Aplicaciones seguras

Xavier Perramon

Índice

Introducción

En este módulo se describen aplicaciones que utilizan técnicas de seguridad en las comunicaciones para proteger los datos intercambiados.

Un ejemplo de aplicaciones que hay que proteger son las que permiten establecer sesiones de trabajo interactivas con un servidor remoto. En la primera parte del módulo veremos una aplicación de este tipo, llamada **SSH**, que define su propio protocolo para que los datos de la aplicación se transmitan cifrados. El mismo protocolo proporciona también mecanismos de autenticación del servidor frente al usuario, y del usuario frente al servidor. Como veremos, el protocolo SSH se puede utilizar para otras aplicaciones aparte de las sesiones interactivas, ya que permite el encapsulamiento de conexiones TCP a cualquier puerto dentro de una conexión SSH.

La segunda parte de este módulo la dedicaremos a otra de las principales aplicaciones que suele ser necesario proteger: el **correo electrónico**. Con los mecanismos de protección adecuados se puede garantizar la **confidencialidad**, es decir, que nadie más que el destinatario o destinatarios legítimos puedan ver el contenido de los mensajes, y la **autenticidad**, es decir que los destinatarios puedan comprobar que nadie ha falsificado un mensaje. Los sistemas actuales de correo electrónico normalmente utilizan las **firmas digitales** para proporcionar el servicio de autenticación de mensaje.

Una particularidad del correo electrónico seguro respecto a otras aplicaciones como SSH es que la protección se realiza preferentemente sobre los mensajes enviados, más que sobre los protocolos de comunicación utilizados. Esto es así porque la confidencialidad y la autenticidad se tiene que garantizar no sólo durante la transmisión de los mensajes, sino también en cualquier otro momento posterior, ya que el destinatario puede guardar los mensajes que recibe y volverlos a leer cuando le convenga.

En este módulo veremos dos de los principales sistemas utilizados actualmente para proteger los mensajes de correo electrónico, **S/MIME** y **PGP**.

Objetivos

Con los materiales asociados a este módulo didáctico alcanzareis los siguientes objetivos:

1. Conocer el mecanismo general de funcionamiento del protocolo SSH y las principales aplicaciones que lo utilizan.

2. Comprender las funciones de seguridad que pueden proporcionar los sistemas de correo electrónico seguro y los mecanismos para alcanzarlas.

3. Identificar el estándar S/MIME como aplicación de la especificación MIME al correo seguro, y conocer el uso que se hace del estándar PKCS #7 y de los certificados digitales X.509.

4. Conocer el método de representación de información segura en PGP y el modelo de confianza mutua del sistema PGP.

1. El protocolo SSH

SSH es una aplicación diseñada para substituir determinadas herramientas de acceso remoto usadas tradicionalmente en los sistemas Unix, como `rsh` (*Remote Shell*), `rlogin` (*Remote Login*) o `rcp` (*Remote Copy*), por nuevas versiones con servicios de seguridad.

El nombre de esta aplicación, SSH, es la abreviatura de *Secure Shell*, que viene a significar "versión segura del programa *Remote Shell*".

La aplicación define un protocolo propio para la transmisión segura de los datos, el **protocolo SSH**. Este protocolo se sitúa directamente por debajo de la capa de transporte, (concretamente del transporte TCP) y, como veremos en este apartado, proporciona servicios análogos a los del protocolo SSL/TLS. Aparte de establecer conexiones seguras, el protocolo SSH también ofrece otras funcionalidades como, por ejemplo, la redirección de puertos TCP o la comunicación entre clientes y servidores de ventanas X, a través de una conexión SSH.

El autor de la primera implementación del SSH, Tatu Ylönen, de la Universidad Tecnológica de Helsinki, publicó el año 1995 la especificación de la versión 1 del protocolo. Desde entonces se ha trabajado en la especificación de una nueva versión del protocolo, la 2.0, actualmente en fase de borrador a la espera de su publicación oficial como RFC. Aunque la funcionalidad que proporciona es básicamente la misma, la nueva versión incorpora muchas mejoras y es sustancialmente distinta de la anterior. La versión antigua y la nueva del protocolo se referencian habitualmente referenciadas como SSH1 y SSH2, respectivamente. En este apartado nos centraremos sobre todo en el protocolo SSH2.

1.1. Características del protocolo SSH

SSH proporciona servicios de seguridad equivalentes a los del protocolo SSL/TLS.

Confidencialidad. SSH sirve para comunicar datos, que habitualemente son la entrada de una aplicación remota y la salida que genera, o bien la información que se transmite por un puerto redirigido, y la confidencialidad de estos

datos se garantiza mediante el cifrado.

Como en el caso del protocolo SSL/TLS, en SSH se aplica un cifrado simétrico a los datos y, por lo tanto, será necesario realizar previamente un intercambio seguro de claves entre cliente y servidor. Una diferencia respecto a SSL/TLS es que en SSH2 se pueden utilizar algoritmos de cifrado distintos en los dos sentidos de la comunicación.

Un servicio adicional que proporciona SSH es la confidencialidad de la identidad del usuario. Mientras que en SSL 3.0 y TLS 1.0, si se opta por autenticar al cliente, éste tiene que enviar su certificado en claro, en SSH (y también en SSL 2.0) la autenticación del usuario se realiza cuando los paquetes ya se mandan cifrados.

Por otro lado, SSH2 también permite ocultar ciertas características del tráfico como, por ejemplo, la longitud real de los paquetes.

Autenticación de entidad. El protocolo SSH proporciona mecanismos para autenticar tanto el ordenador servidor como el usuario que se quiere conectar. La autenticación del servidor suele realizarse conjuntamente con el intercambio de claves. En SSH2 el método de intercambio de claves se negocia entre el cliente y el servidor, aunque actualmente sólo hay uno definido, basado en el algoritmo de Diffie-Hellman.

Para autenticar al usuario existen distintos métodos; dependiendo de cuál se utilice, puede ser necesaria también la autenticación del ordenador cliente, mientras que otros métodos permiten que el usuario debidamente autenticado acceda al servidor desde cualquier ordenador cliente.

Autenticación de mensaje. Igual que en SSL/TLS, en SSH2 la autenticidad de los datos se garantiza añadiendo a cada paquete un código MAC calculado con una clave secreta. También existe la posibilidad de utilizar algoritmos MAC distintos en cada sentido de la comunicación.

Igual que SSL/TLS, SSH también está diseñado con los siguientes criterios adicionales:

Eficiencia. SSH contempla la compresión de los datos intercambiados para reducir la longitud de los paquetes. SSH2 permite negociar el algoritmo que se utilizará en cada sentido de la comunicación, aunque solamente existe uno definido en la especificación del protocolo. Este algoritmo es compatible con el que utilizan programas como `gzip` (RFC 1950–1952).

A diferencia de SSL/TLS, en SSH no está prevista la reutilización de claves de sesiones anteriores: en cada nueva conexión se vuelven a calcular las claves. Esto es así porque SSH está pensado para conexiones que tienen una duración más o menos larga, como suelen ser las sesiones de trabajo interactivas con un ordenador remoto, y no para las conexiones cortas pero consecutivas, que son más típicas del protocolo de aplicación HTTP (que es el que inicialmente se quería proteger con SSL). De todas formas, SSH2 define mecanismos para

intentar acortar el proceso de negociación.

Extensibilidad. En SSH2 también se negocian los algoritmos de cifrado, de autenticación de usuario, de MAC, de compresión y de intercambio de claves. Cada algoritmo se identifica con una cadena de caracteres que representa su nombre. Los nombres pueden corresponder a algoritmos oficialmente registrados, o bien a algoritmos propuestos experimentalmente o definidos localmente.

Algoritmos no oficiales

Los nombres de los algoritmos no oficiales deben de ser de la forma "*`nombre@dominio`*", donde *`dominio`* es un dominio DNS controlado por la organitzación que define el algoritmo (por ejemplo "`cifrado-fuerte@uoc.edu`").

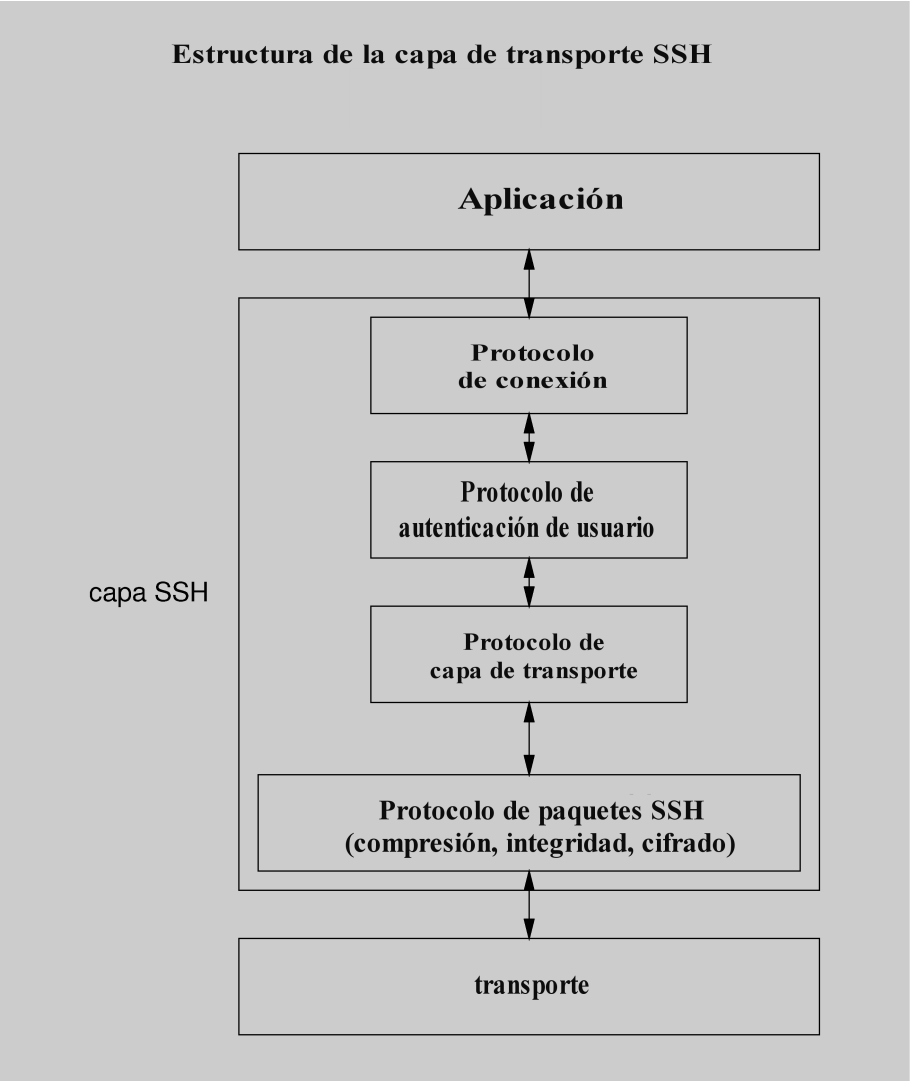
Adaptación de SSH a idiomas locales

Por otro lado, SSH2 facilita la adaptación de las implementaciones a los idiomas locales. Donde el protocolo prevé la transmisión de un mensaje de error o informativo que pueda ser mostrado al usuario humano, se incluye una etiqueta que identifica el idioma del mensaje, de acuerdo con el RFC 1766.

Tanto estos mensajes como los nombres de usuario se representan con el juego de caracteres universal ISO/IEC 10646 mediante la codificación UTF-8, de acuerdo con el RFC 2279 (el código ASCII es un subconjunto porque en UTF-8 los caracteres con código menor a 128 se representan con un solo byte, de valor igual al código).

1.2. La capa de transporte SSH

De la misma forma que en SSL/TLS se distinguen dos subcapas en el nivel de transporte seguro, en SSH también se puede considerar una división en dos subniveles. Además, en SSH2 el nivel superior está estructurado en tres protocolos, uno por encima del otro, como muestra la siguiente figura:



En el nivel inferior de la capa SSH se situa el **protocolo de paquetes SSH**. Los tres protocolos existentes por encima de éste son:

- El **protocolo de capa de transporte**, que se encarga del intercambio de claves.
- El **protocolo de autentificación de usuario**.
- El **protocolo de gestión de las conexiones**.

1.2.1. El protocolo de paquetes SSH

El protocolo de paquetes SSH se encarga de construir e intercambiar las unidades del protocolo, que son los **paquetes SSH**.

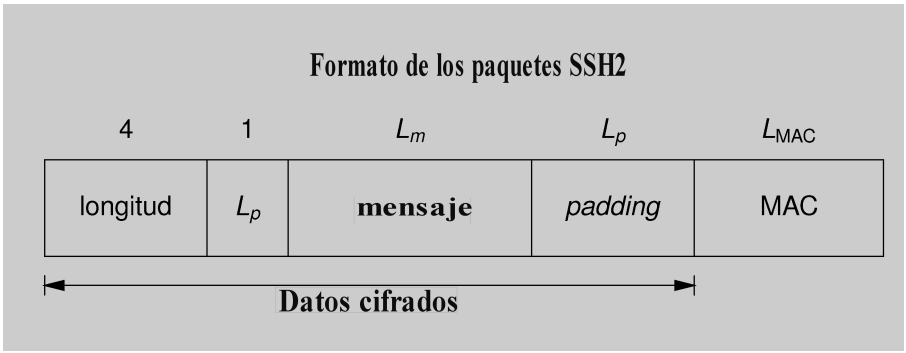
En el momento de enviar datos, a los mensajes de los niveles superiores se las

aplica (por este orden):

- La compresión.
- El código de autenticación MAC.
- El cifrado.

En la recepción, a cada paquete se le aplica el procesamiento inverso (descifrado, verificación de autenticidad y descompresión).

El formato de los paquetes SSH2 es el siguiente:



Los campos existentes en un paquete SSH2 son los siguientes:

- El primero es la longitud del resto del paquete, excluido el MAC (por lo tanto, es igual a $1+L_m+L_p$).
- El segundo campo indica cuántos bytes de *padding* existen. Este número de bytes debe ser tal que la longitud total del paquete, excluido el MAC, sea múltiple de 8 (o de la longitud de bloque en los cifrados de bloque, si es más grande que 8).
- El tercer campo es el contenido del mensaje, comprimido si se da el caso. El primer byte del contenido siempre indica de qué tipo de mensaje se trata, y la estructura del resto de bytes depende del tipo.
- El cuarto campo son los bytes aleatorios de *padding*. Siempre están presentes, incluso cuando el cifrado utilizado sea en flujo, y su longitud tiene que ser como mínimo igual a 4. Por lo tanto, la longitud mínima de un paquete, sin contar el MAC, es de 16 bytes.
- El quinto campo es el código de autentificación MAC, obtenido mediante la técnica HMAC a partir de una clave secreta, un número de secuencia implícito de 32 bits y el valor de los otros cuatro campos del paquete. La longitud del MAC depende del algoritmo acordado, y puede ser 0 si se utiliza el algoritmo nulo.

