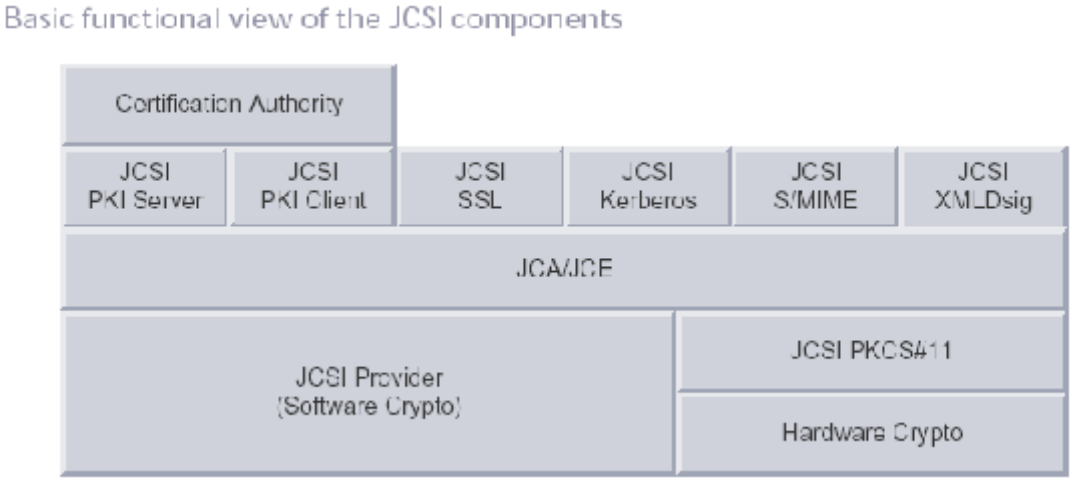


Ilustración 29: Vista básica funcional de los componentes de JCSI

IV.3.4.3.3. Conclusión

Al tratarse de un producto comercial, una licencia acarrea un coste económico que se aleja de nuestros objetivos, por lo cual no se escojerá este proveedor.

IV.3.4.4. BeeCrypt para Java

IV.3.4.4.1. Descripción

Es un CSP de *Virtual Unlimited BV* (http://www.virtualunlimited.com) compatible con JCE 1.2, y debido a que es una empresa holandesa, sus productos se pueden exportar a todo el mundo y no sufren restricciones de exportación. Se basa en una **licencia GNU GPL gratuita** que permite sea usada tanto en proyectos de código abierto como en proyectos cerrados comerciales.

IV.3.4.4.2. Características

BeeCrypt para Java en su versión 2.2.0 incluye:

- Fuentes de entropía para inicializar generadores pseudo-aleatorios
- Generadores pseudo-aleatorios: FIPS-186, Mersenne Twister
- Cifradores de bloque: Blowfish
- Funciones hash con clave: MD5/HMAC, SHA-1/HMAC, SHA-256/HMAC
- Funciones hash: MD5, SHA-1, SHA-256
- Acuerdo de claves de Diffie-Hellman
- Esquema de encriptación DHAES
- Esquema de firma de ElGamal (dos variantes)
- Primitivas básicas RSA y generación de pares de claves

Planeado para versiones futuras:

- Curva elíptica (ECDSA, etc.)
- Conformidad con IEEE P1363
- Más cifradores de bloque (Twofish, etc.)
- Más funcione hash (RIPEMD-160, HAVAL, etc.)
- Primitivas de firma y encriptación RSA de PKCS#1

IV.3.4.4.3. Versión

BeeCrypt 2.2.0

IV.3.4.4.4. Conclusión

No se seleccionará este CSP debido a que por el momento **no incorpora las primitivas de encriptación RSA en PKCS#1**.

IV.3.4.5. Cryptix 3 y Cryptix JCE

IV.3.4.5.1. Descripción

Se encuentran disponibles en la página http://www.cryptix.org y ambos productos son **gratuitos tanto para usos comerciales como si no**.

- *Cryptix 3* es una implementación de la antigua JCE 1.1 (la versión actual de JCE es la 1.2.2) y por esta razón no se elegirá
- *Cryptix JCE* es una implementación del JCE 1.2 que trataba de solucionar los problemas de exportación. Sus características son:

JDKs	1.1, 1.2 y 1.3
Ciphers	Blowfish, CAST5, DES, IDEA, MARS, RC2, RC4, RC6, Rijndael, Serpent, SKIPJACK, Square, TripleDES, Twofish
KeyAgreements	Diffie-Hellman
Modes	CBC, CFB-(8, 16, 24, ..., blocksize), ECB, OFB-(blocksize), openpgpCFB
Hashes	MD2, MD4, MD5, RIPEMD-128, RIPEMD-160, SHA-0, SHA-1, SHA-256/384/512, Tiger
MACs	HMAC-MD2, HMAC-MD4, HMAC-MD5, HMAC-RIPEMD-128, HMAC-RIPEMD-160, HMAC-SHA-0, HMAC-SHA-1, HMAC-Tiger
Signatures	RawDSA, RSASSA-PKCS1, RSASSA-PSS
Assymetric ciphers	RSA/PKCS#1, ElGamal/PKCS#1
SecureRandom SPIs	

Tabla 11: Características de Cryptix JCE

IV.3.4.5.2. Conclusión

No se elegirá *Cryptix JCE* ya que **no está firmado** por Sun Microsystems. Además se encuentra aún **en fase de desarrollo** y **sin documentación**.

IV.4. Proyecto OpenSSL

IV.4.1. Descripción

El Proyecto OpenSSL (http://www.openssl.org) surge para desarrollar un conjunto de herramientas para implementar los protocolos SSL (Secure Sockets Layer) v2 / v3 y TLS (Transport Layer Security) v1 al igual que proporcionar una librería criptográfica de alta potencia para propósito general. OpenSSL está gestionado por un grupo de voluntarios de todo el mundo que se comunican gracias a Internet.

OpenSSL está basado en la librería SSLeay desarrollada por *Eric A. Young* y *Tim J. Hudson*.

IV.4.2. Versión

OpenSSL 0.9.6c

IV.4.3. Instalación

Para su instalación, consultar el apartado de instalación de SuSE Linux 8.0 Proffesional.

IV.4.4. Utilización

No será necesaria por parte del administrador de la Autoridad de Certificación (CA).

Sin embargo **puede resultar conveniente al administrador, un conocimiento de las herramientas que ofrece OpenSSL con el fin de un mayor control de las acciones del programa**, de ahí que a continuación se muestre una descripción de ellas.

IV.4.5. openssl

IV.4.5.1. Descripción

El programa openssl es una herramienta de **línea de comandos** para usar las funciones criptográficas de la librería crypto de OpenSSL desde el shell. Puede usarse para:

- Creación de claves RSA, DH y DSA.
- Creación de certificados X.509, CSR (Certificate Sign Request) y CRL (Certificates Revocated List).
- Cálculo de boletines de mensajes (Message Digests)
- Encriptado y desencriptado
- Pruebas con SSL / TLS Servidores y Clientes
- Manejo de S/MIME firmado o correo encriptado

IV.4.5.2. Sinopsis

```
openssl command [ command_opts ] [ command_args ]

openssl [list-standard-commands | list-message-digest-commands | list-cipher-commands ]

openssl no-XXX [ arbitrary options ]
```

IV.4.5.3. Comandos y opciones

IV.4.5.3.1. Comandos estándar

Comando	*Descripción*
asn1parse	Analiza una secuencia ASN.1
ca	Gestión de la CA.
ciphers	Cipher Suite Description Determination.
crl	Gestión de la CRL.
crl2pkcs7	Conversión de CRL a PKCS#7.
dgst	Cálculo del boletín del mensaje.
dsa	Gestión de los datos DSA.
dsaparam	Generación de parámetros DSA.
enc	Codificación con cifradores.
errstr	Conversión del valor numérico de los errores a una cadena.

dhparam	Generación y gestión de parámetros DH.
gendsa	Generación de parámetros DSA.
genrsa	Generación de parámetros RSA.
ocsp	Utilidad de OCSP (Online Certificate Status Protocol).
passwd	Generación de contraseñas troceadas (hashed passwords).
pkcs12	Gestión de datos PKCS#12.
pkcs7	Gestión de datos PKCS#7.
rand	Generación de octetos pseudo-aleatorios.
req	Gestión de CSR X.509.
rsa	Gestión de datos RSA.
rsautl	Utilidad para firmado, verificación, encriptación y desencriptación RSA.
s_client	Cliente SSL / TLS que puede establecer una conexión transparente con un servidor remoto hablando SSL /TLS. Está implementado sólo para pruebas y usa la mayor parte de la librería SSL de OpenSSL.
s_server	Cliente SSL / TLS que puede establecer una conexión transparente con un servidor remoto hablando SSL /TLS. Está implementado sólo para pruebas y usa la mayor parte de la librería SSL de OpenSSL. Proporciona tanto una línea de comandos como una facilidad HTTP para emular un servidor web con SSL/TLS.
s_time	Temporizador de la conexión SSL.
sess_id	Gestión de datos de sesión SSL.
smime	Procesado de correo S/MIME.
speed	Algoritmo de medida de la velocidad.
verify	Verificado de un certificado X.509.
version	Información de la versión de OpenSSL.
x509	Gestión de los datos del certificado X.509.

Tabla 12: Comandos estándar de openssl (se han eliminado los obsoletos)

IV.4.5.3.2. Comandos de resúmen de mensajes

Comando	*Descripción*
md2	Resúmen MD2
md5	Resúmen MD5
mdc2	Resúmen MDC2
rmd160	Resúmen RMD-160
sha	Resúmen SHA
sha1	Resúmen SHA-1

Tabla 13: Comandos de resúmen de mensajes de openssl

IV.4.5.3.3. Comandos para codificación y cifrado

Comando	*Descripción*
base64	Codificación en Base64.
bf	Cifrado Blowfish modo simple
bf-cbc	Cifrado Blowfish modo CBC
bf-cfb	Cifrado Blowfish modo CFB
bf-ecb	Cifrado Blowfish modo ECB
bf-ofb	Cifrado Blowfish modo OFB
cast	Cifrado CAST modo simple
cast-cbc	Cifrado CAST modo CBC
cast5-cbc	Cifrado CAST5 modo CBC
cast5-cfb	Cifrado CAST5 modo CFB
cast5-ecb	Cifrado CAST5 modo ECB
cast5-ofb	Cifrado CAST5 modo OFB
des	Cifrado DES modo simple
des-cbc	Cifrado DES modo CBC
des-cfb	Cifrado DES modo CFB
des-ecb	Cifrado DES modo ECB
des-ede	Cifrado DES modo EDE
des-ede-cbc	Cifrado DES modo EDE-CBC
des-ede-cfb	Cifrado DES modo EDE-CFB
des-ede-ofb	Cifrado DES modo EDE-OFB
des-ofb	Cifrado DES modo OFB
des3	Cifrado Triple-DES
desx	Cifrado DESX
des-ede3	Cifrado DES modo EDE3
des-ede3-cbc	Cifrado DES modo EDE3-CBC
des-ede3-cfb	Cifrado DES modo EDE3-CFB
des-ede3-ofb	Cifrado DES modo EDE3-OFB
idea	Cifrado IDEA modo simple
idea-cbc	Cifrado IDEA modo CBC
idea-cfb	Cifrado IDEA modo CFB
idea-ecb	Cifrado IDEA modo ECB
idea-ofb	Cifrado IDEA modo OFB
rc2	Cifrado RC2 modo simple
rc2-cbc	Cifrado RC2 modo CBC

rc2-cfb	Cifrado RC2 modo CFB
rc2-ecb	Cifrado RC2 modo ECB
rc2-ofb	Cifrado RC2 modo OFB
rc4	Cifado RC4
rc5	Cifrado RC5 modo simple
rc5-cbc	Cifrado RC5 modo CBC
rc5-cfb	Cifrado RC5 modo CFB
rc5-ecb	Cifrado RC5 modo ECB
rc5-ofb	Cifrado RC5 modo OFB

Tabla 14: Comandos de openssl para codificación y cifrado

IV.4.5.3.4. Paso de argumentos de contraseñas

Algunos comandos aceptan argumentos de contraseñas, usando típicamente `–passin` y `–passout` para contraseñas de entrada y salida respectivamente. Estos permiten que las contraseñas se puedan obtener de diferentes fuentes. Ambas opciones toman un único argumento cuyo formato se describen a continuación. Si no se le pasa ninguna contraseña y ésta es requerida, se le preguntará al usuario: será leída del terminal actual con el eco desactivado.