Bytes de *padding*

Los bytes de *padding* aseguran que la longitud de los datos que hay que cifrar sea la adecuada para los cifrados de bloque.

Cuando se cifran los paquetes, se aplica el cifrado a todos los campos excepto el del MAC, pero incluyendo la longitud. Eso significa que el receptor tiene

que descifrar los 8 primeros bytes de cada paquete para conocer la longitud total de la parte cifrada.

1.2.2. El protocolo de capa de transporte SSH

El protocolo de capa de transporte se encarga del establecimiento de la conexión de transporte, de la autentificación del servidor y intercambio de claves, y de las peticiones de servicio de los demás protocolos.

El cliente se conecta al servidor mediante el protocolo TCP. El servidor debe estar escuchando peticiones de conexión en el puerto asignado al servicio SSH, que es el 22.

El primer paso, una vez establecida la conexión, es negociar la versión del protocolo SSH que se utilizará. Tanto el cliente como el servidor envían una línea que contiene el texto "`SSH-`*`x.y-implementación`*", donde *`x.y`* es el número de versión del protocolo (por ejemplo, `2.0`) e *`implementación`* es una cadena identificativa del software del cliente o servidor. Si los números de versión no concuerdan, el servidor decide si puede continuar o no: si no puede, simplemente cierra la conexión. Antes de esta línea de texto, el servidor también puede enviar otras con mensajes informativos, mientras no empiecen con "`SSH-`".

Compatibilidad con la versión 1

En SSH2 se define un modo de compatibilidad con SSH1, en el cual el servidor identifica su versión con el número 1.99: los clientes SSH2 deben considerar este número equivalente a 2.0, mientras que los clientes SSH1 responderán con su número de versión real.

Cuando se han puesto de acuerdo en la versión, cliente y servidor pasan a intercambiar mensajes con el protocolo de paquetes SSH visto anteriormente, inicialmente sin cifrar y sin MAC. Para ahorrar tiempo, el primer paquete SSH se puede enviar juntamente con la línea que indica la versión, sin esperar a recibir la línea de la otra parte. Si las versiones coinciden, el protocolo continúa normalmente; si no, puede ser necesario reiniciarlo.

En primer lugar, se procede al intercambio de claves. En SSH2 cada parte envía un mensaje `KEXINIT` que contiene una cadena de 16 bytes aleatorios llamada *cookie*, y las listas de algoritmos soportados por orden de preferencia: algoritmos de intercambio de claves y, para cada sentido de la comunicación, algoritmos de cifrado simétrico, de MAC y de compresión. También se incluye una lista de idiomas soportados por los mensajes informativos. Para cada tipo de algoritmo, se escoge el primero de la lista del cliente que esté también en la lista del servidor.

Algoritmos previstos en SSH2

Los algoritmos criptográficos que contempla SSH2 son los siguientes:

- Para el intercambio de claves: Diffie-Hellman.
- Para el cifrado: RC4, Triple DES, Blowfish, Twofish, IDEA y CAST-128.

- Para el MAC: HMAC-SHA1 y HMAC-MD5 (con todos los bytes o sólo con los 12 primeros).

Los paquetes que vienen a continuación son los de intercambio de claves, y dependen del algoritmo escogido (aunque SSH2 sólo prevé el algoritmo de Diffie-Hellman).

Se puede suponer que la mayoría de implementaciones tendrán un mismo algoritmo preferido de cada tipo. De este modo, para reducir el tiempo de respuesta se puede enviar el primer mensaje de intercambio de claves después del `KEXINIT` sin esperar el de la otra parte, utilizando estos algoritmos preferidos. Si la suposición resulta acertada, el intercambio de claves continúa normalmente, y si no, los paquetes enviados anticipadamente se ignoran y se vuelven a enviar con los algoritmos correctos.

Sea cual sea el algoritmo, como resultado del intercambio de claves se obtiene un secreto compartido y un identificador de sesión. Con el algoritmo Diffie-Hellman, este identificador es el *hash* de una cadena formada, entre otras, por las *cookies* del cliente y el servidor. Las claves de cifrado y de MAC y los vectores de inicialización se calculan aplicando funciones *hash* de varias formas a distintas combinaciones del secreto compartido y del identificador de sesión.

Para finalizar el intercambio de claves cada parte envía un mensaje `NEWKEYS`, que indica que el siguiente paquete será el primero que utilizará los nuevos algoritmos y claves.

Todo este proceso se puede repetir cuando sea necesario para regenerar las claves. La especificación SSH2 recomienda hacerlo después de cada gigabyte transferido o de cada hora de tiempo de conexión.

Si se produce algún error en el intercambio de claves, o en cualquier otro punto del protocolo, se genera un mensaje `DISCONNECT`, que puede contener un texto explicativo del error, y se cierra la conexión.

Otros mensajes que se pueden intercambiar en cualquier momento son:

- `IGNORE`: su contenido debe ser ignorado, pero se puede usar para contrarrestar el análisis de flujo de tráfico.
- `DEBUG`: sirven para enviar mensajes informativos.
- `UNIMPLEMENTED`: se envían en respuesta a mensajes de tipo desconocido.

En SSH2, después de finalizado el intercambio de claves el cliente envía un mensaje `SERVICE_REQUEST` para solicitar un servicio, que puede ser au-

tenticación de usuario, o bien acceso directo al protocolo de conexión si no es necesaria autenticación. El servidor responde con `SERVICE_ACCEPT` si permite el acceso al servicio solicitado, o con `DISCONNECT` en caso contrario.

1.2.3. El protocolo de autenticación de usuario

En SSH se contemplan distintos métodos de autenticación de usuario:

1) **Autenticación nula.** El servidor permite que el usuario acceda directamente, sin ninguna comprobación, al servicio solicitado. Un ejemplo sería el acceso a un servicio anónimo.
2) **Autenticación basada en listas de acceso.** A partir de la dirección del sistema cliente y el nombre del usuario de este sistema que solicita el acceso, el servidor consulta una lista para determinar si el usuario está autorizado a acceder al servicio. Ésta es la misma autenticación que utiliza el programa `rsh` de Unix, en el cual el servidor consulta los ficheros `.rhosts` y `/etc/hosts.equiv`. Dada su vulnerabilidad a ataques de falsificación de dirección IP, este método sólo se puede utilizar en SSH1: SSH2 no lo soporta.
3) **Autenticación basada en listas de acceso con autenticación de cliente.** Es igual que el anterior, pero el servidor verifica que el sistema cliente sea efectivamente quien dice ser, para evitar los ataques de falsificación de dirección.
4) **Autenticación basada en contraseña.** El servidor permite el acceso si el usuario da una contraseña correcta. Éste es el método que sigue normalmente el proceso `login` en los sistemas Unix.
5) **Autenticación basada en clave pública.** En lugar de dar una contraseña, el usuario se autentica demostrando que posee la clave privada correspondiente a una clave pública reconocida por el servidor.

En SSH2 el cliente va mandando mensajes `USERAUTH_REQUEST`, que incluyen el nombre de usuario (se puede ir cambiando de un mensaje a otro), el método de autenticación solicitado, y el servicio al que se quiere acceder. Si el servidor permite el acceso responderá con un mensaje `USERAUTH_SUCCESS`; si no, enviará un mensaje `USERAUTH_FAILURE`, que contiene la lista de métodos de autenticación que se pueden continuar intentando, o bien cerrará la conexión si ya se han producido demasiados intentos o ha pasado demasiado tiempo. El servidor puede enviar opcionalmente un mensaje informativo `USERAUTH_BANNER` antes de la autenticación.

Mensaje `USERAUTH_BANNER`

El mensaje `USERAUTH_BANNER` puede incluir, por ejemplo, texto identificativo del sistema servidor, avisos sobre restricciones de uso, etc. Éste es el tipo de información que normalmente muestran los sistemas Unix antes del *prompt* para introducir el nombre de usuario, y suele estar en el fichero `/etc/issue`.

Los mensajes de solicitud de autenticación contienen la siguiente información según el método:

1) Para la autenticación nula no se precisa ninguna información adicional.

2) En la autenticación basada en listas de acceso (solamente aplicable a SSH1) es necesario dar el nombre del usuario en el sistema cliente. La dirección de este sistema se supone disponible por medio de los protocolos subyacentes (TCP/IP).

3) Cuando se utilizan listas de acceso con autenticación de cliente, en SSH2 el cliente envía su nombre DNS completo, el nombre del usuario local, la clave pública del sistema cliente (y certificados, si tiene), la firma de una cadena de bytes que incluye el identificador de sesión, y el algoritmo con el que se ha generado esta firma. El servidor debe validar la clave pública y la firma para completar la autenticación.

4) En la autenticación con contraseña sólo es preciso enviar directamente la contraseña. Por motivos obvios, este método no debería estar permitido si el protocolo de la subcapa de transporte SSH utiliza el algoritmo de cifrado nulo.

5) La autenticación basada en clave pública es parecida a la de listas de acceso con autenticación de cliente. En SSH2 el cliente debe enviar un mensaje que contenga la clave pública del usuario (y los certificados, si tiene), el algoritmo que corresponde a esta clave y una firma en la cual interviene el identificador de sesión. El servidor dará por buena la autenticación si verifica correctamente la clave y la firma.

Opcionalmente, para evitar cálculos e interacciones innecesarias con el usuario, el cliente puede enviar antes un mensaje con la misma información pero sin la firma, para que el servidor responda si la clave pública que se le ofrece es aceptable.

Contraseña caducada

SSH2 prevé el caso de que la contraseña del usuario en el sistema servidor esté caducada y sea necesario cambiarla antes de continuar. El cambio no se debería permitir si no se está utilizando ningún MAC (algoritmo nulo), porque un atacante podría modificar el mensaje que contiene la nueva contraseña.

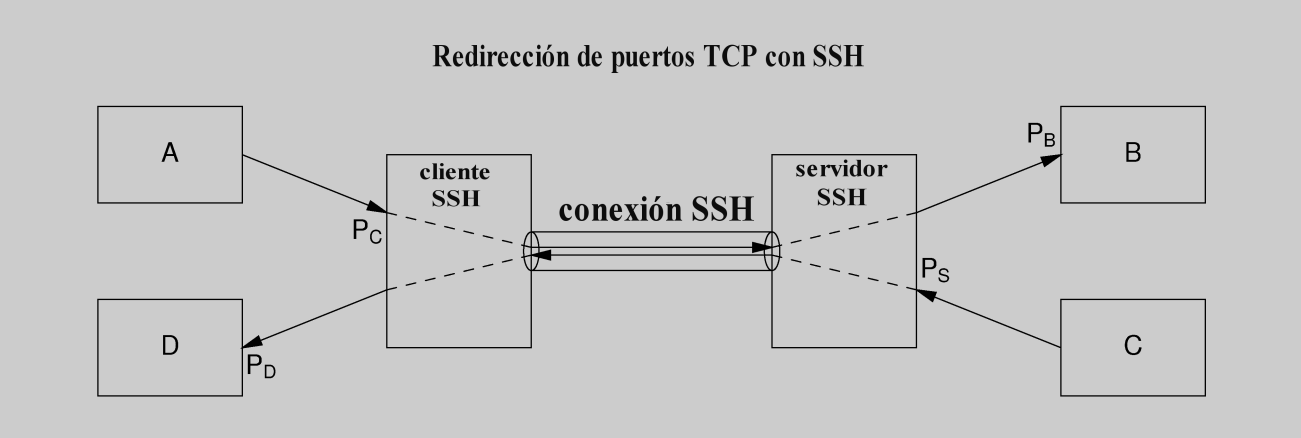
Cuando el proceso de autenticación se haya completado satisfactoriamente, en SSH2 se pasa al servicio que el cliente haya solicitado en su último mensaje `USERAUTH_REQUEST` (el que ha dado lugar a la autenticación correcta). Actualmente sólo existe un servicio definido, que es el de conexión.

1.2.4. El protocolo de conexión

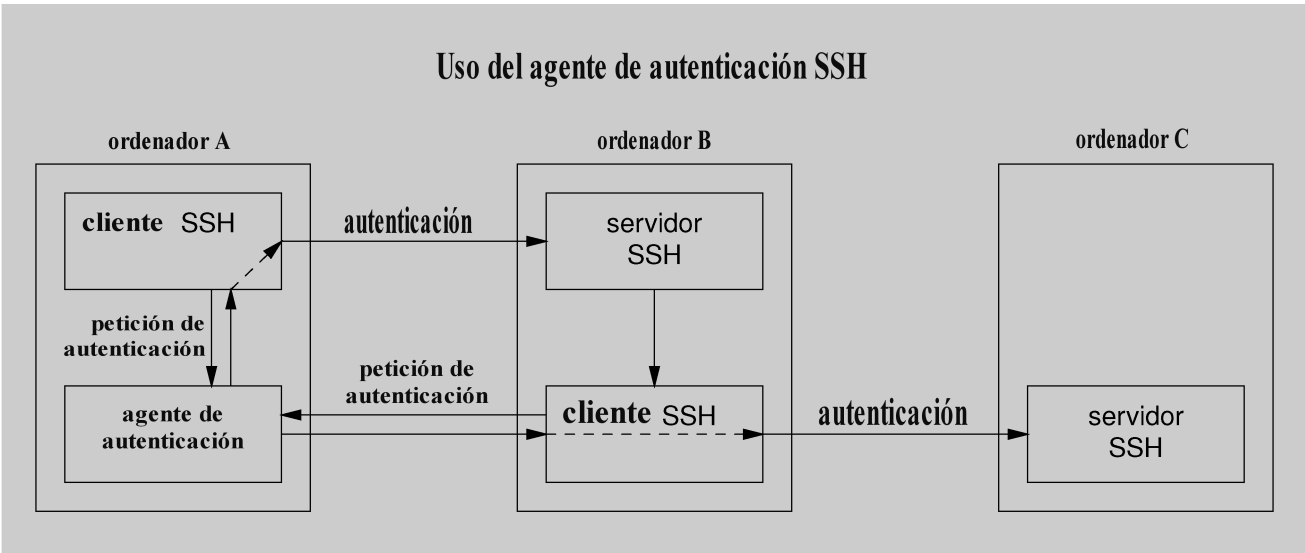
El protocolo de conexión gestiona las sesiones interactivas para la ejecución remota de comandos, mandando los datos de entrada de cliente a servidor y los de salida en sentido inverso. También se encarga de la redirección de puertos TCP.

Como muestra la siguiente figura, con la redirección TCP es posible lograr que las conexiones que se realicen a un determinado puerto P_C del cliente sean redirigidas a un puerto P_B de un ordenador B desde el servidor, o que las conexiones que se realicen a un determinado puerto P_S del servidor sean redirigidas a un puerto P_D de un ordenador D desde el cliente. De esta forma la

conexión SSH se puede utilizar, por ejemplo, como túnel de otras conexiones a través de un cortafuegos que esté situado entre el cliente y el servidor SSH.



Además, SSH contempla la posibilidad de utilizar lo que se conoce como **agente de autenticación**. Este agente es un proceso que permite automatizar la autenticación del usuario basada en claves públicas cuando es necesario realizarla desde un ordenador remoto. Por ejemplo, supongamos la situación de la siguiente figura:



El usuario del ordenador A utiliza un cliente SSH para conectarse al ordenador B y trabajar con una sesión interactiva. El ordenador A puede ser, por ejemplo, un PC portátil en el que el usuario tiene guardada su clave privada y del que no desea que salga nunca esta calve. Entonces resulta que necesita establecer una conexión SSH (por ejemplo, otra sesión interactiva) desde el ordenador B al ordenador C, y se tiene que autenticar con su clave personal.

El cliente del ordenador B, en lugar de realizar directamente la autenticación, para lo que necesitaría la clave privada del usuario, pide al agente del ordenador A que firme el mensaje adecuado para demostrar que posee la clave privada. Este esquema también se puede utilizar localmente por parte de los clientes del mismo ordenador A.

Cada sesión interactiva, conexión TCP redirigida o conexión a un agente es un **canal**. Pueden existir distintos canales abiertos en una misma conexión SSH, cada uno identificado con un número en cada extremo (los números asignados a un canal en el cliente y en el servidor pueden ser diferentes).

SSH2 prevé catorce tipos de mensajes que se pueden enviar durante la fase de conexión. Éstas son algunas de las funciones que permiten realizar los mensajes:

Abrir un canal. Se pueden abrir canales de distintos tipos: sesión interactiva, canal de ventanas X, conexión TCP redirigida o conexión con un agente de autenticación.

Configurar parámetros del canal. Antes de empezar una sesión interactiva el cliente puede especificar si necesita que se le asigne un pseudoterminal en el servidor, como hace el programa `rlogin` de Unix (en cambio, el programa `rsh` no lo necesita) y, si es así, con qué parámetros (tipo de terminal, dimensiones, caracteres de control, etc.). Existen otros mensajes para indicar si se quiere conexión con el agente de autenticación o redirección de conexiones de ventanas X.

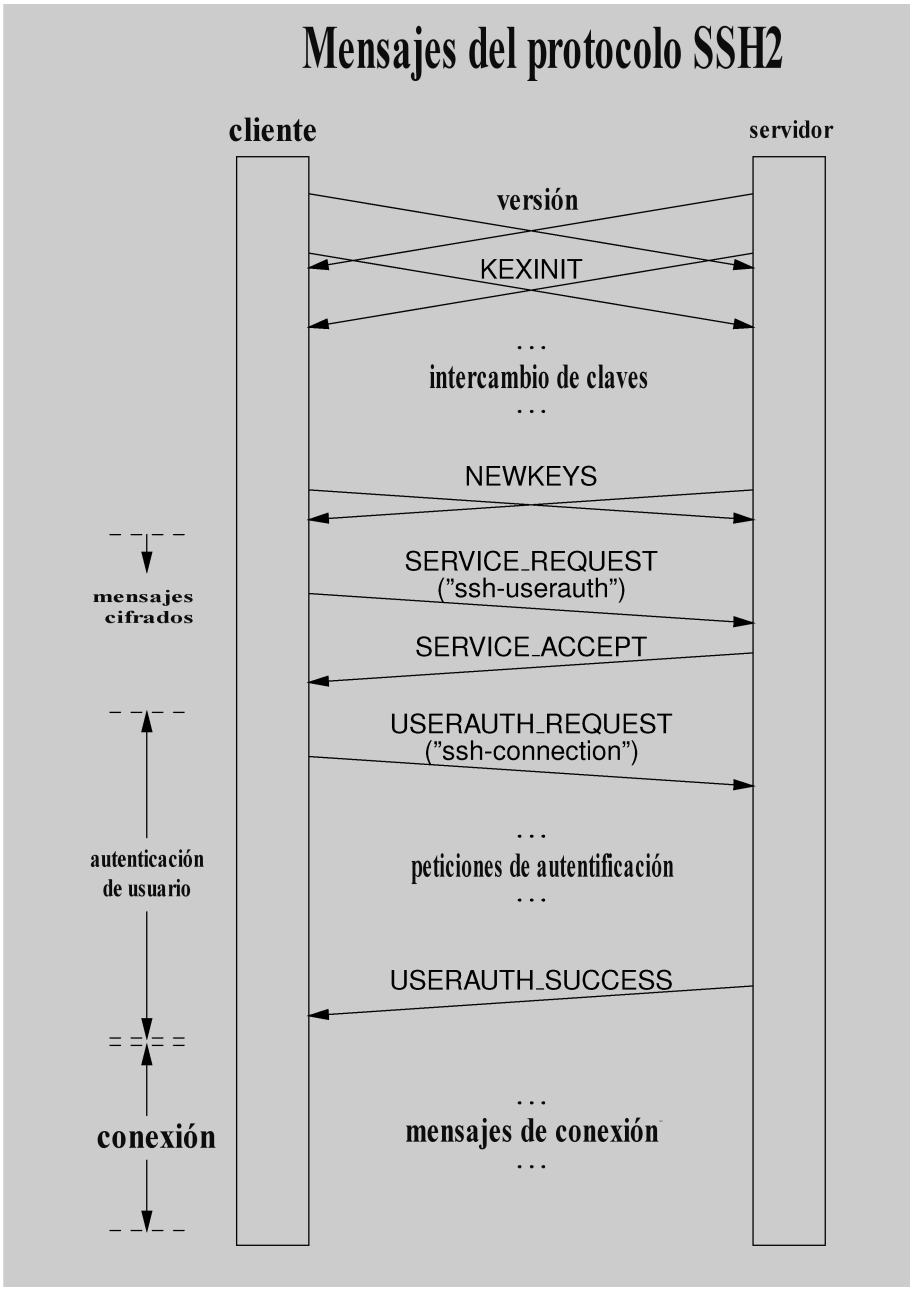
Empezar sesión interactiva. Una vez configurados los parámetros necesarios, el cliente puede dar el nombre de un comando que se deba ejecutar en el servidor (como en `rsh`), o bien indicar qué quiere ejecutar un intérprete de comandos (como en `rlogin`). Además de un proceso remoto, en SSH2 también existe la posibilidad de iniciar un "subsistema", como puede ser, por ejemplo, una transferencia de ficheros.

Enviar datos. En SSH2 existen dos tipos de mensaje con este fin: uno para enviar datos normales en cualquier sentido y para cualquier canal (incluyendo las sesiones interactivas), y otro para enviar datos especiales (por ejemplo, los de la salida de error `stderr`). Además de los datos de la sesión, el cliente también puede enviar un mensaje para indicar que ha recibido una señal o que se ha producido un cambio en las dimensiones del terminal.

Cerrar canal. Cuando termina la ejecución normal del proceso o intérprete de comandos, el servidor envía un mensaje indicando el código de salida (el valor numérico que devuelve el proceso). Si ha terminado a causa de una señal, en SSH2 envía un mensaje con el número de señal. Existen otros mensajes que sirven para indicar que ya no hay más datos de entrada, para solicitar el cierre de un canal desde un extremo, y para confirmar el cierre desde el otro extremo.

Otras operaciones. (no asociadas a un canal abierto). El cliente puede pedir que las conexiones que lleguen a un determinado puerto TCP del servidor le sean redirigidas, para poderlas reenviar a otra dirección.

La siguiente figura resume el intercambio de mensajes en SSH2.



1.3. Ataques contra el protocolo SSH

Muchas de las consideraciones sobre la protección que proporciona SSL/TLS son aplicables también al protocolo SSH. Este protocolo está diseñado para que un atacante no pueda leer el contenido de los mensajes ni alterarlos, y tampoco cambiar la secuencia de los mismos.

La confidencialidad queda garantizada con el método de intercambio de claves basado en criptografía de clave pública, que protege contra los ataques "del hombre a medio camino" que hemos visto en el apartado sobre SSL/TLS. Además, este método permite que el cliente se asegure de que se está conectando al servidor auténtico. Para comprobar que la clave pública que envía el servidor es realmente la suya, se pueden usar certificados, o bien una base de datos local del cliente en la que estén guardadas las claves de los servidores reconocidos. Y para autenticar al usuario mediante una clave pública (la suya o la del cliente desde el cual se conecta, dependiendo del método de autenticación), también existen las dos opciones: certificados o una base de datos de claves en el servidor.

Mejoras en SSH2

El protocolo SSH1 era vulnerable a ataques de repetición, eliminación o reordenación de paquetes porque no utilizaba números de secuencia, y también al reenvío de paquetes en sentido contrario si se utilizaba una sola clave de cifrado para ambos sentidos. Estos problemas ya no están presentes en SSH2.

Si no se usan certificados, el protocolo contempla la posibilidad (aunque no se recomienda) de dar por buena la clave pública de un servidor la primera vez que se establezca una conexión, sin necesidad de ninguna comunicación previa. Esto no es apropiado en un entorno donde la seguridad sea crítica, porque representa una vulnerabilidad a ataques "de hombre a medio camino". En otros entornos, y mientras no se disponga de una infraestructura de claves ampliamente extendida, aceptar directamente claves recibidas por primera vez puede suponer un equilibrio entre comodidad de uso y seguridad.

Una característica interesante añadida a SSH2 es que las longitudes de los paquetes se envían cifradas. Un atacante que vea los datos intercambiados como un flujo de bytes no puede saber dónde empieza y dónde acaba cada paquete SSH2 (si tiene acceso al nivel de paquetes TCP puede intentar hacer deducciones, pero sin una certeza absoluta). Esto, juntamente con la posibilidad de incluir *padding* arbitrario (hasta 255 bytes) y enviar mensajes `IGNORE`, puede servir para ocultar las características del tráfico y dificultar los ataques con texto claro conocido.

Por otra parte, merece la pena señalar que los métodos de autenticación de usuario mediante listas de acceso se basan en la confianza del servidor en el administrador del sistema cliente (del mismo modo que los protocolos `rsh` y `rlogin`):

- Cuando no se autentica el sistema cliente (posibilidad contemplada solamente en SSH1), el servidor sólo tiene que aceptar conexiones que provengan de un puerto TCP privilegiado (menor que 1.024) para que a un usuario cualquiera no le sea fácil enviar paquetes suplantando la identidad de otro.
- Cuando hay autenticación del sistema cliente, el servidor confía que los usuarios no tendrán acceso a la clave privada de este sistema, porque si no podrían utilizarla para generar mensajes de autenticación con la identidad de otro usuario.

Finalmente, igual que pasa con SSL/TLS, el protocolo SSH está diseñado para ofrecer determinadas protecciones, pero el nivel de seguridad que proporcione

en cada caso vendrá dado por las implementaciones y por el uso que se haga del mismo. Se recomienda que se puedan deshabilitar o restringir las características (métodos de autenticación de usuario, redirección de puertos TCP, etc.) que en un determinado entorno impliquen alguna vulnerabilidad o posibilidad de abuso.

1.4. Aplicaciones que utilizan el protocolo SSH

Dado que el objetivo principal de SSH es permitir la ejecución remota de procesos al estilo de los programas `rsh` y `rlogin`, se pueden implementar, y de hecho se han implementado, otros programas (por ejemplo `ssh` y `slogin`) que hagan lo mismo, pero utilizando el protocolo SSH.

Los argumentos pueden ser los mismos: el nombre del servidor, "`-l` *`usuario`*" para especificar el nombre de usuario en el servidor, etc. Los programas también pueden estar configurados para utilizar distintos métodos de autenticación: el basado en los ficheros `.rhosts` y `/etc/hosts.equiv` funciona como en `rsh`/`rlogin`, y el basado en contraseña funciona como en `rlogin`. Si se utiliza autenticación del sistema cliente será preciso guardar su clave privada en algún lugar de acceso restringido. Y si la autenticación de usuario está basada en su clave pública, la clave privada correspondiente se deberá guardar protegida, normalmente cifrada con una clave simétrica derivada de una contraseña o de una *passphrase*.

La implementación original del programa `ssh`, en sus distinta versiones, admite argumentos de la forma "`-L` *`p1`*`:`*`adr`*`:`*`p2`*" y "`-R` *`p1`*`:`*`adr`*`:`*`p2`*" para especificar redirecciones TCP del puerto local (del cliente) o remoto (del servidor) *`p1`*, respectivamente, al puerto *`p2`* del ordenador *`adr`*.

Ejemplos de redirección de puertos TCP con SSH

1) Desde un ordenador llamado `cerca` podemos hacer:

```
ssh -L 5555:srv.lejos.com:23 -l admin medio
```

Si nos autenticamos correctamente, habremos establecido una conexión interactiva con el ordenador `medio` como usuario `admin`, y además, cualquier conexión al puerto 5555 de `cerca`, como ésta:

```
telnet cerca 5555
```

estará redirigida al puerto 23 (TELNET) del ordenador `srv.lejos.com` (pasando por `medio`, y con el tramo entre `cerca` y `medio` protegido con SSH).

2) Ésta sería una forma de proteger una conexión a un servidor HTTP mediante un túnel SSH, suponiendo que podamos autenticarnos ante este servidor:

```
ssh -L 5678:localhost:80 www.lejos.com
```

Un vez realizada esta operación, podemos introducir la dirección `http://localhost:5678/` en cualquier navegador web del ordenador local, y nos llevará automáticamente a la dirección `http://www.lejos.com/` con una conexión cifrada (y si en esta dirección existe una página HTML con referencias no absolutas a páginas del

mismo servidor, estas otras páginas también nos llegarán a través de SSH).

2. Correo electrónico seguro

El correo electrónico es una de las aplicaciones más utilizadas en las redes de computadores. En el caso de Internet, el protocolo sobre el que se sustenta la transferencia de mensajes, el SMTP, fue publicado en el estándar RFC 821 en 1982. Como su nombre indica, la principal característica de este protocolo es su simplicidad. Esto ha permitido que el SMTP, junto con el estándar para el formato de los mensajes (RFC 822) y la especificación MIME, sean la base tecnológica de la gran mayoría de los sistemas de correo actuales.