Comando	*Descripción*
pass:*password*	La contraseña actual es password. Ya que la contraseña es visible a las utilidades (como por ejemplo ps en Unix) esta forma debería ser usada sólo cuando la seguridad no es importante.
env:*var*	Obtiene la contraseña de la variable de entorno var. Debido a que el entorno de otros procesos es visible en ciertas plataformas (por ejemplo en algunos Unix) esta opción, al igual que la anterior deben de ser usadas con precaución.
file:*pathname*	La primera línea del archivo referido por pathname es la conrtaseña. Si se suministra el mismo argumento a -passin y a –passout, entonces la primera línea se usará para la contraseña de entrada, y la siguiente línea para la contraseña de salida. pathname podría referirse a un dispositivo o a una pipe con nombre.
fd:*number*	Lee la contraseña del número de descriptor de fichero indicado. Puede ser utilizado para enviar los datos a través de de un pipe por ejemplo.
stdin	Lee la contraseña a través de la entrada estándar.

Tabla 15: Paso de argumentos de contraseñas

IV.4.6. openssl ca

IV.4.6.1. Descripción

El comando ca es una aplicación de CA. Puede ser usada para firmar peticiones de certificados de variadas formas y generar CRLs. También mantiene una base de datos en modo texto de certificados expedidos y su estado.

IV.4.6.2. Sinopsis

```
openssl ca [-verbose] [-config filename] [-name section] [-gencrl] [-revoke
file] [-subj arg] [-crldays days] [-crlhours hours] [-crlexts section] [-
startdate date] [-enddate date] [-days arg] [-md arg] [-policy arg] [-keyfile
arg] [-key arg] [-passin arg] [-cert file] [-in file] [-out file] [-notext] [-
outdir dir] [-infiles] [-spkac file] [-ss_cert file] [-preserveDN] [-
noemailDN] [-batch] [-msie_hack] [-extensions section] [-extfile section]
```

IV.4.6.3. Opciones

Opción	*Descripción*
-config ***filename***	especifica el fichero de configuración que se usará.
-name ***section***	especifica la sección del fichero de configuración que se usará (sobreescribe default_ca en la sección ca).
-in ***filename***	un fichero de entrada que contiene una única petición de certificado para ser firmado por la CA.
-ss_cert ***filename***	un único certificado autofirmado para ser firmado por la CA.
-spkac ***filename***	un archivo que contiene un único "signed public key and challenge" (SPKAC) de Netscape y valores de campos adicionales para ser firmado por la CA. Ver la sección de NOTES para ver información del formato requerido.
-infiles	si está presente, debería ser la última opción. Todos los argumentos posteriores son asumidos como los nombres de los archivos que contienen las peticiones de certificados.
-out ***filename***	el archivo de salida. Por defecto se usa la salida estándar. Los detalles del certificado también se escribirán a ese archivo.
-outdir ***directory***	el directorio de salida de los certificados. El certificado se escribirá a un archivo consistente del numero de serie en hexagesimal terminados con ".pem"
-cert	el archivo del certificado de la CA.
-keyfile ***filename***	la clave privada con la que firmar la petición.

Opción	*Descripción*
-key *password*	la contraseña utilizada para encriptar la clave privada. Como en algunos sistemas operativos la línea de argumentos es visible (p.e. Unix con la utilidad 'ps'), esta opción debería ser usada con precaución.
-passin *arg*	la fuente de la clave de la contraseña. Para más información sobre el formato de arg ver la sección PASS PHRASE ARGUMENTS en openssl(1).
-verbose	escribe detalles extra sobre las operaciones que se efectúan.
-notext	no saca el certificado en forma de texto al fichero de salida.
-startdate *date*	permite ajustar explícitamente la fecha de comienzo. El formato de la fecha es YYMMDDHHMMSSZ (igual que la estructura UTCTime de ASN1).
-enddate *date*	permite ajustar explícitamente la fecha de expiración.El formato de la fecha es YYMMDDHHMMSSZ (igual que la estructura UTCTime de ASN1).
-days *arg*	número de días de validez del certificado.
-md *alg*	el digestor de mensajes (message digest) a usar. Los posibles valores incluyen md5, sha1 y mdc2. Esta opción también se aplica a los CRLs.
-policy *arg*	esta opción define la política de la CA a usar. Esta es una sección en el archivo de configuración que decide qué campos deberían ser obligatorios o coincidir con el certificado de la CA. Revisa la sección FORMATO DE POLÍTICA para más información.
-msie_hack	esta es una opción heredada para hacer que ca funcione con versiones muy antiguas del control de alistamiento de certificados del IE Usa UniversalStrings para casi todo. Como el antiguo control tiene varios bugs de seguridad, su uso está desaconsejado. El más reciente control "Xenroll" no necesita esta opción.
-preserveDN	Normalmente el orden del DN de un certificado es el mismo que el orden de los campos en la pertinente sección de la política. Cuando esta opción está activada el orden es el mismo que el de la petición. Esta opción se usa por compatibilidad con el antiguo control de alistamiento que sólo aceptaba certificados cuyos DNs coincidían con el orden de la petición Esto no es necesario para "Xenroll".

Opción	*Descripción*
-noemailDN	El DN de un certificado puede contener el campo EMAIL si se presenta en el DN de la petición. Sin embargo, es una buena política tener el e-mail incorporado sólo en la extensión altName del certificado. Cuando esta opción está activada, el campo EMAIL se quita del certificado en cuestión y se incorpora sólo en las extensiones eventualmente presentes. La clavee email_in_dn puede ser usada en el archivo de confguración para activar este comportamiento.
-batch	esto activa el modo en batería. En este modo no habrá pregustas y todos los certificados se certificarán automáticamente.
-extensions *section*	la sección del archivo de configuración que contiene las extensiones del certificado para ser añadidas cuando se expide un certificado (por defecto a x509_extensions a menos que se use la opción -extfile. Si so está presente la sección de extensiones, entonces se creará un certificado V1. Si la sección de extensiones está presente (incluso si está vacía), entonces se creará un certificado V3.
-extfile *file*	un archivo de configuración adicional para obtener las extensiones del certificado (usando la sección por defecto a menos que también se ue la opción -extensions).

Tabla 16: Opciones de openssl ca

IV.4.6.4. Opciones de CRL

Opción	*Descripción*
-gencrl	esta opción genera un CRL basado en la información del fichero índice.
-crldays *num*	el número de días antes de que venza el siguiente CRL. Esto es, los días desde hoy que se ponen en el campo nextUpdate del CRL.
-crlhours *num*	el número de horas antes de que venza en siguiente CRL.
-revoke *filename*	un fichero que contiene un certificado a revocar.
-subj *arg*	sobreescribe el nombre del tema dado en la petición. arg debe ser formateado como /type0=value0/type1=value1/type2=.... Los caracteres pueden ser "escapados" por un \ y los espacios son omitidos.

Opción	*Descripción*
-crlexts *section*	La sección del fichero de configuración que contiene extensiones del CRL a incluir. Si la sección de extensiones del CRL no está presente, entonces se crea un CRL V1. Si la sección de extensiones del CRL está presente (incluso si está vacía), entonces se crea un CRL V2. Las extensiones CRL expecificadas son extensiones CRL y not entradas de extensiones CRL. Debería destacarse que algún software (por ejemplo Netscape) no puede manejar CRL V2.

Tabla 17: Opciones de CRL de openssl ca

IV.4.6.5. Opciones del archivo de configuración

La sección del archivo de configuración que contiene las opciones para la ca se encuentra como sigue: Si se usa la opción -name en la línea de comandos, entonces nombra la sección a ser usada. De otra manera, la sección a ser usada debe ser nombrada en la opción default_ca de la sección ca del archivo de configuración (o en la sección por defecto del archivo de configuración). Además de default_ca, las siguientes opciones son leidas directamente de la sección ca: RANDFILE protege de msie_hack Con la excepción de RANDFILE, este es probablemente un bug y puede cambiar en futuras versiones.

Algunas de las opciones del archivo de configuración son idénticas a las opciones de la línea de comandos. Si la opción está presente en el archivo de configuración y en la línea de comandos, se usa el valor de la línea de comandos. Si una opción está descrita como obligatoria, entonces debe estar presente en el archivo de configuración o en la línea de comandos.

Opciones	*Descripción*
oid_file	Esto especifica un archivo que contiene adicionales OBJECT IDENTIFIERS. Cada línea del archivo consta de la forma numérica del identificador del objeto seguida de un espacio en blanco, después un nombre corto seguido de un espacio en blanco y finalmente un nombre largo.
oid_section	Esto especifica una sección en el archivo de configuración que contiene identificadores de objeto extra. Cada línea debería de constar del nombre corto del identificador del objeto seguido de un = y la forma numérica. Los nombres cortos y los largos son los mismos si se usa esta opción.
new_certs_dir	igual que la opción de línea de comandos -outdir. Especifica el directorio donde se pondrán los nuevos certificados. Obligatorio.
certificate	igual que -cert. Da el archivo que contiene el certificado de la CA. Obligatorio.
private_key	igual que la opción -keyfile. El archivo que contiene la clave privada de la CA. Obligatorio.