SMTP

SMTP es la sigla de *Simple Mail Transfer Protocol*.

MIME

MIME es la sigla de *Multipurpose Internet Mail Extensions*.

Esta gran virtud del SMTP, la simplicidad, es a su vez una fuente de muchos problemas de seguridad, ya que a un atacante puede resultarle sorprendentemente fácil capturar mensajes o enviar mensajes falsos en nombre de otros. En este apartado veremos algunas técnicas existentes para añadir seguridad al servicio de correo electrónico.

Si consideramos el correo electrónico como un protocolo de la capa de aplicación, una posibilidad para proteger los mensajes de correo sería utilizar la seguridad que pueden ofrecer las capas inferiores, como la de red o la de transporte. Por ejemplo, con el protocolo SMTP se puede negociar el uso del transporte seguro SSL/TLS, mediante un comando especial llamado `STARTTLS` (RFC 2487).

Pero en la transferencia de los mensajes pueden intervenir distintos agentes intermedios, y para realizar la comunicación segura de extremo a extremo sería necesario proteger todos los enlaces o intentar hacer una conexión directa entre el sistema del originador y los de los destinatarios. Por otro lado, la naturaleza *store-and-forward* (almacenamiento y reenvío) del servicio de correo electrónico implica que los mensajes sean vulnerables no sólo cuando se transfieren de un nodo intermedio a otro, sino también mientras están almacenados en estos nodos. Esto incluye el sistema de destino final: una vez el mensaje ha llegado al buzón del usuario, su contenido puede ser inspeccionado o modificado por terceros antes de que lo lea el destinatario legítimo, o incluso después de que ya lo haya leído.

Es por estos motivos que se han desarrollado métodos para proteger el correo electrónico en el mismo nivel de aplicación, independientemente del sistema de transporte utilizado. La idea es aplicar las funciones criptográficas necesarias al mensaje antes de entregarlo a los agentes de transferencia del servicio de correo, y éstos sólo deben hacerlo llegar a su destino de forma habitual.

De este modo, por un lado, se puede aprovechar la infraestructura de correo electrónico ya existente, sin necesidad de cambiar los servidores, etc., y por otro, la protección es efectiva durante todo el proceso, incluso mientras el mensaje esté almacenado en el buzón del destinatario.

La mayoría de los sistemas de correo electrónico seguro que se han propuesto siguen este modelo de incorporar la seguridad dentro de los propios mensajes sin modificar el protocolo de transferencia. Algunos de estos sistemas son:

- **PEM** (*Privacy Enhanced Mail*)

  Fue uno de los primeros sistemas de correo seguro que se desarrollaron: la primera versión se publicó en la especificación RFC 989. Estaba basado directamente en el estándar RFC 822, y solamente contemplaba el envío de mensajes de texto ASCII. Actualmente está en desuso, pero algunas de las técnicas que usaba se continúan utilizando actualmente en los sistemas más modernos.

- **MOSS** (*MIME Object Security Services*)

  Fue la primera especificación que utilizó el formato MIME para representar la información relacionada con la seguridad. Estaba basada en el sistema PEM, y se publicó en el documento RFC 1848.

- **PGP** (*Pretty Good Privacy*)

  Uno de los sistemas más populares para añadir confidencialidad y autenticación, no sólo al correo electrónico sino a cualquier tipo de datos. En muchos entornos es el estándar *de facto* para el intercambio seguro de información. Ha ido evolucionando y actualmente existen varias versiones, que incluyen variantes como PGP/MIME, OpenPGP y GnuPG.

- **S/MIME** (*Secure MIME*)

  Se trata de otro sistema que utiliza la tecnología MIME, en este caso basado en la sintaxis definida por el estándar PKCS #7. También dispone de una gran variedad de implementaciones disponibles.

En este apartado analizaremos algunos detalles de los sistemas S/MIME y PGP, pero antes veremos las características generales del correo electrónico seguro.

2.1. Seguridad en el correo electrónico

Cuando se añaden servicios de seguridad al correo electrónico es preciso tener en cuenta las siguientes consideraciones:

- En el correo electrónico la transmisión de mensajes se lleva a cabo de manera no interactiva, y por lo tanto no puede haber negociación de algoritmos o intercambio de claves cuando se envía un mensaje. Esto significa que posiblemente será necesario un paso adicional para obtener la clave requerida para comunicarse con un determinado corresponsal (consultar una base de datos de claves públicas, pedir al usuario que envíe su clave, etc.). Todos los sistemas de correo seguro deben prever algún mecanismo de **distribución de claves**.
- Una funcionalidad básica del correo electrónico es el envío de un mismo mensaje a **múltiples destinatarios**. Con el correo seguro, si es preciso utilizar parámetros criptográficos distintos para cada destinatario, una solución poco eficiente sería efectuar envíos separados. Pero si queremos aprovechar las capacidades de los sistemas de correo existentes, debemos utilizar una técnica que permita combinar en un solo mensaje toda la información necesaria para que pueda ser procesada por cada uno de los destinatarios.

Los servicios que proporcionan los sistemas de correo electrónico seguro son básicamente dos:

Otros servicios

Hay otros servicios que no pueden ofrecer todos los sistemas de correo seguro: confidencialidad del flujo de tráfico (que no se sepa quién manda mensajes a quién y cuándo, y qué longitud tienen), protección contra negación de recepción (que un usuario no pueda decir que no ha recibido un mensaje, o que un tercero no pueda borrar mensajes para que el destinatario no los lea) o contra ataques de repetición de mensajes, etc.

Confidencialidad. Mediante las técnicas de cifrado, se puede garantizar que un mensaje sólo podrá ser leído por sus destinatarios legítimos.

Autenticación de mensaje. Los mensajes pueden incluir un código de autenticación (un código MAC o una firma digital) para que los destinatarios puedan verificar que han sido generados por el originador auténtico, y que nadie los ha modificado o falsificado.

Cada uno de estos dos servicios puede basarse en técnicas criptográficas de clave simétrica o de clave pública.

2.1.1. Confidencialidad

Para que un originador A pueda enviar un mensaje cifrado a un destinatario B, es preciso que ambos hayan acordado el uso de una determinada **clave de intercambio** k_{AB}. Esto se puede realizar con una comunicación segura "fuera de línea" (por ejemplo, cara a cara), o bien con un mecanismo de distribución de claves.

La clave de intercambio k_{AB} puede ser una clave simétrica o una clave pública.

Si es simétrica, se puede utilizar la misma en ambos sentidos de la comunicación, de A a B y de B a A ($k_{AB} = k_{BA}$). El uso de claves de intercambio simétricas estaba contemplado en el estándar PEM, pero hoy no es muy habitual.

La situación más normal es que la clave de intercambio sea una clave pública, y entonces las claves correspondientes a un destinatario B son todas la misma: $k_{1B} = k_{2B} = k_{3B} = \ldots = k_{\text{pub}_B}$.

> Para cifrar un mensaje de correo se utiliza siempre un algoritmo de cifrado de clave simétrica, dado que los de clave pública son mucho más costosos, especialmente si el mensaje es largo.

El método que se utiliza para enviar un mensaje cifrado es el llamado **sobre digital**, que consiste en:

1) Generar aleatoriamente una clave de cifrado simétrica k_S, distinta para cada mensaje. Esta clave k_S se llama **clave de cifrado de contenido** o bien, por analogía con los protocolos de transporte, **clave de sesión**.

2) Cifrar el mensaje M con esta clave simétrica y obtener $C = E(k_S, M)$.

3) Para cada destinatario B_y del mensaje, cifrar la clave de sesión con la clave pública de este destinatario y obtener $K_{B_y} = E(k_{\text{pub}_{B_y}}, k_S)$.

4) Construir un nuevo mensaje añadiendo al mensaje cifrado C todas las claves cifradas K_{B_y}.

5) Enviar a los destinatarios este mensaje (el mismo para todos).

Por lo tanto, si un mensaje confidencial se tiene que transmitir a N destinatarios, no es necesario enviar N copias del mensaje cifradas con la clave de cada uno de ellos. Se puede utilizar la misma copia para todos, con lo cual es posible aprovechar los mecanismos ya existentes para enviar un mensaje a múltiples destinatarios.

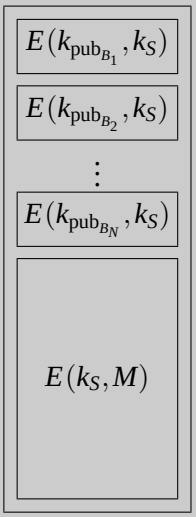
Sobre digital

El nombre de *sobre digital* proviene de la analogía con el correo tradicional: un mensaje de correo electrónico sin cifrar es como una postal, cuyo contenido puede leer todo el mundo, mientras que un mensaje cifrado de esta forma es como una carta con sobre, que sólo puede abrir la persona que figura como destinatario.

Mensajes a un único destinatario

La técnica del sobre digital se utiliza para los mensajes dirigidos a cualquier número de destinatarios, que también puede ser sólo uno.

Mensaje con sobre digital



En la recepción, cada destinatario B_y deberá seleccionar la K_{B_y} correspondiente a su clave pública, descifrarla con su clave privada $k_{\text{priv}_{B_y}}$ para obtener la clave de sesión k_S, y finalmente descifrar C con esta clave de sesión para recuperar el mensaje M.

Listas de distribución de correo

Las listas de distribución de correo son un caso especial, porque el originador de un mensaje puede no saber a qué destinatarios llegará. Una posibilidad es utilizar una única clave de intercambio simétrica, conocida por todos los miembros de la lista (con el problema que esto comporta de no garantizar la autenticidad). Otra posibilidad consiste en que el agente que expande la lista reciba los mensajes cifrados con una clave propia de la lista, y los reenvíe cifrados con las claves de cada destinatario.

2.1.2. Autenticación de mensaje

Para la autenticación de los mensajes también se pueden utilizar técnicas simétricas o de clave pública.

Las técnicas simétricas consisten en añadir al mensaje un código MAC, calculado con una clave secreta compartida con el destinatario. Esto significa que se debe calcular un código MAC distinto para cada destinatario, y además, que no hay protección contra un posible repudio por parte del originador.

Por este motivo, los sistemas de correo seguro actuales utilizan las técnicas de autenticación de clave pública, es decir, las firmas digitales.

La firma de un mensaje puede ser verificada por cualquier persona que lo reci-

ba y conozca la clave pública del firmante, y hasta puede reenviar el mensaje a otros usuarios, que también podrán comprobar su autenticidad. Y por otro lado, las firmas proporcionan el servicio de no repudio.

2.1.3. Compatibilidad con los sistemas de correo no seguro

Si queremos utilizar la infraestructura SMTP existente para el correo seguro, debemos tener presente que este protocolo impone ciertas restricciones para intentar maximizar la interoperabilidad entre las implementaciones, incluidas las más antiguas.

Algunas de estas restricciones son, por ejemplo, que en principio los mensajes sólo pueden contener caracteres ASCII, y que las líneas de los mensajes no pueden tener más de 1.000 caracteres. Actualmente muchos agentes SMTP son capaces de trabajar sin estas restricciones pero de todas formas debemos tenerlas en cuenta porque no sabemos por que implementaciones pueden pasar nuestros mensajes.

Además, SMTP define una representación para los mensajes que puede ser distinta de la representación local de cada sistema. El proceso que se encarga de enviar los mensajes tiene que transformarlos del formato local al formato SMTP en el momento de transferirlos. Y a la inversa, cuando llega un mensaje vía SMTP normalmente se convierte al formato local antes de almacenarlo en el buzón de los destinatarios.

Ejemplos de transformación a formato SMTP

Un ejemplo típico de transformación es el de los finales de línea: en SMTP se representan con los caracteres <CR><LF>, mientras que en Unix se representan solamente con <LF>.

Otro ejemplo: algunos lectores de correo en Unix, especialmente los más antiguos, interpretan que en un buzón de mensajes la secuencia de caracteres "`From ` " al inicio de línea indica el inicio de un nuevo mensaje. En estos sistemas, cuando llega un mensaje que contiene esta secuencia al inicio de una línea, se añade automáticamente el carácter "`>`" delante de la línea, (los lectores de correo más modernos utilizan el campo `Content-Length` de la cabecera para saber dónde termina cada mensaje y dónde empieza el siguiente).

Por lo tanto, cuando se deben aplicar operaciones criptográficas a un mensaje, es preciso hacerlo sobre una **codificación canónica** que sea convertible al formato local de forma no ambigua.

De este modo, si tenemos que mandar un mensaje confidencial, cifraremos la forma canónica del mensaje para que cuando el receptor la descifre pueda convertirla a su formato local. Y si tenemos que calcular un código de autenticación o una firma, lo haremos también sobre la forma canónica, para que

el receptor sepa exactamente a partir de qué datos se ha generado y pueda realizar su verificación.

Cada sistema de correo seguro puede definir su codificación canónica. Por ejemplo, los sistemas basados en MIME utilizan el modelo de la especificación RFC 2049. En el correo multimedia las reglas de codificación dependen del tipo de contenido, pero en el caso de mensajes de texto, por simplicidad, lo más normal es que la codificación canónica coincida con el formato SMTP, es decir, con caracteres ASCII y con líneas acabadas en <CR><LF>.

Por otra parte, existen agentes de correo que pueden introducir modificaciones en los mensajes para adaptarlos a sus restricciones. Algunos ejemplos son: poner a 0 el 8^{avo} bit de cada carácter, cortar las líneas demasiado largas insertando finales de línea, eliminar los espacios en blanco al final de línea, convertir los tabuladores en secuencias de espacios, etc.

Dado que la información criptográfica constará en general de datos arbitrarios, debemos utilizar mecanismos de protección del contenido para garantizar que ninguna de estas modificaciones afectará a los mensajes seguros. Éste es el mismo problema que se planteó en el correo MIME para enviar contenidos multimedia, y la solución consiste en utilizar una **codificación de transferencia**, como puede ser la codificación "base 64".

Codificación base 64

La codificación base 64 consiste en representar cada grupo de 6 bits con un carácter ASCII de un juego de 64 ($2^6 = 64$), formado por las letras, los dígitos y los símbolos "+" y "/".

2.2. S/MIME

S/MIME (*Secure MIME*) es una especificación de correo seguro basada en la norma PKCS #7, que fue desarrollada inicialmente por RSA Data Security.

El sistema de correo S/MIME utiliza la técnica MIME para trasmitir mensajes protegidos criptográficamente según el formato PKCS #7.

Uso de MIME en S/MIME

Dado que se utiliza la representación MIME, S/MIME se puede integrar con otros protocolos de aplicación aparte del correo electrónico que también hacen uso de MIME como, por ejemplo, HTTP.

Durante los primeros años se elaboraron distintos borradores de la especificación S/MIME, que fueron adoptados por varios implementadores independientes. Las versiones iniciales definían dos perfiles de uso: el normal, que no era exportable fuera de los Estados Unidos, y el restringido, que imponía un límite de 40 bits secretos en las claves simétricas. Por motivos de interoperabilidad, muchos de los sistemas S/MIME que se desarrollaron seguían el perfil restringido.

En 1998 se publicaron los documentos informativos RFC 2311 y 2312, que recogen las características comunes a la mayoría de implementaciones que

existían entonces, en lo que se conoce como versión 2 de S/MIME. En esta versión ya no se hace distinción entre perfil normal o restringido.

Uso de PKCS #7 en S/MIME v2

En la versión 2 de S/MIME se utiliza PKCS #7 versión 1.5 (RFC 2315).

Algoritmos previstos en S/MIME versión 2

La versión 2 de S/MIME prevé los siguientes algoritmos criptográficos:

- Para el cifrado simétrico: Triple DES y RC2.
- Para los resúmenes de mensajes (*hash*): MD5 y SHA-1.
- Para el cifrado de clave pública: RSA.

El estándar oficial de correo seguro S/MIME es la llamada versión 3, publicada en los documentos RFC 2632–2634, de junio de 1999. Una diferencia respecto a S/MIME versión 2 es que, en lugar de utilizar directamente el estándar PKCS #7, se basa en el nuevo estándar CMS (RFC 2630). El CMS mantiene la compatibilidad con PKCS #7 pero incluye nuevos métodos para el intercambio de claves, en particular el método Diffie-Hellman (descrito en el RFC 2631).

CMS

CMS es la sigla de *Cryptographic Message Syntax*.

La versión 3 está diseñada en general para ofrecer la máxima compatibilidad posible con la versión 2. De todas formas, conviene saber que la versión 3 soporta nuevos servicios, conocidos como ESS, que incluyen:

ESS

ESS es la sigla de *Enhanced Security Services*.

- Recibos firmados.
- Etiquetas de seguridad, que dan información sobre el nivel de sensibilidad del contenido de un mensaje (según una clasificación definida por una determinada política de seguridad).
- Listas de correo seguras.
- Certificados de firma, que permiten asociar directamente una firma con el certificado necesario para validarla.

2.2.1. El formato PKCS #7

PKCS #7 es un formato para representar mensajes protegidos criptográficamente. Cuando la protección está basada en criptografía de clave pública, en PKCS #7 se utilizan certificados X.509 para garantizar la autenticidad de las claves.

La norma PKCS #7 define unas estructuras de datos para representar cada uno de los campos que forman parte de un mensaje. A la hora de intercambiar estos datos se deben codificar según las reglas especificadas por la notación ASN.1 (la misma que se utiliza para representar los certificados X.509 y las CRL).

Ésta es la estructura general de un mensaje PKCS #7, descrita con la misma notación que utilizamos para los certificados ("opc." significa opcional y "rep." significa repetible):

Campo	**Tipo**
`contentType`	identificador único
`content` (opc.)	`Data`, `SignedData`, `EnvelopedData`, `SignedAndEnvelopedData`, `DigestedData`, o `EncryptedData`

El campo `contentType` es un identificador que indica cuál de las seis estructuras posibles hay en el campo `content`. Estas estructuras son:

1) `Data`: sirve para representar datos literales, sin aplicarles ninguna protección criptográfica.

2) `SignedData`: representa datos firmados digitalmente.

3) `EnvelopedData`: representa un mensaje con sobre digital (es decir, un mensaje cifrado simétricamente al que se añade la clave simétrica cifrada con la clave pública de cada destinatario).

4) `SignedAndEnvelopedData`: representa datos firmados y "cerrados" en un sobre digital.

5) `DigestedData`: representa datos a los cuales se les añade un resumen o *hash*.

6) `EncryptedData`: representa datos cifrados con clave secreta.

El campo `content` es opcional, porque en ciertos casos existe la posibilidad de que los datos de un mensaje no estén dentro del propio mensaje, sino en algún otro lugar.

De estos seis posibles tipos de contenido, los tres últimos no se utilizan en S/MIME: para datos firmados y con sobre se utiliza una combinación de `SignedData` y `EnvelopedData`, y los mensajes que sólo contienen datos con *hash* o datos cifrados simétricamente no se envían nunca con correo electrónico seguro.

Por lo tanto, los tipos de contenido PKCS #7 que puede haber en un mensaje S/MIME son `Data`, `SignedData` o `EnvelopedData`.

1) El tipo `Data`

Si el contenido PKCS #7 es de tipo `Data`, no está estructurado de ningún modo especial, sino que simplemente consiste en una secuencia de bytes. Cuando se utiliza en S/MIME, su contenido debe ser una **parte de mensaje MIME**, con sus cabeceras y su cuerpo en forma canónica.

El tipo `Data` no aparece nunca solo en un mensaje S/MIME, sino que siempre se usa en combinación con alguno de los otros tipos PKCS #7, es decir, con `SignedData` o con `EnvelopedData`.

2) El tipo `SignedData`

El tipo `SignedData` básicamente contiene datos, representados recursivamente con otro mensaje PKCS #7, y la firma de estos datos generada por uno o más firmantes. La estructura del contenido de tipo `SignedData` es la siguiente:

Campo	Tipo
`version`	entero
`digestAlgorithms` (rep.)	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`contentInfo`	mensaje PKCS #7
`certificates` (opc. rep.)	certificado X.509
`crls` (opc. rep.)	CRL
`signerInfos` (rep.)	
`version`	entero
`issuerAndSerialNumber`	
`issuer`	DN
`serialNumber`	entero
`digestAlgorithm`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`authenticatedAttributes` (opc. rep.)	atributo X.501
`digestEncryptionAlgorithm`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`encryptedDigest`	cadena de bytes
`unauthenticatedAttributes` (opc. rep.)	atributo X.501

El significado de cada campo es el siguiente:

- El campo `version` indica la versión del formato de la estructura `SignedData`.
- El campo `digestAlgorithms` es una lista de los algoritmos de resumen o *hash* que han utilizado los firmantes para firmar los datos.
- El campo `contentInfo` contiene los datos que hay que firmar, rep-

Algoritmos de *hash*

El campo `digestAlgorithms` aparece antes que los datos para facilitar el procesado de la estructura `SignedData` en un solo paso: sabiendo cuáles son los algoritmos de *hash*, a medida que se leen los datos se pueden ir calculando los resúmenes.

resentados en forma de mensaje PKCS #7. Este mensaje normalmente será de tipo `Data` (para firmar datos literales) o de tipo `EnvelopedData` (para firmar un mensaje confidencial).

- En el campo `certificates` hay certificados o cadenas de certificados que pueden ser útiles para verificar la autenticidad de las claves públicas de los firmantes.
- En el campo `crls` aparece una lista de CRLs que se pueden usar juntamente con los certificados del campo anterior.
- El campo `signerInfos` contiene una estructura para cada firmante, con los siguientes subcampos:

– `version` indica la versión del formato de esta estructura.

– `issuerAndSerialNumber` sirve para saber cual es la clave pública del firmante. En lugar de especificar directamente la clave, se da el nombre de una CA y un número de serie de certificado. Estos datos identifican de forma única un certificado (una misma CA no puede generar dos certificados con el mismo número de serie), y en este certificado, que puede ser uno de los que hay en el campo `certificates`, debe haber la clave pública del firmante.

– `digestAlgorithm` es el algoritmo de *hash* que ha usado este firmante (tiene que ser uno de los que había en el campo `digestAlgorithms` del principio).

– `authenticatedAttributes` es un conjunto de atributos que se añaden a los datos sobre los cuales se calcula la firma.

– `digestEncryptionAlgorithm` es el algoritmo con el que el firmante ha cifrado el *hash* para calcular la firma.

– `encryptedDigest` es la firma, es decir, el *hash* cifrado con la clave privada del firmante.

– `unauthenticatedAttributes` es un conjunto adicional de atributos que no intervienen en la firma.

Identificación de las claves públicas

El método que usa PKCS #7 para identificar una clave pública, a partir de un nombre de CA y un número de serie, se definió de esta forma por compatibilidad con el sistema de correo seguro PEM.

Como podemos ver, PKCS #7 permite que cada firmante añada atributos a su firma, que pueden ser autenticados o no autenticados. Cada uno de estos atributos sigue la estructura definida en la Recomendación X.501, que simplemente consta de un nombre de atributo y de uno o más valores.

Cuando hay atributos autenticados, uno de ellos tiene que ser obligatoriamente un atributo llamado `messageDigest`, que tiene como valor el *hash* del campo `contentInfo`. En este caso, la firma se calcula a partir del *hash* de todo el subcampo `authenticatedAttributes`. De este modo se puede comprobar la autenticidad de los datos del mensaje (`contentInfo`) y del resto de atributos autenticados.

Ejemplo de atributo autenticado

Un ejemplo típico de atributo autenticado es el atributo `signingTime`, que indica cuándo se generó la firma.

Cuando no hay atributos autenticados, la firma se calcula simplemente a partir del *hash* del campo `contentInfo`.

3) El tipo `EnvelopedData`

El tipo `EnvelopedData` contiene datos con sobre digital. Los datos corresponden recursivamente a otro mensaje PKCS #7. La estructura del contenido de tipo `EnvelopedData` es la siguiente:

Campo	Tipo
`version`	entero
`recipientInfos` (rep.)	
`  version`	entero
`  issuerAndSerialNumber`	
`  issuer`	DN
`  serialNumber`	entero
`  keyEncryptionAlgorithm`	
`  algorithm`	identificador único
`  parameters`	(depende del algoritmo)
`  encryptedKey`	cadena de bytes
`encryptedContentInfo`	
`  contentType`	identificador único
`  contentEncryptionAlgorithm`	
`  algorithm`	identificador único
`  parameters`	(depende del algoritmo)
`  encryptedContent` (opc.)	cadena de bytes

El significado de cada campo es el siguiente:

- El campo `version` indica la versión del formato de la estructura `EnvelopedData`.
- El campo `recipientInfos` contiene una estructura para cada destinatario del mensaje, con los siguientes subcampos:

– `version` indica la versión del formato de esta estructura.

– `issuerAndSerialNumber` sirve para saber a qué destinatario corresponde esta estructura. Cada destinatario debe buscar entre los elementos del campo `recipientInfos` el que tenga este subcampo igual a la CA y el número de serie de su certificado.

– `keyEncryptionAlgorithm` indica con qué algoritmo de clave pública se ha cifrado la clave de sesión.

– `encryptedKey` es la clave de sesión cifrada con la clave pública de este destinatario.

- En el campo `encryptedContentInfo` hay la información sobre los datos cifrados, con los siguientes subcampos:

– `contentType` indica que tipo de mensaje PKCS #7 hay en los datos cifrados: normalmente será `Data` (cuando se cifran datos literales) o

`SignedData` (cuando se cifra un mensaje firmado).

– `contentEncryptionAlgorithm` es el algoritmo simétrico con el que se han cifrado los datos (utilizando la clave de sesión).

– `encryptedContent` son los datos (es decir, un mensaje PKCS #7) cifrados.

Como hemos visto, los contenidos de tipo `SignedData` y `EnvelopedData` contienen recursivamente otros mensajes PKCS #7. La combinación que se puede encontrar en un mensaje S/MIME será una de estas cuatro:

- `SignedData`[`Data`] (mensaje con datos firmados).
- `SignedData`[`EnvelopedData`[`Data`]] (mensaje con datos cifrados y firmados).
- `EnvelopedData`[`Data`] (mensaje con datos cifrados).
- `EnvelopedData`[`SignedData`[`Data`]] (mensaje con datos firmados y cifrados).

Podemos ver, por lo tanto, que si un mensaje se tiene que firmar y cifrar, se pueden realizar las dos operaciones en cualquier orden, según interese. Por ejemplo, firmar primero y cifrar después permite que el destinatario guarde el mensaje descifrado para verificaciones posteriores, posiblemente por parte de terceros. Cifrar primero y firmar después permite verificar la autenticidad de un mensaje sin necesidad de descifrarlo.

2.2.2. Formato de los mensajes S/MIME

Un mensaje S/MIME es un mensaje MIME con las siguientes características:

- Su tipo de contenido (campo `Content-Type` de la cabecera MIME) es "`application/pkcs7-mime`".
- En su cuerpo hay una estructura PKCS #7 codificada según la notación ASN.1.

Para los mensajes S/MIME que sólo estén firmados existe una representación alternativa, llamada **firma en claro**, que veremos más adelante.

Compatibilidad con versiones anteriores de S/MIME

Para los tipos de contenido, como "`pkcs7-mime`" y otros que veremos a continuación, las primeras versiones de S/MIME utilizaban nombres que empezaban por "`x-`".

En las versiones experimentales de algunos protocolos es habitual utilizar un prefijo como éste para representar valores que aún no están estandarizados. Por compati-

bilidad con estas versiones, las aplicaciones de correo S/MIME deberían tomar en consideración los nombres antiguos equivalentes a los nombres sin prefijo, como por ejemplo:

Nombre antiguo	Nombre actual
`x-pkcs7-mime`	`pkcs7-mime`
`x-pkcs7-signature`	`pkcs7-signature`
`x-pkcs10`	`pkcs10`

La cabecera MIME `Content-Type`, además del valor "`application/pkcs7-mime`", debe tener al menos uno de estos dos parámetros:

- `smime-type`: indica el tipo de contenido PKCS #7 que hay en el cuerpo del mensaje.
- `name`: indica un nombre del fichero asociado al contenido del mensaje.

Fichero asociado a un mensaje S/MIME

El parámetro `name` sirve para mantener la compatibilidad con las primeras versiones de S/MIME, en las cuales no estaba definido el parámetro `smime-type`. En estas versiones, la especificación del tipo de contenido PKCS #7 se realizaba con la extensión de un nombre de fichero. La parte del nombre que haya antes de la extensión es indiferente, pero por convencionalmente suele ser "`smime`".

Para especificar este nombre de fichero se puede utilizar el parámetro `name` de la cabecera `Content-Type`, y también el parámetro `filename` de la cabecera MIME `Content-Disposition` (definida en la especificación RFC 2183), con el valor de esta cabecera igual a "`attachment`".

Además, dado que el cuerpo del mensaje son datos binarios (la representación ASN.1 de una estructura PKCS #7), normalmente habrá una cabecera `Content-Transfer-Encoding` con valor "`base64`" para indicar que estos datos están codificados en base 64.

Existen tres formatos básicos de mensajes S/MIME: los mensajes con sobre digital, los mensajes firmados, y los mensajes firmados en claro.