Opciones	*Descripción*
RANDFILE	un archivo usado para leer y escribir información semilla para números aleatorios, o un socket EGD (ver RAND_egd(3)).
default_days	igual que la opción -days. El número de días que se certifica el certificado.
default_startdate	igual que la opción -startdate. La fecha de inicio para certificar un certificado. Si no se especifica, se usa la fecha actual.
default_enddate	igual que la opción -enddate. Tanto esta opción como default_days (o la línea de comando equivalente) debe estar presente.
default_crl_hours **default_crl_days**	igual que las opciones -crlhours y -crldays. Estas deben ser sólo usadas si ninguna de las dos está presenteal generar la CRL.
default_md	igual que la opción -md. El digestor de mensajes a usar. Obligatorio.
database	el archivo de la base de datos de texto a usar. Obligatorio. Este archivo debe estar presente aunque inicialmente está vacío.
serialfile	un fichero de texto que contiene siguiente número de serie en hexagesimal para usar. Obligatorio. Este archivo debe estar presente y contener un númer de serie válido.
x509_extensions	igual que -extensions.
crl_extensions	igual que -crlexts.
preserve	igual que -preserveDN
email_in_dn	igual que -noemailDN. Si quieres quiter el campo EMAIL del DN del certificado, simplemente fíjalo a 'no'. Si no está presente, por pefecto se permite el campo EMAIL en el DN del certificado.
msie_hack	igual que -msie_hack
policy	igual que -policy. Obligatorio. Ver la sección FORMATO DE POLÍTICA para más información.

Opciones	*Descripción*
nameopt **certopt**	estas opciones permiten el formato usado para mostrar detalles el certificado cuando se pide al usuario que confirme la firma. Todas las opciones soportadas por las opciones x509 -nameopt y -certopt pueden ser usadas aquí excepto no_signame y no_sigdump que están activadas permanentemente y no pueden ser desactivadas (esto es a causa de que la firma del certificado no puede ser mostrada ya que el certificado no ha sido firmado por el momento). Por conveniencia, los valores de default_ca son aceptados por ambos para producir una salida sensata. Si ninguna de las dos opciones está presente, se usa el formato usado en versiones previas de OpenSSL. El uso del antiguo formato está muy desaconsejado ya que sólo muestra los campos mencionados en la sección de política, maltrata los tipos de cadena y no muestra las extensiones.
copy_extensions	determina cómo se debería manejar las extensiones en la petición de certificado. Si se especifica none o esta sección no está presente, entonces la extensiones son ignoradas y no se copian en el certificado. Si se especifica copy entonces cualquier extensión presente en la petición que no estén ya presentes en el certificado, será copiado en el certificado. Si se especifica a copyall entonces todas las extensiones en la petición serán copiadas al certificado: si la extensión ya está presente,en el certificado, será borrada antes. Ver la sección de ADVERTENCIAS antes de usar esta opción. El principal uso de esta opción es permitir que las peticiones de certificados sustituyan ciertos valores para ciertas extensiones como subjectAltName.

Tabla 18: Opciones del archivo de configuración de openssl ca (/usr/share/ssl/openssl.cnf)

IV.4.6.6. Formato de la política

La sección de política consta de un conjunto de variables correspondientes a campos del DN del certificado. Si el valor es match, entonces el valor del campo debe ser el mismo que en el certificado de la CA. Si el valor es supplied, entonces debe estar presente. Si el valor es optional entonces puede estar presente o no. Cualquier campo no mencionado en la sección de política es borrado en silencio a menos que la opción -preserveDN esté activada aunque esto puede ser originado más por una casualidad que por un comportamiento intencionado.

IV.4.6.7. Restricciones

No es posible certificar dos certificados con el mismo DN: este es un efecto lateral de cómo está indexada una base de datos en modo texto y no puede ser fácilmente arreglada sin introducir otros problemas.

IV.5. Apache httpd

IV.5.1. Descripción

Apache httpd (http://httpd.apache.org) es el **servidor HTTP** de Apache Software Foundation (http://www.apache.org). Surge del esfuerzo colaborativo de voluntarios con el ánimo de crear una robusta, de calidad comercial y libremente disponible implementación de código abierto de un servidor HTTP (web). Apache ha sido el servidor web más popular en Internet desde Abril de 1996.

Apache httpd es un servidor HTTP basado en el servidor de la NCSA verión 1.3 (o 1.4). Arregla numerosas fallas de dicho servidor incluyendo nuevas características y tiene un API que permite ser adaptado a las necesidades de los usuarios.

Junto con el módulo mod-ssl se convierte en un servidor HTTPS.

Se puede encontrar más información en la ayuda del propio paquete apache (/usr/local/httpd/htdocs/) y en la documentación que trae el paquete apache-doc (/usr/local/httpd/htdocs/manual/).

IV.5.2. Versión

apache 1.3.23-73

IV.5.3. Instalación

Para su instalación, consultar el apartado de instalación de SuSE Linux 8.0 Proffesional.

IV.5.4. Utilización

Se utiliza implícitamente en la utilización de Jakarta.

IV.6. Jakarta-Tomcat

IV.6.1. Descripción

El Proyecto Jakarta (http://jakarta.apache.org) es un servidor perteneciente al Proyecto Apache (http://www.apache.org) y basado en éste que permite implementar **Java en el lado del servidor**.

Tomcat (http://jakarta.apache.org/tomcat) es un **contenedor de Servlet/JSP** desarrollado en el Proyecto Jakarta. Tomcat 4 implementa las especificaciones Servlet 2.3 y Java Server Pages (JSP) 1.2 de Java Software e incluye algunas características adicionales que hacen de ella una útil plataforma para desarrollar y desplegar aplicaciones y servicios web.

Para más información dirigirse a /opt/jakarta/webapps/tomcat-docs.

IV.6.2. Versión

Tomcat 4.0.1-127

IV.6.3. Instalación

Para su instalación, consultar el apartado de instalación de SuSE Linux 8.0 Proffesional.

IV.6.4. Configuración

Para hacer que sea un servidor seguro mediante el protocolo HTTPS, primero debemos crear una clave RSA (en este caso de 2048 bits) en un keystore con el alias `tomcat` de la siguiente manera:

```
gab:~ # /usr/java/j2sdk1.4.0_02/bin/keytool -genkey -alias tomcat -keyalg RSA
-keystore /opt/jakarta/conf/.keystore -keysize 2048
Escriba la contraseña del almacén de claves:  miclave
¿Cuáles son su nombre y su apellido?
  [Unknown]:  192.168.0.3
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  Departamento de Telematica
¿Cuál es el nombre de su organización?
  [Unknown]:  Universidad de Sevilla
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:  Sevilla
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:  Spain
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  ES
¿Es correcto CN=192.168.0.3, OU=Departamento de Telematica, O=Universidad de
Sevilla, L=Sevilla, ST=Spain, C=ES?
  [no]:  y

Escriba la contraseña clave para <tomcat>
        (INTRO si es la misma contraseña que la del almacén de claves):
```

Notar que cuando se pregunta el nombre y apellido, en realidad se pregunta acerca del CommonName (CN) y ahí se ha de introducir el nombre de la máquina a la que se accede de forma "literal", es decir, que es distinto poner 192.168.0.3, www.us.es o us.es, por ejemplo.

Para crear a partir de la CSR X.509 en formato PEM siguiendo el estándar PKCS#10:

```
gab:~ # /usr/java/j2sdk1.4.0_02/bin/keytool -certreq -alias tomcat -keystore
/opt/jakarta/conf/.keystore -file tomcat.csr
Escriba la contraseña del almacén de claves:  miclave
```

Una vez remitida a la CA (que puede ser la nuestra) y firmada por ésta, obtenemos el certificado (tomcat.cer).

Se importa el certificado a la keystore bajo el mismo alias de tomcat:

```
gab:/opt/jakarta/conf # /usr/java/j2sdk1.4.0_02/bin/keytool -import -alias
tomcat -file ./tomcat.cer -keystore /opt/jakarta/conf/.keystore
Se ha añadido el certificado al almacén de claves
```

Para configurar el servidor, hay que especificar dónde está localizada la keystore que contiene el alias tomcat. Para ello modificamos `SSL HTTP/1.1 Connector` del archivo de configuración `/opt/jakarta/conf/server.xml`:

```
    <!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
    <Connector className="org.apache.catalina.connector.http.HttpConnector"
               port="8080" minProcessors="5" maxProcessors="75"
```

```
               enableLookups="true" redirectPort="8443"
               acceptCount="10" debug="0" connectionTimeout="60000"/>
    <!-- Note : To disable connection timeouts, set connectionTimeout value
     to -1 -->

   <!-- Define an SSL HTTP/1.1 Connector on port 8443 -->
   <!--
   <Connector className="org.apache.catalina.connector.http.HttpConnector"
               port="8443" minProcessors="5" maxProcessors="75"
               enableLookups="true"
             acceptCount="10" debug="0" scheme="https" secure="true">
      <Factory className="org.apache.catalina.net.SSLServerSocketFactory"
               clientAuth="false" protocol="TLS"/>
    -->
    </Connector>
```

Resultando este otro código:

```
    <!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
    <Connector className="org.apache.catalina.connector.http.HttpConnector"
               port="80" minProcessors="5" maxProcessors="75"
               enableLookups="true" redirectPort="8443"
               acceptCount="10" debug="0" connectionTimeout="60000"/>
    <!-- Note : To disable connection timeouts, set connectionTimeout value
     to -1 -->

    <!-- Define an SSL HTTP/1.1 Connector on port 8443 -->

    <Connector className="org.apache.catalina.connector.http.HttpConnector"
               port="443" minProcessors="5" maxProcessors="75"
               enableLookups="true"
             acceptCount="10" debug="0" scheme="https" secure="true">
      <Factory className="org.apache.catalina.net.SSLServerSocketFactory"
               clientAuth="false" keystoreFile="/opt/jakarta/conf/.keystore"
keystorePass="miclave" protocol="TLS" />
    </Connector>
```

Donde se ha cambiado el puerto HTTP y HTTPS han pasado de ser 8080 y 8443 en la configuración por defecto respectivamente a 80 y 443 que son los típicos para HTTP y HTTPS respectivamente. Además se ha quitado el comentario (<!--- y -->) al `SSL HTTP/1.1 Connector` y se especifica el fichero donde se guarda el keystore y la clave que utiliza al añadirle el código `keystoreFile="/opt/jakarta/conf/.keystore" keystorePass="miclave"` al Factory. Para mayor información, consultar `/opt/jakarta/webapps/tomcat-docs/ssl-howto.html`

Las clases de los servlets están en `/opt/jakarta/webapps/examples/WEB-INF/classes/`. Y los JSPs están en `/opt/jakarta/webapps/examples/jsp`. La propia distribución de Tomcat trae el archivo (que se debe incorporar al CLASSPATH) `/opt/jakarta/common/lib/servlet.jar` que nos proporciona las clases necesarias para poder compilar los servlets que creemos.

IV.6.5. Utilización

Para **arrancar el servidor** standalone jakarta-tomcat debemos ejecutar el script que nos proporciona la propia instalación a tal efecto:

```
gab:~ # /opt/jakarta/bin/startup.sh
```

Y cuando queramos **cerrar el servidor** ejecutaremos:

```
gab:~ # /opt/jakarta/bin/shutdown.sh
```

IV.7. mutt

IV.7.1. Descripción

Mutt (http://www.mutt.org) es un **cliente de correo MIME basado en texto** pequeño pero muy potente. Mutt es altamente configurable con características avanzadas como asociaciones de teclas, hilos de correo, búsqueda de expresiones regulares y lenguaje de concordancia de patrones para seleccionar grupos de mensajes.

IV.7.2. Versión

mutt 1.3.27i-62

IV.7.3. Instalación

Para su instalación, consultar el apartado de instalación de SuSE Linux 8.0 Proffesional.

IV.7.4. Utilización

No será necesaria por parte del administrador de la Autoridad de Certificación (CA).

IV.8. Otras aplicaciones de utilidad

He aquí una lista de otros programas que han sido de utilidad tanto en la creación como documentación y memoria de este Proyecto:

Aplicación	*S.O.*	*Descripción*
KDE 3.0	Linux	Entorno gráfico. Podría haber sido cualquier otro en Linux. Facilita enormemente el trabajo.
OpenOffice.org 1.0.1	Linux /Windows	Procesador de texto que tiene la ventaja de que sirve tanto para Linux como para Windows con lo que facilitamos el trabajo al pasar de un sistema operativo a otro.
Web Konqueror 3.0	Linux	Navegador. Similiar al cómodo explorador de Windows
Netscape Navigator 6.0	Linux /Windows	Navegador. Para comprobar los resultados. Tiene la ventaja de que funciona tanto en Linux como en Windows con lo que tras crear el código (en Linux) podemos ver seguidamente los resultados.
Internet Explorer 6.0	Windows	Navegador. Para comprobar los resultados. Tiene el inconveniente de funcionar solamente en Windows por lo que es necesaria al menos una red de área local para poder utilizarlo.
Konsole 1.1	Linux	Consola de la línea de comandos muy útil y necesaria para por ejemplo compilar, ya que nos encontramos normalmente en un entorno de ventanas como es el KDE.
KWrite 4.0	Linux	Editor de texto muy cómodo para KDE. Resalta con colores las diferentes partes del código tanto en Java, HTML, ...
JSDK 1.4.0	Linux /Windows	Paquete de desarrollo estándar de Java. Es necesario para poder crear aplicaciones y poder ejecutarlas. Si no necesitásemos crearlas, sino solamente ejecutarlas, nos bastaría con el Entorno de Ejecución de Java (JRE 1.4.0)
Adobe Acrobat	Windows	
Ark 2.1.9	Linux	Herramienta de archivado de KDE. Similar al popular WinZip de Windows, para descomprimir archivos .zip y poder ver el contenido de los archivos .jar
KCalc 1.3.1	Linux	Calculadora para KDE. Útil para hacer conversiones entre representaciones decimales y hexagesimales y también pequeños cálculos.
KSnapshot	Linux	Herramienta para capturar la pantalla. Útil a la hora de realizar la ayuda

Tabla 19: Otras aplicaciones de utilidad en este Proyecto

V. Mastin y XMastin

V.1. Mastin

V.1.1. Descripción

Aplicación de línea de comandos que encripta o desencripta ficheros a partir de la clave extraida de un KeyStore creado por la herramienta keytool de Java

El algoritmo que se usa es el algoritmo asimétrico RSA inventado por Rivest, Adleman y Shamir y que viene recogido en el PKCS#1.

El modo de funcionamiento será ECB (Electronic Code Book) y el relleno será OAEP (Optimal Asymmetric Encryption Padding) definido en el PKCS#1 v2.

La clave, ya sea pública o privada, se extraerá de la entrada en un KeyStore definida por un alias

Se podrá especificar la ruta del KeyStore (`-KeyStorePath` ***`keyStorePath`***), su tipo (`-KeyStoreType` ***`keyStoreType`***), y su clave (`-KeyStorePassword` ***`keyStorePassword`***).

Si se desea utilizar un alias distinto a "mykey", se deberá especificar (`-Alias` ***`alias`***) así como su clave (`-KeyPassword` ***`keyPassword`***) en caso a que sea distinta a la del KeyStore (`-KeyStoreType` ***`keyStoreType`***)

Al estar escrito en Java, se puede ejecutar en cualquier Sistema Operativo, siendo testados tanto en Windows (XP) como en Linux (SuSE Linux 8.0).

Creado por Gabriel Babiano Huete (gabrielbabiano@yahoo.es) para su Proyecto Fin de Carrera titulado "*Estudio e implementación de una Autoridad de Certificación*".

V.1.2. Estructura interna

Mastin v1.0 utiliza como CSP a BouncyCastle (BC) a través del cifrador `javax.crypto.Cipher` tras haber sido declarada con `java.security.Security`.

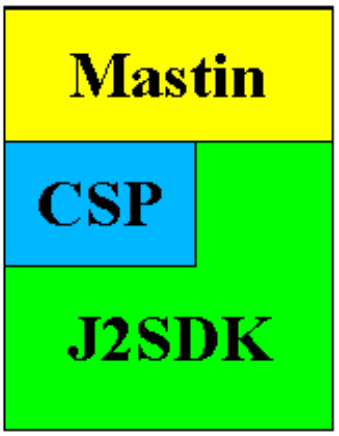


Ilustración 30: Estructura de bloques de Mastin

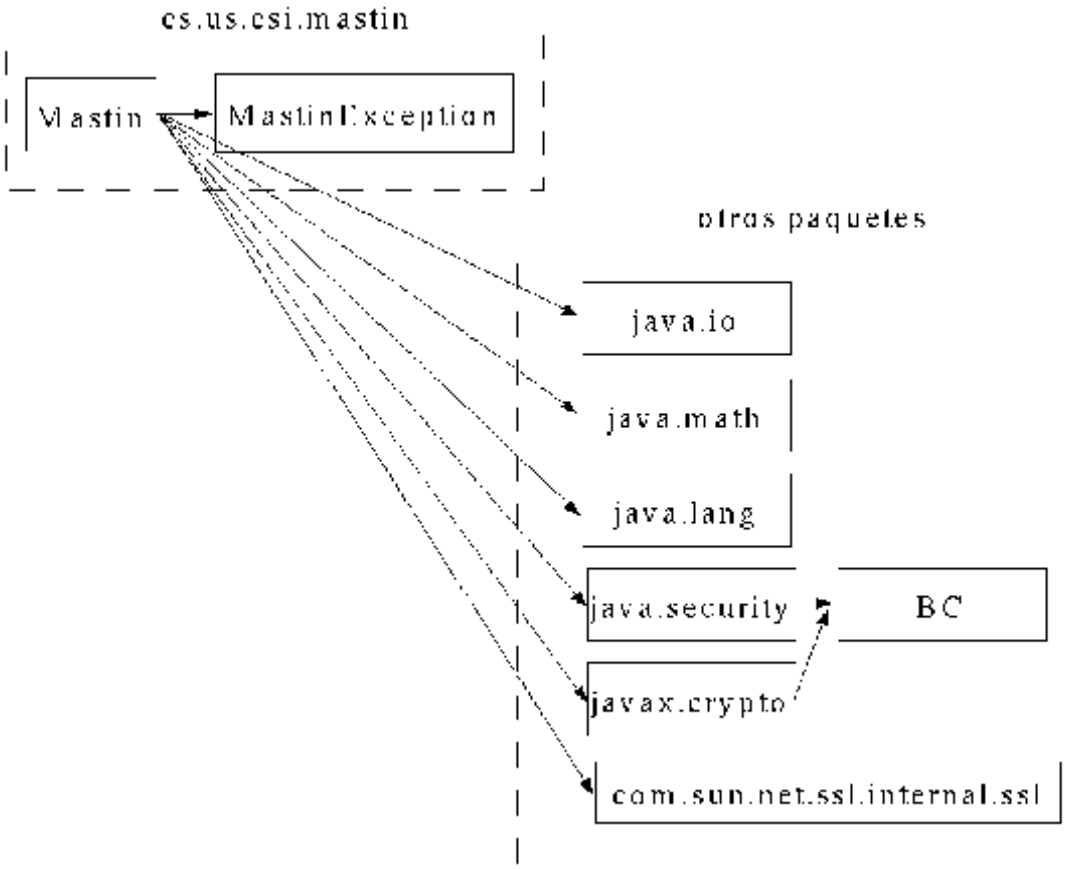


Ilustración 31: Diagrama de clases de Mastin

`MastinException` es una clase que me permite lanzar las excepciones provocadas en la aplicación Mastin.

El paquete `com.sum.net.ssl.internal.ssl` contiene las clases necesarias para poder extraer las claves públicas o privadas del KeyStore.

V.1.3. API

Además del método main, la API de Mastin v1.0 contiene otras funciones. A destacar:

- `getKeyFromKeyStore`: que básicamente obtiene desde una keystore especificada, una clave privada en caso de que el alias que se especifica corresponda a una entrada de tipo clave o una clave pública en caso de que dicho alias corresponda a una entrada de tipo certificado.
- `workWithFiles`: que básicamente encripta o desencripta según el modo especificado con el algoritmo RSA, en modo ECB y con relleno OAEP un fichero de entrada en otro de salida

Para mayor información se recomienda consultar la API que se encuentra en formato HTML (/mastin/Mastin_v1.0/docs/api/index.html) o bien el propio código comentado profusamente.

V.1.4. Versión

Mastin v1.0

V.1.5. Archivos

Nombre del archivo	*Descripción*
Mastin.java	Fuente de la clase Mastin
Mastin.class	Bytecode de la clase Mastin
MastinException.java	Fuente de la clase MastinException
MastinException.class	Bytecode de la clase MastinException
XMastin.java	Fuente de la clase XMastin
XMastin.class	Bytecode de la clase XMastin
instalarMastin.sh	Shell script que instala la aplicación Mastin y XMastin en Linux
instalarMastin.bat	Ejecutable por lotes que instala la aplicación Mastin y XMastin en Windows
desinstalarMastin.sh	Shell script que desinstala la aplicación Mastin y XMastin en Linux
desinstalar Mastin v1.0 y XMastin v1.0.bat	Ejecutable por lotes que desinstala la aplicación Mastin y XMastin en Windows
index.html	Manual de usuario de Mastin v1.0 y XMastin v1.0

Tabla 20: Descripción de los principales archivos de las aplicaciones Mastin v1.0 y XMastin v1.0

V.1.6. Software requerido

Consultar el Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

V.1.7. Instalación

Consultar el Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

V.1.8. Utilización

Consultar el Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

V.1.9. Desinstalación

Consultar el Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

V.1.10. Pruebas

Se han realizado pruebas de rendimiento con el siguiente equipo:

Hardware	
Máquina:	PC clónico
Procesador:	Athlon K7 1200MHz
Memoria RAM:	256Mb
Software	
Sistema Operativo:	SuSE Linux 8.0 Proffesional
Entorno Gráfico:	K.D.E. 3.0
Java:	Java 2 SDK 1.4.0_02
Cryptographic Service Provider:	Bouncy Castle Provider 1.15

Tabla 21: Equipo de pruebas de rendimiento de Mastin v1.0

Tamaño fichero entrada (bytes)	***Tamaño fichero salida (bytes)***	***Tiempo requerido (s)***		***Rendimiento entrada (bytes/s)***		***Rendimiento salida (bytes/s)***	
		clave publica	clave privada	clave publica	clave privada	clave publica	clave privada
52106	62464	1	41	52106	1270	**62464**	**1523**
111271	133120	2	89	55635	1250	**66560**	**1495**
203036	242944	5	162	40607	1238	**48588**	**1499**
306412	366592	7	243	43773	1260	**52370**	**1508**

Tamaño fichero entrada (bytes)	*Tamaño fichero salida (bytes)*	*Tiempo requerido (s)*		*Rendimiento entrada (bytes/s)*		*Rendimiento salida (bytes/s)*	
		clave publica	clave privada	clave publica	clave privada	clave publica	clave privada
415552	497152	11	332	37777	1251	**45195**	**1497**
559607	669440	13	447	43046	1251	**51495**	**1497**
824922	986880	21	691	39282	1193	**46994**	**1428**
1042000	1246720	27	811	38592	1284	**46174**	**1537**
1317903	1576704	36	1054	36608	1250	**43797**	**1495**

Tabla 22: Resultado de las pruebas de rendimiento de Mastin v1.0 para una clave de 2048 bits tanto en encriptación como en desencriptación

Los resultados son prácticamente idénticos para encriptación y para desencriptación con el mismo tipo de clave. Las pequeñas variaciones existen entre los resultados dependen del resto de las tareas que se ejecutan en paralelo en el PC.