1) Mensajes S/MIME con sobre digital

Un mensaje S/MIME con sobre digital tiene las siguientes características:

- En el cuerpo del mensaje hay una estructura PKCS #7 con tipo de contenido `EnvelopedData`.
- El valor del parámetro `smime-type` es "`enveloped-data`".
- Si se especifica un nombre de fichero asociado, su extensión es `.p7m` (por ejemplo, "`smime.p7m`").

Mensajes firmados y cifrados

El contenido `EnvelopedData` que hay en un mensaje S/MIME con sobre digital puede contener una estructura `SignedData`: entonces se trata de un mensaje firmado y cifrado.

Éste es un ejemplo de mensaje S/MIME con sobre digital:

```
Date: Mon, 1 Mar 2004 11:46:10 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 1
To: usuario-2@uoc.edu
MIME-Version: 1.0
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
        name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"

MIAGCSqGSIb3DQEHA6CAMIACAQAxgDBzAgEAMC8wKjELMAkGA1UEBhMCRVMxDDAKBgNVBAoT
A1VPQzENMAsGA1UEAxMEQ0EtMQIBBjANBgkqhkiG9w0BAQEFAAQuCYHs970aZmmqKTr3gemZ
LzHVtB266O/TrIv4shSvos8Ko8mUSQGov0JSIugmeDBzAgEAMC8wKjELMAkGA1UEBhMCRVMx
DDAKBgNVBAoTA1VPQzENMAsGA1UEAxMEQ0EtMQIBBzANBgkqhkiG9w0BAQEFAAQuC14oIhps
+mh8Wxp79A81uv2ltG3vt6J9UdJQcrDL92wD/jpw1IKpoR224LT4PQAAMIAGCSqGSIb3DQEH
ATARBgUrDgMCBwQIZbTj6XqCRkGggARAF8K8apgPtK7JPS1OaxfHMDXYTdEG92QXfAdTPetA
FGuPfxpJrQwX2omWuodVxP7PnWT2N5KwE1oc6faJY/zG0AAAAAAAAAAAAAA=
```


2) Mensajes S/MIME firmados

Un mensaje S/MIME firmado tiene un formato análogo al de los mensajes con sobre digital. Sus características son:

- En el cuerpo del mensaje hay una estructura PKCS #7 con tipo de contenido `SignedData`.
- El valor del parámetro `smime-type` es "`signed-data`".
- Si se especifica un nombre de fichero asociado, su extensión es la misma que en los mensajes con sobre digital, es decir `.p7m` (por ejemplo, "`smime.p7m`").

Mensajes cifrados y firmados

El contenido `SignedData` que hay en un mensaje S/MIME firmado puede contener una estructura `Enveloped`: entonces se trata de un mensaje cifrado y firmado.

Este es un ejemplo de mensaje S/MIME firmado:


```
Date: Mon, 1 Mar 2004 11:47:25 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 2
To: usuario-2@uoc.edu
MIME-Version: 1.0
Content-Type: application/pkcs7-mime; smime-type=signed-data;
        name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"

MIAGCSqGSIb3DQEHAqCAMIACAQExDjAMBggqhkiG9w0CBQUAMIAGCSqGSIb3DQEHAaCAJIAE
OUNvbnRlbnQtVHlwZTogdGV4dC9wbGFpbg0KDQpFeGVtcGxlIGRlIG1pc3NhdGdlIHNpZ25h
dC4NCgAAAAAAAKCAMIIBSDCCAQWgAwIBAgIBBjANBgkqhkiG9w0BAQQFADAqMQswCQYDVQQG
EwJFUzEMMAoGA1UEChMDVU9DMQ0wCwYDVQQDEwRDQS0xMB4XDTAwMDkwMTAwMDAwMFoXDTEw
MDkwMTAwMDAwMFowVTELMAkGA1UEBhMCRVMxDDAKBgNVBAoTA1VPQzElMCMGCSqGSIb3DQEJ
ARYWdXN1YXJpLTFAY2FtcHVzLnVvYy5lczERMA8GA1UEAxMIdXN1YXJpLTEwSTANBgkqhkiG
9w0BAQEFAAM4ADA1Ai4MNRLOaI30lrRQjyyznTjQTs/vLveiFadRiGKlVNnsFkGx/EwHdFDy
7z4CpbtnAgMBAAEwDQYJKoZIhvcNAQEEBQADLgACRZrDsL/MJv9VZdbxNMpbjKcwwFPPVG9L
TqOZ8sTdAF09UnFsSj5jE0ABAPEAADGAMIGBAgEBMC8wKjELMAkGA1UEBhMCRVMxDDAKBgNV
BAoTA1VPQzENMAsGA1UEAxMEQ0EtMQIBBjAMBggqhkiG9w0CBQUAMA0GCSqGSIb3DQEBAQUA
BC4DMwI+4fvRqBPhFj/wB7gI+Or7nSYfkqP1fxbjdTqwu9B5jsnxDIs+PUYsboQIAAAAAAAA
AAA=
```


3) Mensajes S/MIME firmados en claro

Cuando se envía un mensaje firmado, los receptores que utilicen un lector de correo apropiado podrán leer el mensaje y verificar la firma. Muchas veces interesa que el mensaje pueda ser leído por todos, aunque no se disponga de un lector con soporte para correo seguro.

Lo que se hace en estos casos es formar un mensaje con dos partes: la primera se representa como un mensaje normal, que puede ser leído por cualquier cliente de correo, y la segunda es la firma de la primera. Un mensaje de este tipo se llama **mensaje firmado en claro**.

De esta forma, quien disponga de un lector de correo seguro podrá leer el mensaje y verificar la firma, y quien utilice un lector tradicional podrá igualmente leer el mensaje, aunque no podrá comprobar si la firma es auténtica.

Una de las especificaciones del estándar MIME, el RFC 1847, define la forma de añadir una firma a un mensaje MIME. El mensaje resultante tiene las siguientes características:

- El tipo de contenido del mensaje (cabecera MIME `Content-Type`) es "`multipart/signed`".
- La cabecera `Content-Type` tiene tres parámetros obligatorios.

– `boundary`: como en todos los mensajes de tipo `multipart`, este parámetro indica el delimitador que se utiliza para separar las partes.

– `protocol`: indica el tipo de contenido que hay en la parte del mensaje que contiene la firma.

– `micalg`: indica el algoritmo o algoritmos de *hash*, también llamado MIC, con los que está calculada la firma (para facilitar el procesado del mensaje en un solo paso).

- El cuerpo del mensaje consta de dos partes MIME.

– La primera parte es el mensaje sobre el cual se calcula la firma. Tiene la estructura de una parte MIME, con cabeceras y cuerpo. Como caso particular, puede tratarse de un mensaje MIME multiparte si, por ejemplo, se quiere firmar un mensaje con imágenes o documentos anexos.

– La segunda parte contiene la firma, calculada a partir de la forma canónica de la parte anterior. Esta segunda parte también debe tener la estructura de una parte MIME, y el valor de su cabecera `Content-Type` debe ser igual al del parámetro `protocol` de la cabecera `Content-Type` del mensaje principal.

MIC

MIC es la sigla de *Message Integrity Code*, que es la nomenclatura que utilizaba el sistema PEM para referirse al método de autenticación de mensaje (cuando se utilizan claves públicas, este método es una firma digital).

Cuando se aplica esta técnica MIME de firmas en claro a S/MIME, las características de los mensajes son las siguientes:

- El parámetro `protocol` y, por lo tanto la cabecera `Content-Type` de la segunda parte del mensaje, debe tener el valor "`application/pkcs7-signature`".
- Si se especifica un número de fichero asociado a la firma, es decir, a la segunda parte, su extensión es `.p7s` (por ejemplo, "`smime.p7s`").

- En el cuerpo de la segunda parte del mensaje hay una estructura PKCS #7 con tipo de contenido `SignedData`, pero sin el campo `content` en el campo `contentInfo`.

A la hora de enviar un mensaje S/MIME con firma en claro, en la estructura `SignedData` de la segunda parte no se incluyen los datos firmados porque ya están en la primera parte del mensaje. En el momento de verificar la firma, el receptor debe actuar igual que si en la estructura `SignedData` de la segunda parte hubieran los datos de la primera parte.

Campo `content` no presente

Recordad que en un mensaje PKCS #7 el campo `content` es opcional y, por lo tanto existe la posibilidad de que no esté presente. En los mensajes S/MIME con firma en claro se aprovecha esta posibilidad.

En este caso, se dice que la estructura PKCS #7 tiene datos firmados **no incluidos** (*detached*), a diferencia del otro formato de mensajes firmados, en el cual la estructura PKCS #7 tiene los datos firmados **incluidos** (*attached*).

Éste es un ejemplo de mensaje S/MIME firmado en claro:

```
Date: Mon, 1 Mar 2004 11:47:40 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 3
To: usuario-2@uoc.edu
MIME-Version: 1.0
Content-Type: multipart/signed; boundary="20040301104740";
        protocol="application/pkcs7-signature"; micalg=md5

--20040301104740
Content-Type: text/plain

Ejemplo de mensaje firmado.

--20040301104740
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIAGCSqGSIb3DQEHAqCAMIACAQExDjAMBggqhkiG9w0CBQUAMIAGCSqGSIb3DQEHAQAAoIAw
ggFIMIIBBaADAgECAgEGMA0GCSqGSIb3DQEBBAUAMCoxCzAJBgNVBAYTAkVTMQwwCgYDVQQK
EwNVT0MxDTALBgNVBAMTBENBLTEwHhcNMDAwOTAxMDAwMDAwWhcNMTAwOTAxMDAwMDAwWjBV
MQswCQYDVQQGEwJFUzEMMAoGA1UEChMDVU9DMSUwIwYJKoZIhvcNAQkBFhZ1c3VhcmktMUBj
YW1wdXMudW9jLmVzMREwDwYDVQQDEwh1c3VhcmktMTBJMA0GCSqGSIb3DQEBAQUAAzgAMDUC
Lgw1Es5ojfSWtFCPLLOdONBOz+8u96IVp1GIYqVU2ewWQbH8TAd0UPLvPgKlu2cCAwEAATAN
BgkqhkiG9w0BAQQFAAMuAAJFmsOwv8wm/1Vl1vE0yluMpzDAU89Ub0tOo5nyxN0AXT1ScWxK
PmMTQAEA8QAAMYAwgYECAQEwLzAqMQswCQYDVQQGEwJFUzEMMAoGA1UEChMDVU9DMQ0wCwYD
VQQDEwRDQS0xAgEGMAwGCCqGSIb3DQIFBQAwDQYJKoZIhvcNAQEBBQAELgMzAj7h+9GoE+EW
P/AHuAj46vudJh+So/V/FuN1OrC70HmOyfEMiz49RixuhAgAAAAAAAAAAA==
--20040301104740--
```

2.2.3. Distribución de claves con S/MIME

Como hemos visto hasta este punto, el método que se utiliza en PKCS #7 y por lo tanto, en S/MIME, para identificar los usuarios y sus claves públicas es por medio de sus certificados X.509.

Esto quiere decir que un usuario no necesita verificar las identidades de los demás porque de esto ya se encargan las autoridades de certificación. Lo único que debe hacer el usuario es comprobar si el certificado (o cadena de certificados) de su corresponsal está firmado por una CA reconocida y es un certificado válido, es decir, no está caducado ni revocado.

Idealmente, la distribución de los certificados de los usuarios se debería poder realizar mediante el servicio de directorio X.500, pero si este servicio no está disponible se pueden utilizar otros métodos alternativos.

S/MIME define un tipo especial de mensaje que sirve para transportar certificados o listas de revocación. Se trata de un mensaje S/MIME con las siguientes características:

- En el cuerpo del mensaje hay una estructura PKCS #7 con tipo de contenido `SignedData`, pero sin datos firmados (campo `content` del elemento `contentInfo`) ni firmas (campo `signerInfos` con 0 elementos). Por lo tanto, los campos con información útil son `certificates` y `crls`.
- El valor del parámetro `smime-type` es "`certs-only`".
- Si se especifica un nombre de fichero asociado, su extensión es `.p7c` (por ejemplo, "`smime.p7c`").

Éste es un ejemplo de mensaje S/MIME con sólo certificados:

```
Date: Mon, 1 Mar 2004 11:48:05 +0100
From: usuario-1@uoc.edu
Subject: Mi certificado y el de la CA
To: usuario-2@uoc.edu
MIME-Version: 1.0
Content-Type: application/pkcs7-mime; smime-type=certs-only;
        name="smime.p7c"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7c"

MIAGCSqGSIb3DQEHAqCAMIACAQExADALBgkqhkiG9w0BBwGggDCCAUgwggEFoAMCAQICAQYw
DQYJKoZIhvcNAQEEBQAwKjELMAkGA1UEBhMCRVMxDDAKBgNVBAoTA1VPQzENMAsGA1UEAxME
Q0EtMTAeFw0wMDA5MDEwMDAwMDBaFw0xMDA5MDEwMDAwMDBaMFUxCzAJBgNVBAYTAkVTMQww
CgYDVQQKEwNVT0MxJTAjBgkqhkiG9w0BCQEWFnVzdWFyaS0xQGNhbXB1cy51b2MuZXMxETAP
BgNVBAMTCHVzdWFyaS0xMEkwDQYJKoZIhvcNAQEBBQADOAAwNQIuDDUSzmiN9Ja0UI8ss504
0E7P7y73ohWnUYhipVTZ7BZBsfxMB3RQ8u8+AqW7ZwIDAQABMA0GCSqGSIb3DQEBBAUAAy4A
AkWaw7C/zCb/VWXW8TTKW4ynMMBTz1RvS06jmfLE3QBdPVJxbEo+YxNAAQDxMIIBGzCB2aAD
AgECAgEBMA0GCSqGSIb3DQEBBAUAMCoxCzAJBgNVBAYTAkVTMQwwCgYDVQQKEwNVT0MxDTAL
BgNVBAMTBENBLTEwHhcNMDAwOTAxMDAwMDAwWhcNMTAwOTAxMDAwMDAwWjAqMQswCQYDVQQG
EwJFUzEMMAoGA1UEChMDVU9DMQ0wCwYDVQQDEwRDQS0xMEgwDQYJKoZIhvcNAQEBBQADNwAw
NAItDgNwpzTW3wWW7che7zeVoV4DbqznSQfm5hnUe3kkZOelPU4o8DJqMav2JyxjAgMBAAEw
DQYJKoZIhvcNAQEEBQADLgACXnrqZYIk3CY+641wJs99q7mIC4bPK3O75IskUrvGxs1PvZRt
yoj8zZDJtcAAADGAAAAAAAAAAAA=
```

Finalmente, existe otro tipo de mensaje S/MIME para enviar **peticiones de certificación** a una CA. Una petición de certificación es un mensaje que contiene los datos necesarios para que la CA genere un certificado, básicamente el nombre del titular y su clave pública.