El tiempo requerido es independiente del tamaño del fichero de entrada, pero sí depende del número de bloques encriptados.

V.2. XMastin

V.2.1. Descripción

Aplicación para un entorno gráfico escrita en Java que utiliza el API suministrado por Mastin para realizar las mismas funciones que éste.

Al estar escrito en Java, se puede ejecutar en cualquier sistema operativo, siendo testados tanto en Windows (XP) como en Linux (SuSE Linux 8.0). El LookAndFeel que utiliza XMastin es aquél que permite tener el mismo aspecto bajo cualquier sistema operativo.

Creado por Gabriel Babiano Huete (gabrielbabiano@yahoo.es) para su Proyecto Fin de Carrera titulado "Estudio e implementación de una Autoridad de Certificación".

V.2.2. Estructura interna

Como se ha comentado anteriormente, XMastin utiliza las siguientes funciones del API de Mastin:

- `getKeyFromKeyStore`: que básicamente obtiene desde una keystore especificada, una clave privada en caso de que el alias que se especifica corresponda a una entrada de tipo clave o una clave pública en caso de que dicho alias corresponda a una entrada de tipo certificado.
- `workWithFiles`: que básicamente encripta o desencripta según el modo especificado con el algoritmo RSA, en modo ECB y con relleno OAEP un fichero de entrada en otro de salida

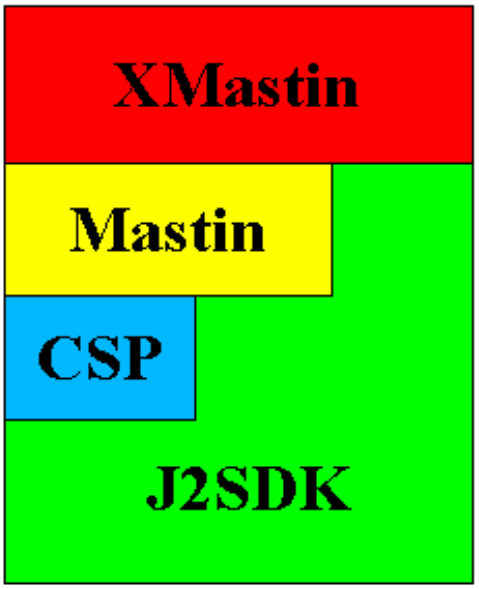


Ilustración 33: Estructura de bloques de XMastin

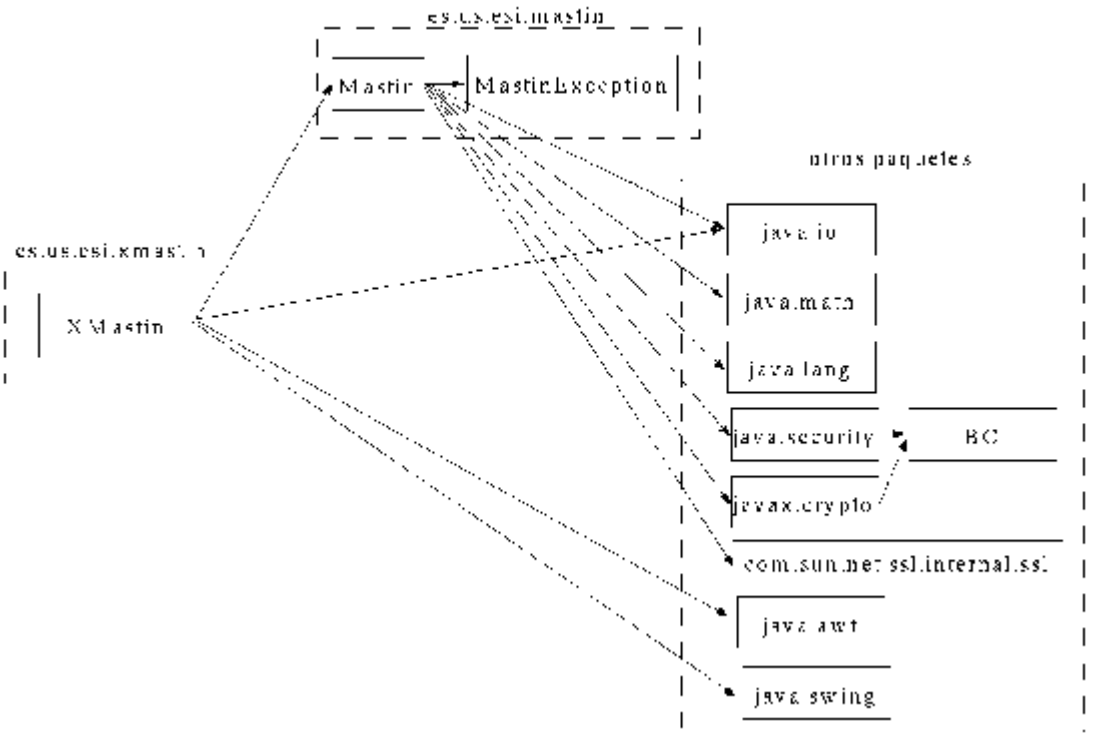


Ilustración 32: Diagrama de clases de XMastin

El paquete `com.sum.net.ssl.internal.ssl` contiene las clases necesarias para poder extraer las claves públicas o privadas del KeyStore.

V.2.3. Versión

XMastin v.10

V.2.4. Software requerido

Consultar el Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

V.2.5. Instalación

Consultar el Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

V.2.6. Utilización

Consultar el Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

V.2.7. Desinstalación

Consultar el Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

VI. Autoridad de Certificación (CA)

VI.1. Autoridad de Certificación (CA)

VI.1.1. Descripción

Conjunto de páginas HTML y servlets escritos en Java que vía HTTPS tratan de implementar una autoridad de certificación. Se acompañan de una serie de shell scripts comentados para la simplificación de las tareas del administrador. Está basado en utilidades proporcionadas por el Proyecto OpenSSL. Creado por Gabriel Babiano Huete (gabrielbabiano@yahoo.es) para su Proyecto Fin de Carrera titulado *"Estudio e implementación de una Autoridad de Certificación"*.

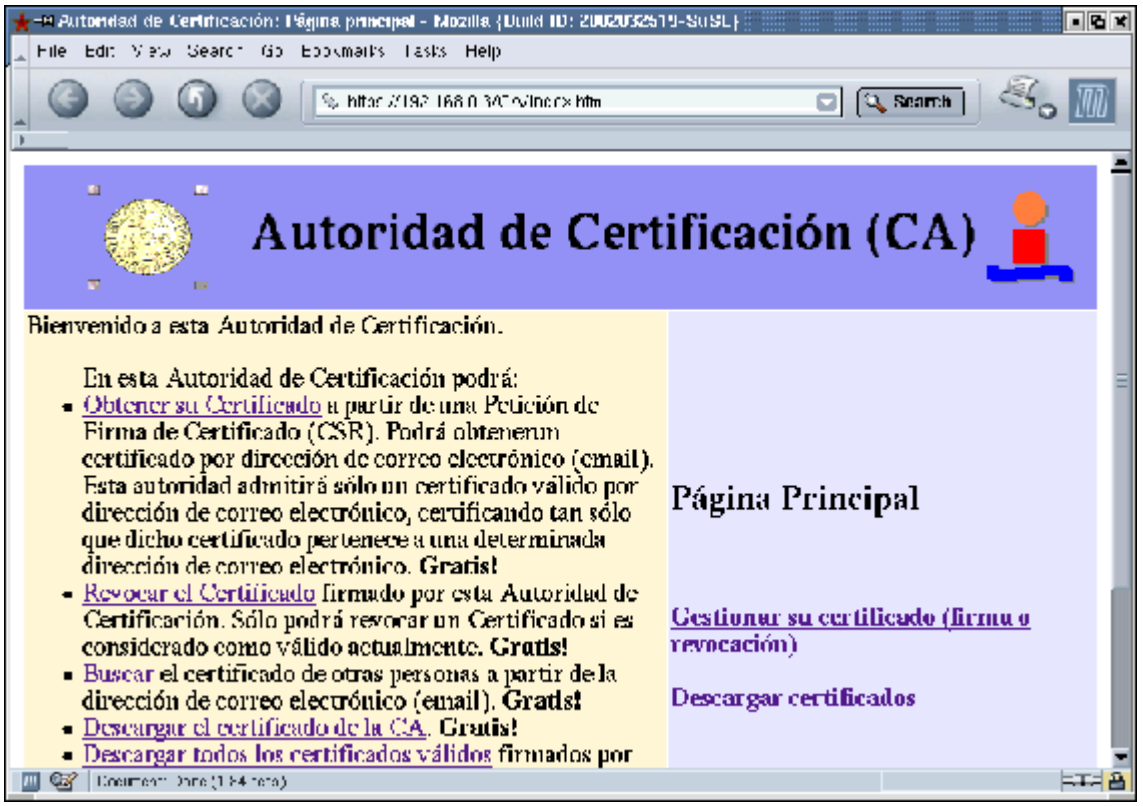


Ilustración 34: Vista de la página principal de la CA

VI.1.2. Características

Esta autoridad implementa las siguientes funciones:

- **Certificación** inmediata de Peticiones de Firma de Certificado (CSR) según el estándar PKCS#10. Sólo tendrá validez sólo un certificado por dirección de correo electrónico y por DistiguishedName (DN).
- **Revocación** de certificados firmados por esta autoridad. La Lista de Certificados Revocados (CRL) se actualizará automáticamente tras cada revocación.
- **Búsqueda** y entrega de certificados válidos de usuarios. Disponible según el estándar X.509 o PKCS#7. Tanto en codificación PEM como DER. Tanto en archivo como directamente al navegador.
- Entrega del **certificado raiz** de la autoridad. Disponible según el estándar X.509 o PKCS#7. Tanto en codificación PEM como DER. Tanto en archivo como directamente al navegador.
- Entrega de la **Lista de Certificados Revocados (CRL) actualizada** hasta el momento. Disponible según el estándar X.509 o PKCS#7. Tanto en codificación PEM como DER. Tanto en archivo como directamente al navegador.