A veces se utiliza como petición de certificación un certificado X.500 autofirmado por el interesado. Otras veces se utiliza una estructura de datos definida expresamente a tal efecto, especificada en la norma PKCS #10.

El tipo de mensaje S/MIME utilizado para enviar peticiones de certificación tiene las siguientes características:

- En el cuerpo del mensaje hay una estructura PKCS #10.
- El valor de la cabecera `Content-Type` es "`application/pkcs10`".
- Si se especifica un nombre de fichero asociado, su extensión es `.p10` (por ejemplo, "`smime.p10`").

2.3. PGP y OpenPGP

PGP (*Pretty Good Privacy*) es un software que proporciona funciones criptográficas y de gestión de claves, desarrollado inicialmente por Philip Zimmermann en 1990. Se puede utilizar para proteger cualquier tipo de datos, pero su uso más habitual consiste en enviar mensajes de correo electrónico cifrados y/o firmados.

Una de las características distintivas de PGP es el método que utiliza para certificar la autenticidad de las claves públicas. En lugar de recorrer a autoridades de certificación, como hace S/MIME, cada usuario puede certificar directamente las claves que está convencido que son auténticas. Y también puede tomar decisiones respecto a una clave desconocida en función de quienes sean los usuarios que hayan certificado esta clave.

Otra característica propia de PGP es la eficiencia en el intercambio de los mensajes ya que, siempre que sea posible, los datos se comprimen antes de cifrarlos y/o después de firmarlos.

Con el paso del tiempo han ido apareciendo distintas versiones de PGP, algunas de las cuales con distintas variantes como, por ejemplo, versiones "internacionales" para cumplir las restricciones de exportación de software criptográfico fuera de los Estados Unidos, o versiones que para poder ser distribuidas de forma gratuita no incluyen algoritmos sujetos a patentes en ciertos países.

- Las versiones de PGP hasta la 2.3 se consideran obsoletas: el formato de las firmas es incompatible con el de las versiones posteriores.
- Después de la versión 2.5 se introdujo otro cambio de formato, a causa de las condiciones de uso de algoritmos patentados en los Estados Unidos (en concreto, el algoritmo RSA).
- Las versiones 2.6.x y sus variantes han sido durante mucho tiempo las más difundidas. El formato de los mensajes, llamado indistintamente "versión 2" o "versión 3" (V3), está documentado en la especificación RFC 1991.
- En las versiones 4.x se introdujeron, entre otras novedades, las claves de una sola función: claves sólo para cifrar o sólo para firmar.
- Lo que se empezó a diseñar con el nombre de "PGP 3.0" finalmente dio lugar a las versiones 5.x, que utilizan un nuevo formato para los mensajes, conocido como "versión 4" (V4) o "OpenPGP" y documentado en la es-

Algoritmos en PGP 2.6.x

Los algoritmos que utilizan las versiones 2.6.x de PGP son: IDEA para el cifrado simétrico, RSA para el cifrado de clave pública, y MD5 par el *hash*.

pecificación RFC 2440.

- Las nuevas versiones (PGP 6 y posteriores) añaden mejoras a la funcionalidad, pero manteniendo la compatibilidad con las anteriores.
- También se está desarrollando en paralelo, como parte del proyecto GNU, el software GnuPG (*GNU Privacy Guard*), basado en OpenPGP y de distribución totalmente libre (no utiliza algoritmos patentados como RSA o IDEA).

Algoritmos en PGP 5.x

PGP 5.x contempla el uso de nuevos algoritmos, como Triple DES y CAST-128 para el cifrado simétrico, DSA y ElGamal para las firmas, y SHA-1 y RIPEMD-160 para el *hash*. Si trabaja solamente con IDEA, RSA y MD5, las versiones 5.x pueden generar mensajes totalmente compatibles con las versiones 2.6.x.

2.3.1. Formato de los mensajes PGP

Los datos que procesa PGP se codifican con unas estructuras de datos llamadas **paquetes PGP**. Un mensaje PGP está formado, pues, por uno o más paquetes PGP.

GnuPG

Podéis encontrar información sobre el proyecto GnuPG en `www.gnupg.org`.

Un paquete PGP es una secuencia de bytes, con una cabecera que indica de qué tipo de paquete se trata y su longitud y, a continuación, unos campos de datos que depenen del tipo de paquete.

El formato V3 define diez tipos de paquetes, mientras que el formato V4 define catorce. A continuación veremos los principales tipos de paquetes PGP.

1) Paquete de datos literales

Sirve para representar datos en claro, sin cifrar (sería el análogo del contenido `Data` en PKCS #7).

En un paquete de este tipo existe un campo que da el valor de los datos, y otro que indica si este valor se debe procesar como texto o como datos binarios. En el primer caso, las secuencias <CR><LF> que haya en el texto corresponden a finales de línea y se pueden convertir a la representación local cuando se tengan que visualizar o guardar en fichero, mientras que en el segundo caso no se tienen que modificar.

2) Paquete de datos comprimidos

En este tipo de paquete hay un campo que indica el algoritmo de compresión, y otro que contiene una secuencia de bytes comprimida. Cuando se descomprimen estos bytes, el resultado debe ser uno o más paquetes PGP.

Algoritmos de compresión

El algoritmo de compresión que utiliza PGP es el ZIP (RFC 1951), y OpenPGP utiliza también el algoritmo ZLIB (RFC 1950).

Normalmente lo que se comprime es un paquete de datos literales, opcionalmente precedido por un paquete de firma (que veremos a continuación).

3) Paquete de datos cifrados con clave simétrica

El contenido de este paquete es directamente una secuencia de bytes cifra-

dos con un algoritmo simétrico. El resultado de descifrarlos tiene que ser un o más paquetes PGP.

Típicamente lo que se cifra simétricamente son paquetes de datos en claro o paquetes de datos comprimidos.

Los paquetes de este tipo se usan para enviar un mensaje de correo cifrado con sobre digital, o bien cuando el usuario quiere simplemente cifrar un fichero. En el primer caso, es preciso adjuntar al mensaje la clave simétrica cifrada de tal forma que sólo la pueda descifrar el destinatario o destinatarios. Esto se realiza con paquetes cifrados con clave pública (el tipo de paquete PGP que veremos a continuación). En el segundo caso, la clave no se guarda en ningún sitio sino que el usuario la tiene que recordar cuando quiera descifrar el fichero. En realidad, el usuario no da directamente la clave de cifrado si no una *passphrase*, a partir de la cual se obtiene la clave simétrica aplicándole una función de *hash*.

Cifrado simétrico en PGP

PGP utiliza el modo CFB para el cifrado simétrico. En lugar de especificar aparte el vector de inicialización (VI) se usa un VI igual a cero, pero a los datos que hay que cifrar se les antepone una cadena de 10 bytes: los 8 primeros son aleatorios, y los otros 2 son redundantes (iguales al 7º y 8º) y sirven para que el destinatario compruebe si ha usado la clave correcta para descifrar.

4) Paquete de datos cifrados con clave pública

En este tipo de paquete hay un campo que sirve para identificar la clave pública utilizada, otro que indica el algoritmo de cifrado, y otro con los datos cifrados.

Habitualmente este paquete se utilizaba para cifrar una clave de sesión, con la cual se habrá generado un paquete de datos cifrados simétricamente, para enviar un mensaje con sobre digital. La clave pública utilizada en este caso es la de cada uno de los destinatarios del mensaje.

5) Paquete de firma

Un paquete de este tipo contiene campos con la siguiente información:

- Clase de firma, que puede ser:
  - Firma de datos binarios.
  - Firma de texto canónico.
  - Certificado, es decir, asociación de clave pública con nombre de usuario.
  - Revocación de clave pública.
  - Revocación de certificado.
  - Fechado (*timestamp*).
- Fecha y hora en que se creó la firma.
- Identificador de la clave con la que se ha creado.
- Algoritmos utilizados para el *hash* y el cifrado asimétrico.
- La firma, que se obtiene aplicando los algoritmos especificados en los datos que hay que firmar, concatenados con los campos autenticados. En el formato V3 estos campos autenticados son la clase de firma y la fecha de creación. En el formato V4 se puede especificar que otros campos se autentican.

- Otros campos añadidos en el formato V4, entre los cuales hay:

– Fecha de caducidad de la firma y de la clave.

– Comentarios del autor de la firma.

– Motivo de revocación (en el caso de las firmas de revocación).

El modo de saber a qué datos corresponde una firma depende del contexto. Si es la firma de un mensaje de correo, en el mismo mensaje tiene que haber el paquete con los datos (normalmente datos literales) despues del de firma. Si es un certificado o una revocación, la firma tiene que ir después de los paquetes de clave pública y de nombre de usuario correspondientes (a continuación veremos estos dos tipos de paquetes PGP).

También es posible firmar el contenido de un fichero y dejar el paquete de firma en un fichero separado. Esto se suele hacer cuando se distribuyen programas (como, por ejemplo, el propio PGP) para garantizar que una versión es auténtica y no constituye un "caballo de Troya". En este caso la asociación entre datos y firma se puede establecer mediante los nombres de los ficheros: por ejemplo, el fichero con la firma se puede llamar como el original, pero con la extensión `.sig`.

6) Paquete de clave pública

Este tipo de paquete contiene la siguiente información relativa a una clave pública:

- La fecha de creación de la clave.
- El algoritmo al que corresponde la clave.
- Los valores de los componentes de la clave. Si el algoritmo es RSA, estos valores son el módulo n y el exponente público e.

La clave pública de un usuario se utiliza para enviarle datos cifrados o para verificar las firmas que genere. Pero los paquetes correspondientes (datos cifrados con clave pública o firma, respectivamente) no contienen el valor de la clave pública utilizada, sino solamente su **identificador de clave**.

El identificador de una clave pública es un número de ocho bytes que se puede utilizar para buscar el valor de la clave en una base de datos.

Identificadores de claves PGP repetidos

Las implementaciones no tienen que suponer que los identificadores de clave sean únicos: podría haber dos claves PGP distintas con el mismo identificador. Sin embargo, la probabilidad de que pase esto es muy baja (a no ser que se haga deliberadamente), porque puede haber 2^{64} ($> 10^{19}$) identificadores distintos.

Si, por ejemplo, una firma está generada con una clave que tiene un determinado identificador, y resulta que hay dos claves con este identificador, es preciso verificarla con cada una de las claves para comprobar si es válida.

7) Paquete de nombre de usuario

Otros campos del paquete de clave pública

En el formato V3 hay un campo que indica el período de validez de la clave, pero todas las implementaciones lo ponen a 0, que indica que las claves son válidas para siempre. En el formato V4 esta información se especifica en los paquetes de firma. Por otro lado, en V4 están previstos componentes de la clave pública para otros algoritmos además del RSA.

Valor del identificador de clave

En las claves V3 (que son siempre claves RSA) el identificador es igual a los 8 bytes de menos peso del módulo público n. En las claves V4 es igual a los 8 bytes de menos peso de la huella (más adelante veremos que son las huellas PGP).

El contenido de un paquete de este tipo es simplemente una cadena de caracteres, que se utiliza para identificar el propietario de una clave pública. Por tanto, tiene la misma función que el DN del sujeto en los certificados X.509, pero sin ninguna estructura predefinida.

Aunque su formato es libre, se suele seguir el convenio de identificar a los usuarios con direcciones de correo electrónico RFC 822 como, por ejemplo:

```
Philip R. Zimmermann <prz@acm.org>
```

8) Paquete de clave privada

Este tipo de paquete sirve para guardar los componentes de la clave privada de un usuario. Nunca existe ningún motivo para enviarlo a otro usuario y, por lo tanto, el formato exacto del paquete puede depender de la implementación.

Para asegurar la confidencialidad, en el fichero donde se guarde este paquete los componentes secretos de la clave deberían estar cifrados, normalmente con una clave simétrica derivada de una *passphrase*. De este modo, cada vez que el usuario quiera descifrar o firmar un mensaje con su clave privada, deberá indicar esta *passphrase* para poder obtener los valores necesarios.

Un usuario puede tener varias claves, asociadas al mismo o a distintos nombres. En el fichero en el que hayan los paquetes de clave privada, a continuación de cada uno habrá el paquete o paquetes de nombre de usuario correspondientes.

9) Paquete de nivel de confianza en una clave

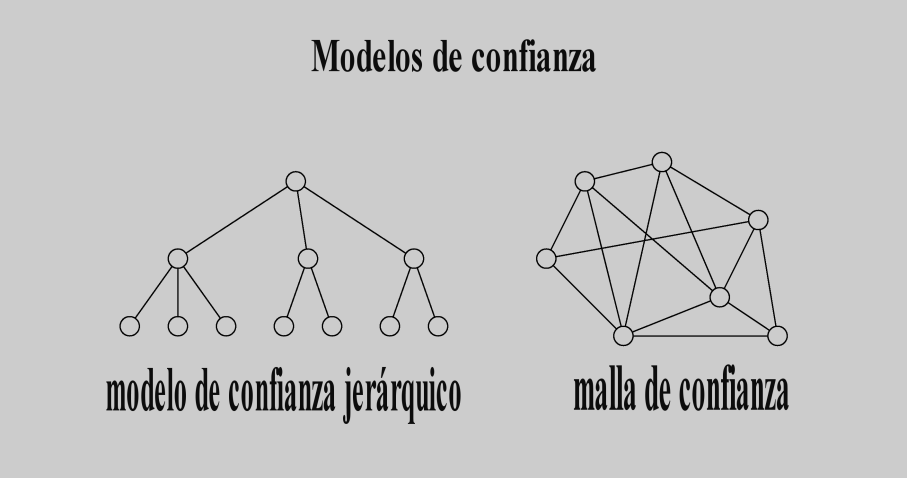
Este tipo de paquete tampoco se envía nunca sino que solamente se guarda en el almacén de claves propio de cada usuario, ya que únicamente tiene significado para quien lo ha generado. Sirve para indicar el grado de fiabilidad de una clave certificadora, es decir, se utiliza para asociar otras claves con nombres de usuarios.

En el formato V3 hay otro tipo de paquete que sirve para incluir comentarios, pero se ha suprimido en el formato V4 porque no lo utilizaba ninguna implementación.

El formato V4 también utiliza otros cinco tipos de paquetes, entre ellos los relacionados con las llamadas **subclaves**. Una clave puede tener asociadas una o más subclaves: normalmente, la clave principal se utiliza para firmar y las subclaves, para cifrar.