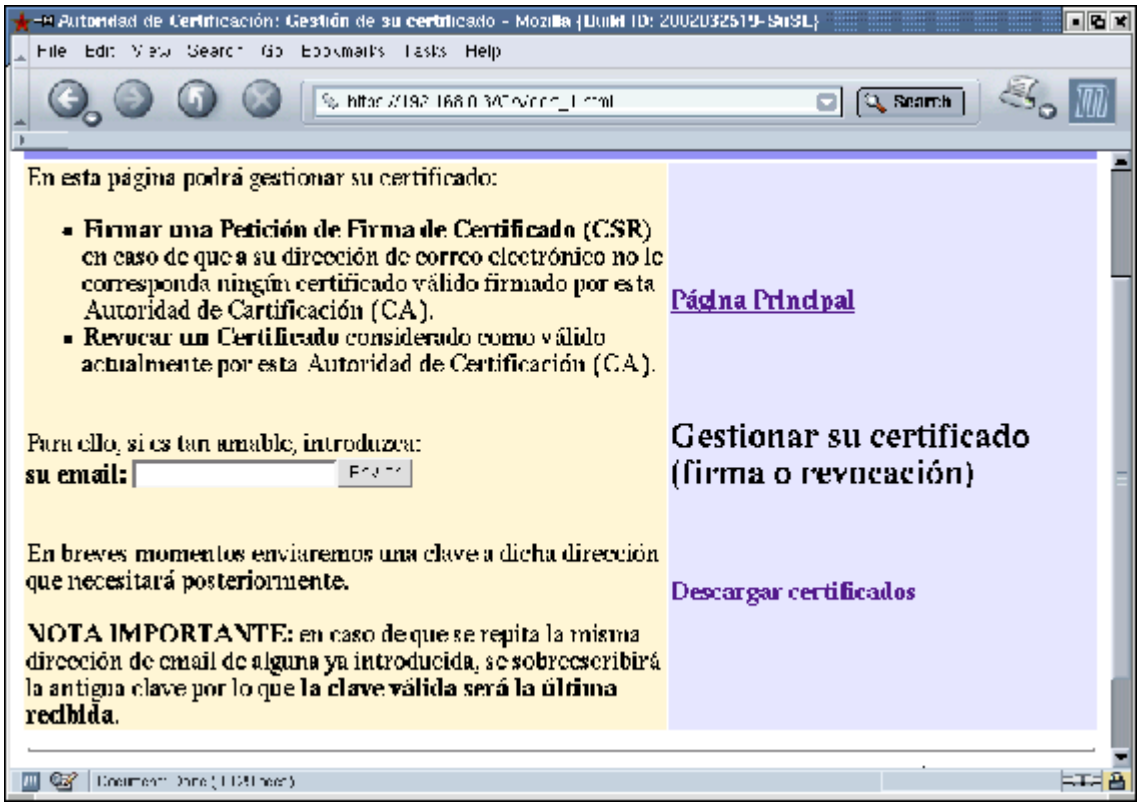


Ilustración 35: Vista de la página inicial para gestionar (firmar o revocar) un certificado

- Entrega de la **colección de certificados válidos** en esta autoridad de certificación incluido el certificado raiz de la autoridad de certificación junto con la CRL actualizada según el estándar PKCS#7. Tanto en codificación PEM como DER. Tanto en archivo como directamente al navegador.

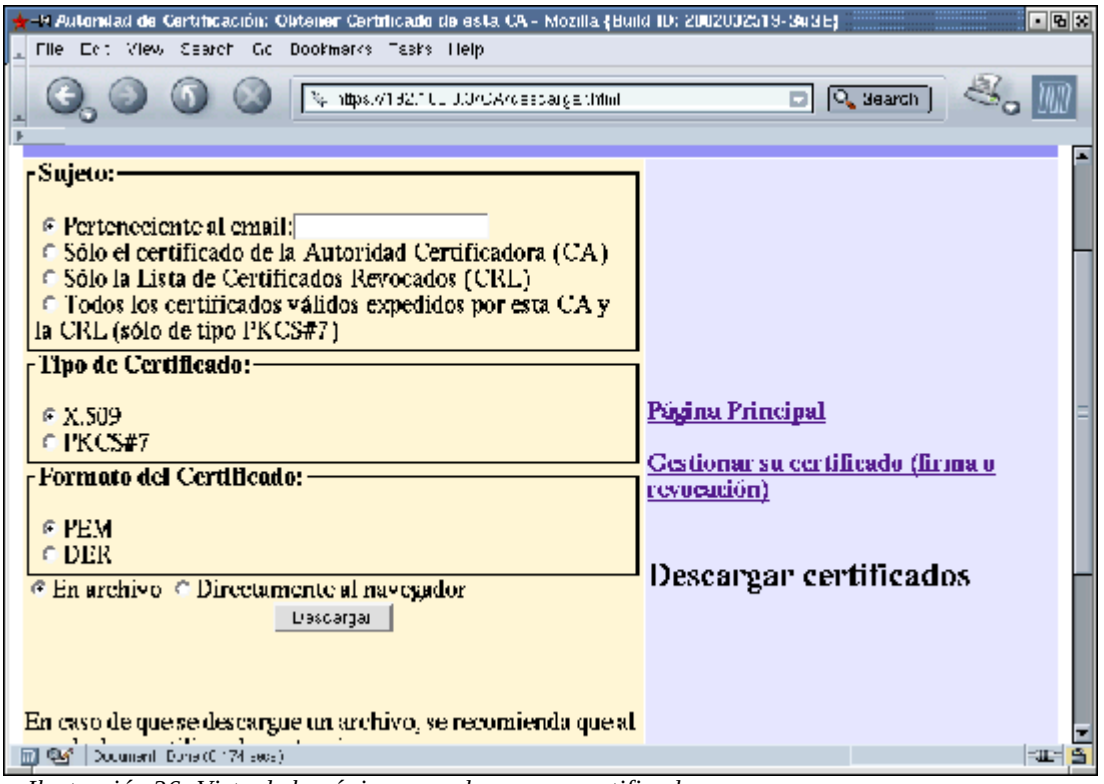


Ilustración 36: Vista de la página para descargar certificados

La certificación de la identidad del usuario está limitada a la de su dirección de correo electrónico y esta autoridad no valida los datos incluidos en el DistiguishedName (DN) del certificado (de forma similar a los antiguos certificados de clase 1 de VeriSign). Dicha certificación de la dirección de correo electrónico se lleva a cabo con la entrega de una clave vía email a la dirección de correo electrónico que pretende certificar su CSR. La política de las claves es la siguiente:

- Para asegurar que quien pretende tanto que la autoridad le firme su CSR como la revocación de un certificado válido, la autoridad genera en cada ocasión una clave de tipo numérico y la remite instantáneamente a la dirección de correo electrónico que lo solicita.
- **Por cada intento de certificación o de revocación** (se lleve a cabo o no) **se generará y enviará una clave distinta**. **La única clave válida será la última recibida en dicha dirección de correo electrónico**.

Esta aplicación ha sido testada sobre una distribución SuSE Linux 8.0 Proffesional con los directorios por defecto.

VI.1.3. Funcionamiento

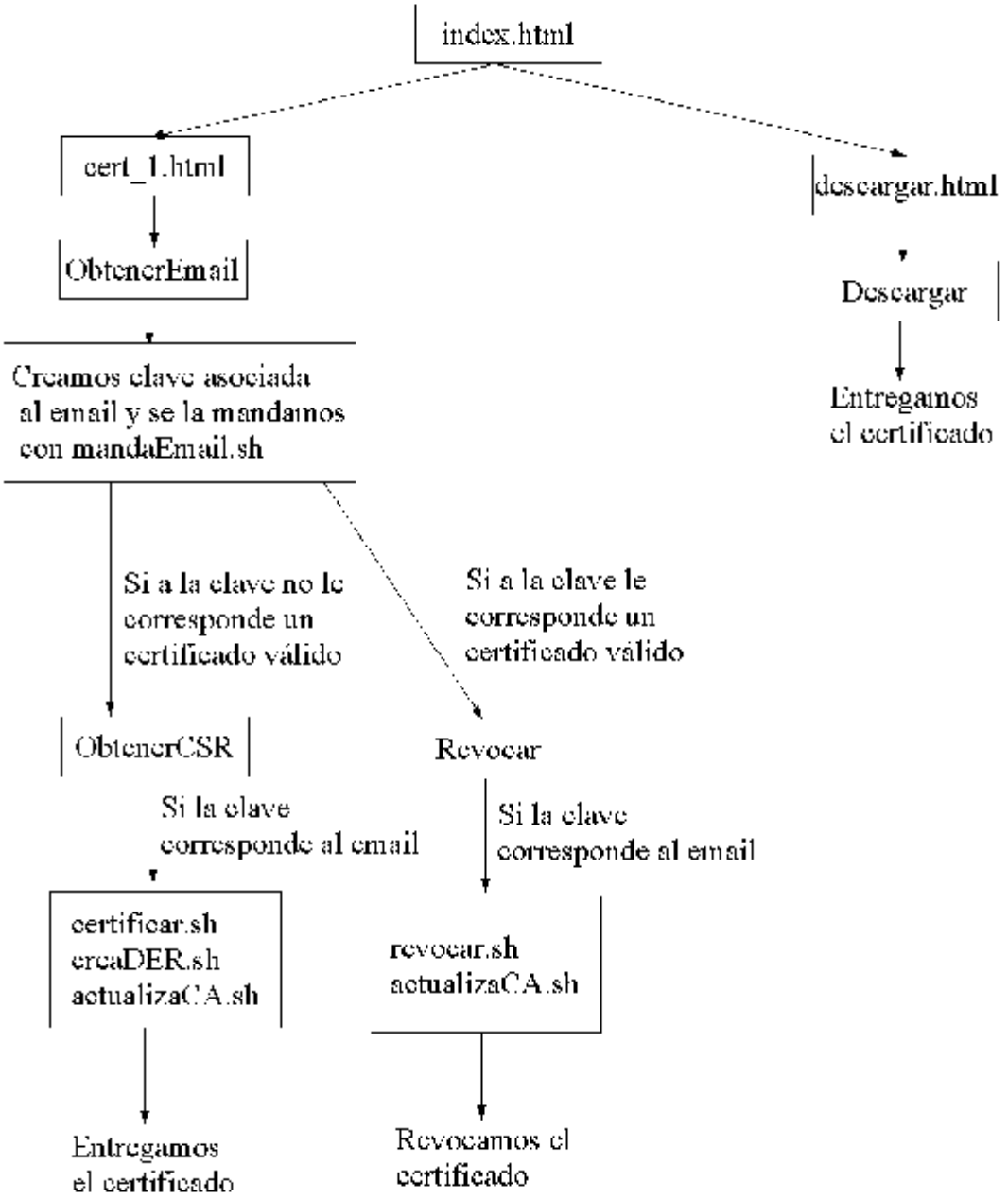


Ilustración 37: Funcionamiento de la Autoridad de Certificación (CA)

VI.1.4. Estructura interna

Como se puede ver en la ilustración, desde todas las páginas HTML se puede acceder a cualquiera de ellas. Dichas páginas contiene formularios que se sirven de HttpServlets para realizar su cometido y crear páginas de forma dinámica e incluso con otros formularios y servlets dentro de ellas.

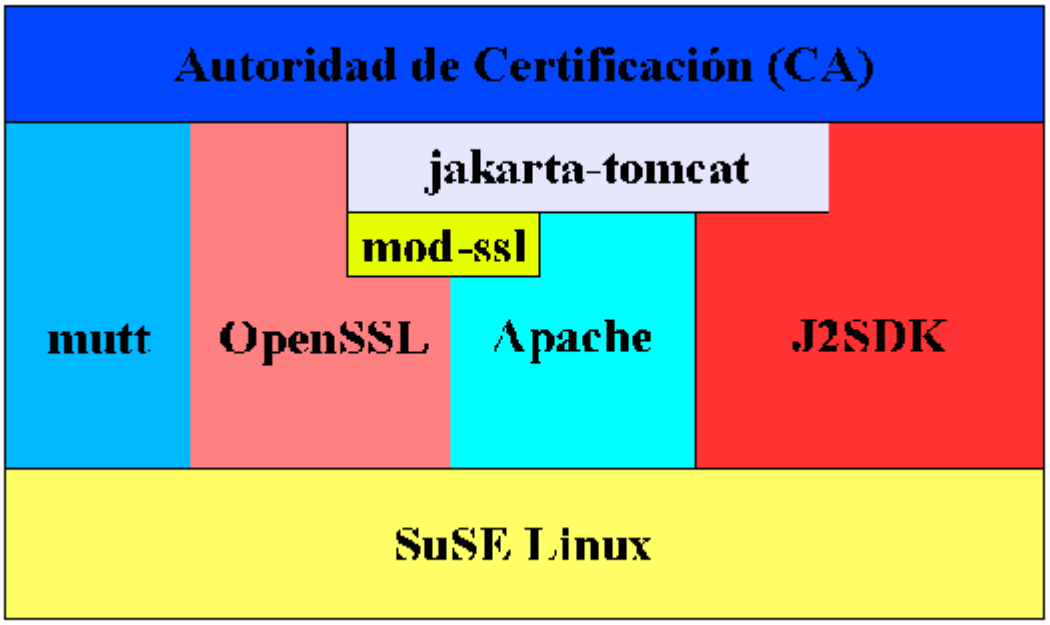


Ilustración 38: Estructura de bloques de la Autoridad de Certificación (CA)

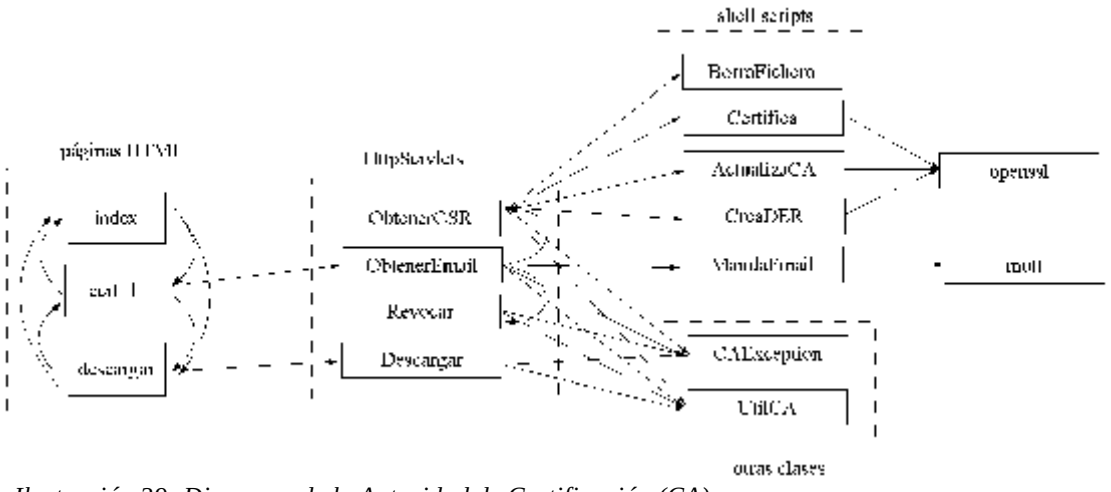


Ilustración 39: Diagrama de la Autoridad de Certificación (CA)

Los servlets se sirven de las shell scripts para ejecutar líneas de comando que ejecutan otras herramientas como `openssl` para gestionar la CA, certificados y CRL o `mutt` como cliente de correo. Los servlets también utilizan funciones de otras clases como `CAException` que

implementa la Exception que lanza la CA o `UtilCA` que contiene funciones de utilidad como son:

Función	*Descripción*
`fileToHashtable`	Función que pasa el contenido de un archivo a una Hashtable
`hashtableToFile`	Función que escribe el contenido de una Hashtable a un archivo
`stringToFile`	Función que me escribe el contenido de una cadena a un archivo
`fileToString`	Función que me pasa el contenido de un archivo a una cadena

Tabla 23: Funciones de la clase UtilCA

VI.1.5. Versión

Autoridad de Certificación (CA) v1.0

VI.1.6. Archivos

Nombre del archivo	*Descripción*
`index.html`	Página principal de la Autoridad de Certificación (CA).
`cert_1.html`	Página para gestionar (firmar o revocar) un certificado.
`descargar.html`	Página para descargar certificados y/o CRLs.
`Descargar.java`	Fuente del HttpServlet Descargar. Extrae por el método POST los datos del certificado pedido (formato, codificación, ...) del formulario de descargar.html y lo entrega segun el tipo MIME adecuado.
`Descargar.class`	Bytecode del HttpServlet Descargar.

Nombre del archivo	*Descripción*
`ObtenerEmail.java`	Fuente del HttpServlet ObtenerEmail. Extrae por el método POST la dirección de correo electrónico del solicitante, genera una clave pseudo-aleatoria segura y la envía ayudado del shell script `mandaEmail.sh`. Para poder posteriormente validar la pareja email / clave se introduce en un Hashtable que se almacena en el fichero miIndex.txt. Además envía una página HTML generada dinámicamente (estableciendo con `setMimeType("text/HTML")` el tipo MIME de la HttpResponse). Si el email no tiene un certificado asociado firmado por la CA en estado válido, la página HTML muestra un formulario donde se puede introducir, además de la clave asociada a la dirección de email, la CSR del cliente según el estándar PKCS#10 en codificación PEM. Los datos de dicho formulario se envía al HttpServlet ObtenerCSR mediante el método POST. Si por el contrario, la dirección de email tiene un certificado asociado firmado por la CA en estado válido, se enviará al cliente una página HTML en la que si así lo desea, se podrá revocar en esta CA dicho certificado asociado con sólo enviar la clave asociada a la dirección de email al HttpServlet Revocar mediante el método POST.
`ObtenerEmail.class`	Bytecode del HttpServlet ObtenerEmail.
`ObtenerCSR.java`	Fuente del HttpServlet ObtenerCSR. Obtiene la CSR del cliente según el estándar PKCS#10 en codificación PEM mediante el método POST del formulario de la página HTML generada dinámicamente. Procede a firmar dicho CSR (mediante `certifica.sh`) en caso de que la clave asociada a la dirección de email obtenida del formulario coincida con la del Hashtable almacenada en `miIndex.txt`. En caso de que se haya firmado correctamente, se crea copia el certificado en otros formatos y codificaciones mediante creaDER.sh. y posteriormente se envía según el formato y codificación pedida (ajustando el tipo MIME de HttpResponse).
`ObtenerCSR.class`	Bytecode del HttpServlet ObtenerCSR.

Nombre del archivo	*Descripción*
`Revocar.java`	Fuente del HttpServlet Revocar. Revoca un certificado en estado válido firmado por esta CA. Para cerciorarse de la autenticidad del cliente debe comparar la pareja email / clave que obtiene del formulario mediante el método POST con la almacenada en la Hashtable contenida en `miIndex.txt`. En caso de que así sea, lo revoca e informa de ello al cliente mediante una página HTML generada dinámicamente. En caso de que no sea así, no hace nada e indica del error en la clave al cliente.
`Revocar.class`	Bytecode del HttpServlet Revocar.
`UtilCA.java`	Fuente de la clase UtilCA. Contiene funciones de utilidad como las ya comentadas: fileToHashtable, hashtableToFile, stringToFile y fileToString para pasar de Hashtable o cadena a archivo y viceversa.
`UtilCA.class`	Bytecode de la clase UtilCA.
`CAException.java`	Fuente de la clase CAException. Es la Exception que lanzaremos en la CA.
`CAException.class`	Bytecode de la clase CAException.
`mandaEmail.sh`	Shell script que permite mandar emails gracias a la aplicación de línea de comandos `mutt`.
`actualizaCA.sh`	Shell script que actualiza la CA (CRL, la cadena de certificados de toda la CA junto con la CRL en formato PKCS#7) gracias a la aplicación de línea de comandos `openssl`.
`certifica.sh`	Shell script que firma una CSR pasada como parámetro en un certificado gracias a la aplicación de línea de comandos `openssl`.
`creaDER.sh`	Shell script que convierte los certificados PEM en DER gracias a la aplicación de línea de comandos `openssl`.
`desinstalarCA.sh`	Shell script que desinstala la CA.
`iniciaCA.sh`	Shell script que inicia la CA. Borra los archivos antiguos y crea los nuevos necesarios para el correcto funcionamiento de la CA.
`makeCA.sh`	Shell script que compila las fuentes de las clases de la CA mediante `javac`.
`revoca.sh`	Shell script que revoca un certificado pasado como parámetro gracias a la aplicación de línea de comandos `openssl`.
`borraFichero.sh`	Shell script que borra un fichero pasado como parámetro.

Tabla 24: Descripción de los principales archivos de la Autoridad de Certificación (CA). Dentro de los propios archivos se incluye una descripción más detallada e informa de los parámetros para cada script.

VI.1.7. Instalación

Consultar el Anexo C: Manual de administrador de la Autoridad de Certificación (CA)

VI.1.8. Configuración y puesta en marcha

Consultar el Anexo C: Manual de administrador de la Autoridad de Certificación (CA)

VI.1.9. Utilización

Consultar el Anexo C: Manual de administrador de la Autoridad de Certificación (CA)

VII. Presupuesto

VII.1. Personal dedicado

En el desarrollo de este proyecto ha estado involucrado un **Ingeniero de Telecomunicación** a tiempo completo durante **ciento noventa días**.

VII.2. Planificación presupuestaria.

Sueldo base mensual (31 días / mes)	€ 1.302,00
TOTAL DÍA	**€ 42,00**

Tabla 25: Sueldo de un Ingeniero de Telecomunicación

Vacaciones retribuidas	7,50%
Indemnización de despido	1,75%
Seguro accidentes	2,50%
Subsidio familiar	2,00%
Subsidio vejez	11,50%
Días enfermo	1,50%
Plus cargas familiares	4,50%
Gratificaciones extraordinarias	20,00%
Otros conceptos	20,00%
TOTAL OBLIGACIONES SOCIALES	**73.7%**

Tabla 26: Relación obligaciones sociales

VII.2.1. Importe salarial efectivo total

Salario diario	€ 42,00
Obligaciones sociales	€ 30,95
TOTAL DÍA	**€ 72,95**

Tabla 27: Importe salarial efectivo total

VII.3. Importe total salarial

Sueldo diario	€ 72,95
Días empleados	190
SUELDO GLOBAL	**€ 13.860,50**

Tabla 28: Importe total salarial

VII.4. Costes materiales

Equipo informático	€ 900,00
Consumibles informáticos	€ 100,00
Material edición	€ 60,00
Acceso telefónico a internet	€ 110,00
TOTAL MATERIAL	**€ 1.170,00**

Tabla 29: Costes materiales

VII.5. Presupuesto total

Total salarios	€ 13.860,50
Total material	€ 1.170,00

SUBTOTAL	€ 15.030,50
IVA (16%)	€ 2.404,88
TOTAL PROYECTO	**€ 17.435,38**

Tabla 30: Presupuesto total

Presupuesto total: € 17.435,38

VIII. Líneas de continuación

- Estudiar e implementar la posible extensión de la aplicación XMastin al correo seguro S/MIME (con uso o no de criptografía simétrica).
- Estudiar e implementar la posible extensión de la aplicación XMastin al uso de dispositivos bajo el estándar PKCS#11.
- Estudiar e implementar la posible extensión de la Autoridad de Certificación (CA) en el uso de directorios LDAP.
- Estudiar e implementar el uso de criptografía simétrica o asimétrica en la encriptación de flujos de datos (voz, video) en tiempo real mediante el uso de SSL.

IX. Conclusiones

Tras la realización del Proyecto se puede hacer balance de los **logros conseguidos**:

- Amplio y profundo **estudio de la criptología**: criptografía, criptoanálisis, criptografía simétrica y asimétrica, sus algoritmos, ventajas, inconvenientes, y cómo se complementan; funciones digestoras, algoritmos de distribución de claves, la PKI, las Autoridades de Certificación, los certificados, sus estándares y tipos de codificación, los estándares PKCS, etc.
- Estudio de la Arquitectura Criptográfica de Java (**JCA**) y de la Extensión Criptográfica de Java (**JCE**).
- Estudio de los distintos Proveedores de Servicios Criptográficos (**CSP**) disponibles en la actualidad (noviembre de 2002) tanto comerciales como no comerciales. Dicho estudio será de gran utilidad para líneas futuras de continuación.
- Desarrollo en Java de una aplicación en línea de comandos junto con su API (**Mastin v1.0**) que utiliza las claves almacenadas en un KeyStore para encriptar con el algoritmo RSA según el estándar PKCS#1 en modo de funcionamiento Electronic Code Book (ECB) con relleno Optimal Asymmetric Encryption Padding (OAEP) definido en el PKCS#1 v2.
- Desarrollo en Java de una apliación (**XMastin v1.0**) que hace uso del API Mastin v1.0 con el fin de simplificar su uso y hacerlo más amigable al usuario.

  La implementación de ambas aplicaciones se realizó en Java para lograr la máxima portabilidad entre plataformas.

  La tarea de generación y almacenamiento de claves recae en la herramienta de gestión de claves keytool de Java con el objetivo siempre presente de la máxima portabilidad entre plataformas.
- Desarrollo del **Manual de usuario de Mastin v1.0 y XMastin v1.0.**
- Configuración y puesta en marcha de un servidor HTTP Seguro (**HTTPS**) a partir de apache y jakarta-tomcat para dar servicio a la Autoridad de Certificación (CA) .
- Implementación de una **Autoridad de Certificación (CA)** basada en OpenSSL y a través del protocolo HTTPS, con el principal objetivo de ser independiente del navegador.
- Desarrollo del **Manual de administrador de la Autoridad de Certificación (CA) v1.0**.

En la realización de este Proyecto **se han cumplido los objetivos marcados** de forma más que satisfactoria. Aunque existen diversas maneras de realizar este Proyecto, se eligió la forma probablemente más efectiva y se dejan las puertas abiertas a futuras mejoras o reimplementaciones que pudieran realizarse.

Debido a que la realización ha durado **ciento noventa días**, la información vertida en su memoria puede que se encuentre anticuada en un futuro cercano. Esto es así porque el mundo de las Tecnologías de la Información cambia muy rápidamente, y lo que hoy se considera actual puede que no sea así dentro de pocos meses.

Las **fuentes de información** en las que se ha basado han sido muy variadas desde la fiable (?) aunque estática bibliografía hasta las dinámicas (?) pero poco fiables páginas web.

Por la realización del Proyecto, su autor ha obtenido numerosos **beneficios** en su trabajo: desde una introducción al mundo de la criptografía actual hasta la creación de páginas HTML pasando por la introducción a los paquetes AWT y Swing de Java. Su enumeración sería larga y seguramente no completa.

A quien pudiera servir de ayuda en futuros proyectos, aclaraciones sugerencias o diversas cuestiones dejo mi email a su disposición gabrielbabiano@yahoo.es.

Anexo A: PKCS

Los Public-Key Cryptography Standards (PKCS) son un conjunto de estándares para la criptografía de clave pública desarrollados por *RSA Laboratories* en cooperación con un consorcio informal, que originalmente incluía a Apple, Microsoft, DEC, Lotus, Sun y el MIT.

Las PKCS están diseñados para datos tanto binarios como ASCII.

Los estándares publicados son PKCS #1, #3, #5, #7, #8, #9, #10 #11, #12, y #15. Los PKCS #13 y #14 están siendo desarrollados en la actualidad.

Los PKCS incluyen estándares de implementación tanto independientes como específicos de un algoritmo. Se soportan algunos algoritmos, incluyendo RSA y el intercambio de claves de Diffie-Hellman y además estos dos están detallados específicamente.

Los PKCS también define una sintaxis independiente del algoritmo para firmas digitales y certificados etendidos. Esto permite implementar cualquier algoritmo criptográfico conforme a una sintaxis estandarizada y así alcanzar la interoperatibilidad.

La enumeración de los estándares PKCS es la siguiente:

PKCS	***Descripción***
PKCS #1	Define mecanismos para encriptar y firmar datos usando el criptosistema de clave pública RSA.
PKCS #3	Define el protocolo de intercambio de claves Diffie-Hellman (DH).
PKCS #5	Describe un método para encriptar una cadena con una clave secreta derivada de una contraseña (PBE).
PKCS #6	Está siendo retirada a favor de la versión 3 de X.509.
PKCS #7	Define una sintaxis general para mensajes que incluyen mejoras critptográficas tales como firmas digitales y encriptación.
PKCS #8	Describe un formato para la información de la clave privada. Dicha información incluye la clave privada para un algoritmo de clave pública y opcionalmente un conjunto de atributos.
PKCS #9	Define un conjunto de tipos de atributos para usar en otros estándares PKCS.
PKCS #10	Describe una sintaxis para las Peticiones de Firma de Certificados (CSR).

PKCS	*Descripción*
PKCS #11	Define una interfaz de programación independiente de la tecnología llamado Cryptoki, para dispositivos criptográficos tales como tarjetas inteligentes y tarjetas PCMCIA.
PKCS #12	Especifica un formato portátil para guardar o transportar claves privadas de usuario, certificados, etc.
PKCS #13	Pretende definir mecanismos para encriptar y firmar datos usando Criptografía de Curva Elíptica (CCE).
PKCS #14	Está actualmente en desarrollo y cubre la generación de números pseudo-aleatorios.
PKCS #15	Es un complemento al PKCS #11 ofreciendo un estándar al formato de credenciales criptográficos almacenados en cryptographic tokens.

Los documentos que detallan los estándares PKCS se pueden obtener del servidor de RSA Security que es accesible desde http://www.rsasecurity.com/rsalabs/pkcs/ o vía ftp anónimo por ftp://ftp.rsasecurity.com/pub/pkcs/doc/.

Anexo B: Manual de usuario de Mastin v1.0 y XMastin v1.0

Anexo C: Manual de administrador de la Autoridad de Certificación (CA)

Bibliografía

Título	**Seguridad y comercio en el Web**
Autores	***Simon Garfinkel y Gene Spafford***
Editorial	McGRAW-HILL INTERAMERICANA EDITORES, S.A. de C.V.
Año	1999
ISBN	970-10-2142-8
Título original	Web security and commerce
Editorial	O'Reilly &Associates
Año	1998
ISBN	1-56592-269-7
Comentarios	Libro interesante aunque al tratar de una temática muy técnica, se está quedando anticuado.

Título	**Edición Especial Seguridad en Java**
Autores	***Jamie Jaworski y Paul J. Perrone***
Editorial	PEARSON EDUCACIÓN, S.A.
Año	2001
ISBN	84-205-3134-0
Título original	Java Security Handbook
Editorial	Sams Publishing
Año	2000
ISBN	0-672-31602-1
Comentarios	Libro que puede servir como manual de referencia para JCA

Título	JAVA 2: Manual de usuario y tutorial, 2ª edición
Autor	*Agustín Froufe Quintas*
Editorial	RA-MA Editorial
Año	2000
ISBN	84-7897-429-6
Comentarios	Libro de Java que trata varios paquetes básicos (servlets, ficheros, swing, AWT, ...)y que contiene ejemplos que pueden servir como esqueleto.

Título	Superutilidades para Webmasters
Autor	*Jesús Bobadilla Sancho*
Editorial	McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S.A.U.
Año	1999
ISBN	84-4812423-5
Comentarios	Libro de iniciación a determinados temas como pueden ser HTML o de seguridad en el web

Recomendación X.509

ITU-T

Java Cryptography Extension (JCE) Reference Guide

Sun Microsystems

Java Cryptography Architecture (JCA) API Specification & Reference

Sun Microsystems

Documentación de OpenSSL: ca, crl, crl2pkcs7, pkcs7, x509

OpenSSL

Acrónimos

	A
AEA	Advanced Encryption Algorithm
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
ASN.1	Abstract Sintax Notation 1
AWT	Abstract Window Toolkit
	B
	C
C	CountryCode
CA	Certification Authority
CBC	Cipher Block Chaining
CCE	Criptografía de Curva Elíptica
CD	Compact Disc
CDROM	Compact Disc Read Only Memory
CFB	Cipher FeedBack
CGI	Common Gateway Interface
CN	CommonName
CPS	Certificate Practice Statement
CPU	Central Proccesing Unit
CRL	Certificates Revokated List
CSP	Cryptographic Service Provider
CSR	Certificate Signing Request
	D
DEA	Data Encryption Algorithm

DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DH	Diffie-Hellman
DHE	Diffie-Hellman Elíptico
DN	DistinguishedName
DPA	Differential Power Analysis
DSS	Digital Signature Standard
	E
ECB	Electronic Code Book
	F
FEAL	Fast data Enciphment ALgorithm
FNMT	Fábrica Nacional de Moneda y Timbre
	G
GNU	GNU's Not UNIX
GPL	General Public Licence
GUI	Graphic User Interface
	H
HMAC	Hash Message Authenticaction Codes
HTML	Hyper Text Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
	I
IAIK	Applied Information Procesing and Communications (en alemán)
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ITU	International Telecommunication Union
	J
J2SDK	Java 2 Standard Development Kit
JCA	Java Cryptographic Architecture
JCE	Java Cryptographic Extension
JCSI	Java Crypto and Security Implementation
JFC	Java Foundation Classes