2.3.2. Distribución de claves PGP

Como hemos comentado al inicio de este apartado, la certificación de claves en PGP no sigue un modelo jerárquico, como el de las autoridades de certificación X.509, sino un modelo descentralizado de confianza mutua, a veces llamado **malla de confianza** (*web of trust*).



Cuando un usuario genera su par de claves PGP (pública y privada), a la clave pública le tiene que asociar un nombre de usuario y, a continuación, tiene que autocertificar esta asociación, es decir, firmar con su clave privada la concatenación de la clave pública y el nombre. El paquete de la clave pública, el del nombre de usuario y el de la firma forman un **bloque de clave**.

Lectura complementaria

En el documento `www.cl.cam.ac.uk/Research/Security/Trust-Register/gtr1998introduction.pdf` podéis encontrar una comparación de los distintos modelos de confianza.

Opcionalmente, el usuario puede asociar más de un nombre a la clave (por ejemplo, si tiene diversas direcciones de correo electrónico) y, entonces, el bloque estará formado por la clave pública y tantos pares nombre–firma como sea preciso.

Autocertificación de claves

Cuando se genera un nuevo par de claves, las versiones modernas de PGP firman automáticamente la clave pública y el nombre de usuario con la propia clave privada. En las versiones más antiguas no lo hacían de esta forma y se tenía que generar el autocertificado manualmente.

En este caso el usuario puede enviar este bloque a otros con los que tenga que mantener correspondencia. Cada receptor, si está convencido de que la clave efectivamente corresponde al usuario originador, la certificará añadiendo otro paquete de firma al bloque (o uno para cada nombre, si hay más de uno y también los considera auténticos). De este modo, cada usuario dispondrá de un almacén de claves públicas certificadas (*keyring*) que podrá utilizar para cifrar mensajes y verificar firmas de sus corresponsales.

Claves de revocación

En OpenPGP también está prevista la existencia de claves autorizadas a revocar otras claves.

Además de paquetes de claves públicas, nombres de usuario y firmas de certificación, en un *keyring* también pueden haber firmas de revocación para invalidar una clave o para anular un certificado. La revocación debe estar firmada

por la propia clave (la que se quiere invalidar o la que generó el certificado que se quiere anular) y, una vez emitida, es irrevocable.

Otra posibilidad para realizar la distribución es a través de un **servidor de claves PGP**, que gestiona un almacén global de claves públicas con sus certificados (y revocaciones, si es el caso). Varios de estos servidores están sincronizados entre si, de modo que las actualizaciones del almacén que se realicen en uno de ellos se propagan automáticamente a todos los demás.

Servidores de claves PGP

En la dirección `www.rediris.es/cert/servicios/keyserver/` hay uno de estos servidores de claves PGP.

Cualquier usuario puede enviar a un servidor PGP los certificados de su clave, o de las claves de otros usuarios, para que los añada al *Keyring* global. En este caso se pueden realizar consultas a los servidores para obtener la clave y los certificados asociados a un determinado nombre de usuario (o a los nombres que contengan una determinada subcadena, si no se conoce su valor exacto).

***keyring* global**

El tamaño del "*keyring* global" almacenado en los servidores de claves PGP es actualmente de más de 2 Gbytes. En la dirección `bcn.boulder.co.us/~neal/pgpstat/` podéis encontrar un interesante estudio estadístico sobre las claves de este almacén.

Es importante señalar que los servidores PGP no son autoridades de certificación: cualquier clave pública que se les envíe será añadida al almacén sin realizar ninguna verificación respecto a la identidad del propietario.

Es responsabilidad de cada usuario que quiera utilizar una clave de un servidor PGP asegurarse de la autenticidad de dicha clave. Para ello, se puede tener en cuenta que otros usuarios han certificado esta clave.

Claves PGP falsas

Son famosas, por ejemplo, las claves con nombre `<president@whitehouse.gov>` que han sido mandadas a los servidores PGP por personas que no tienen ninguna relación con esta dirección de correo.

2.3.3. El proceso de certificación PGP

Para facilitar el intercambio y la certificación de claves, PGP asigna a cada clave pública una **huella** (*fingerprint*), que es simplemente un *hash* del valor de la clave.

Esta huella se utiliza para que un usuario pueda comprobar que la clave que ha recibido de otro, o de un servidor de claves, sea efectivamente la que quería recibir y no una falsificación. El identificador de clave no es suficiente para este fin, ya que es posible para un impostor construir una clave pública de la cual conozca la clave privada y que tenga el mismo identificador que otra clave pública. En cambio, construir una clave con la misma huella que otra es prácticamente imposible.

Cálculo de la huella

Para calcular la huella de las claves V3, la función *hash* que se aplica es MD5 (16 bytes), mientras que para las claves V4 es SHA-1 (20 bytes).

El uso de la huella facilita la comprobación de la autenticidad de la clave. La alternativa sería comprobar todos los bytes uno por uno, lo cual puede ser incómodo y pesado teniendo en cuenta que actualmente se suelen utilizar claves de 1.024 bits (256 dígitos hexadecimales), o incluso de 2.048 bits.

Supongamos, por ejemplo, que un usuario *A* necesita certificar la clave de otro usuario *B* porque tiene que intercambiar con él mensajes seguros. El usuario *A* puede pedir a *B* que le mande su clave pública por correo electrónico tradicional, o bien obtenerla de un servidor de claves. Entonces *A* tiene que comprobar que nadie ha manipulado el mensaje de respuesta, obteniendo de *B* la siguiente información por un canal aparte (no por correo electrónico):

- El identificador de la clave pública de *B*.
- La huella de esta clave.
- El algoritmo y el número de bits de la clave (el número de bits puede ser necesario para evitar colisiones en la huella).

Métodos para comunicar la información sobre una clave pública

Un canal seguro para enviar la información anterior puede consistir, por ejemplo, en la comunicación directa "cara a cara" en una conversación telefónica. También hay quien se imprime el valor de la huella PGP en su tarjeta, o quien lo hace accesible vía `finger` o HTTP (pero este método no es tan seguro).

Otra posibilidad es organizar una *key-signing party* o "reunión de firma de claves".

Aunque PGP trabaja internamente con identificadores de clave de 8 bytes, cuando tiene que mostrar sus valores al usuario solamente muestra los 4 bytes de menos peso. Por ejemplo, si una clave tiene por identificador el valor hexadecimal 657984B8C7A966DD, el usuario solamente ve el valor C7A966DD. Esto da más comodidad sin aumentar significativamente, en la práctica, las posibilidades de repetición.

Identificadores de cuatro bytes repetidos

En el caso límite, más de dos tercios de los habitantes de la Tierra podrían tener clave PGP sin que hubiera ningún identificador de 4 bytes repetido.

Éste sería, pues, un ejemplo de toda la información necesaria para certificar una clave:

```
bits /keyID    User ID
1024R/C7A966DD Philip R. Zimmermann <prz@acm.org>
Key fingerprint =  9E 94 45 13 39 83 5F 70  7B E7 D8 ED C4 BE 5A A6
```

Cuando el usuario *A* ha comprobado que los valores que le ha comunicado *B* coinciden con los calculados a partir de la clave pública recibida electrónicamente, ya puede certificar que esta clave corresponde al nombre o nombres de usuario que identifican al usuario *B*.

2.3.4. Integración de PGP con el correo electrónico

Como hemos visto anteriormente, un mensaje PGP consta de una secuencia de paquetes PGP. Pueden existir distintas combinaciones:

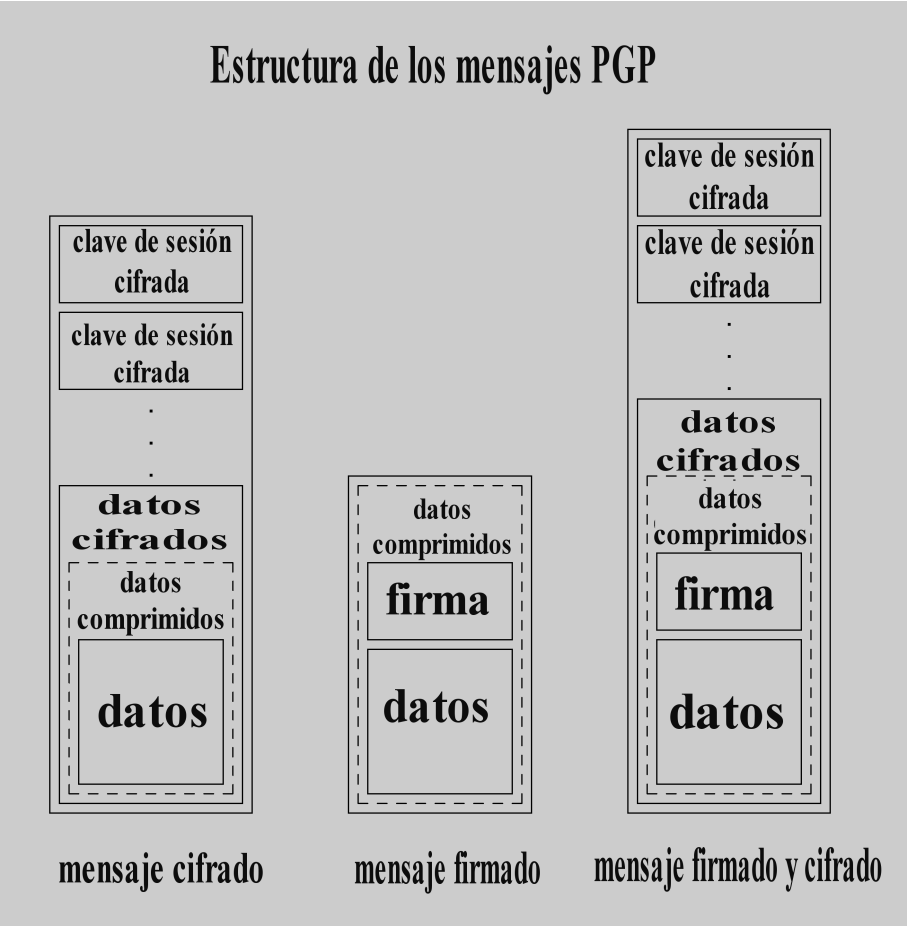
- Si es un mensaje cifrado con sobre digital, primero hay tantos paquetes

como destinatarios, cada uno con la clave de sesión cifrada con la clave pública del destinatario correspondiente. A continuación aparece el cuerpo del mensaje, posiblemente comprimido, en un paquete cifrado simétricamente con la clave de sesión.

- Si es un mensaje firmado, primero encontramos el paquete de firma, y después el cuerpo del mensaje en un paquete de datos literales. Opcionalmente, estos dos paquetes se pueden incluir dentro de un paquete comprimido.
- Si es un mensaje firmado y cifrado, la estructura es como la de los mensajes cifrados, con la excepción de que cuando se descifra el paquete de datos cifrados el resultado es un mensaje firmado, es decir, un paquete de firma seguido de un paquete de datos literales, o bien un paquete comprimido que cuando se descomprime da los dos paquetes anteriores.

Cifrado de la clave de sesión en OpenPGP

OpenPGP también contempla la posibilidad de cifrar la clave de sesión con claves simétricas, usando un nuevo tipo de paquete.



Los mensajes construidos de esta manera contendrán datos binarios arbitrarios. Para enviarlos a través de los sistemas de correo electrónico tradicionales se pueden utilizar dos técnicas: encapsulación RFC 934 y MIME (con el método llamado PGP/MIME). Con la encapsulación RFC 934 se pueden representar mensajes cifrados y/o firmados, mensajes firmados en claro, o bloques de claves públicas.

La técnica de encapsulación RFC 934

La especificación RFC 934 define una técnica para combinar uno o más mensajes en un único cuerpo. Ésta es la técnica que utiliza MIME para juntar distintas partes en un mensaje multiparte.

La encapsulación RFC 934 consiste en concatenar los mensajes que hay que combinar, poniéndolos simplemente uno a continuación de otro, y utilizando delimitadores de encapsulación para indicar dónde empieza cada uno y dónde termina el último.

El texto que haya antes del primer delimitador se considera como un "prólogo" y el que hay después del último delimitador se considera como un "epílogo", pero ninguno de los dos forma parte del mensaje encapsulado.

Como delimitadores de encapsulación se utilizan líneas que empiezan con un guión seguido de otro carácter que no sea espacio. Si dentro de los mensajes que hay que encapsular hay alguna línea que empiece por un guión, simplemente se le antepone un guión y un espacio. En el momento de desencapsular, a las líneas que empiecen por guión y espacio se les suprimen estos dos caracteres. Las que empiecen por guión y otro carácter serán consideradas como delimitadores. Esto permite la encapsulación recursiva de mensajes compuestos dentro de otro mensaje compuesto.

1) Mensajes PGP cifrados y/o firmados

Éste es un ejemplo de mensaje PGP, codificado con el método de encapsulación RFC 934.

```
Date: Mon, 1 Mar 2004 11:35:40 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 4
To: usuario-2@uoc.edu

-----BEGIN PGP MESSAGE-----
Version: 2.6.3y

iQBDAwUBObNQzFDy7z4CpbtnAQF9aQFrBtyRK8bdaPFlht7KeFzO/N0lJTcnYhbS
TvlZsTwr6+iQJqHP5nKnYr0W/Q9mo60AI3QAAAAAAEV4ZW1wbGUgZGUgbWlzc2F0
Z2Ugc2lnbmF0Lg0K
=8gbQ
-----END PGP MESSAGE-----
```

Como podemos ver en el ejemplo, la estructura del mensaje es la siguiente:

- Como delimitador inicial de encapsulación se utiliza la cadena "`BEGIN PGP MESSAGE`" entre dos secuencias de cinco guiones, y el delimitador final es igual pero cambiando "`BEGIN`" por "`END`".
- . Después del delimitador inicial puede haber distintas cabeceras, con campos como `Version` para indicar qué versión de PGP ha generado el mensaje, `Comment` para introducir comentarios del usuario, o `Charset`, para especificar el juego de caracteres utilizado en el texto del mensaje.
- Después de las cabeceras aparece una línea en blanco, y el paquete o paquetes PGP que forman el mensaje codificado en base 64.
- Inmediatamente después de los paquetes PGP y antes del delimitador de final, aparece una línea de cinco caracteres: el primero es el signo '"=" y los otros cuatro son la codificación en base 64 de un CRC de 24 bits de todos los bytes de los paquetes. Este CRC sirve para comprobar que

Juego de caracteres en OpenPGP

En OpenPGP, el juego de caracteres por defecto es Unicode (ISO/IEC 10646), codificado con UTF-8 (RFC 2279)

no se hayan producido modificaciones en el mensaje que hayan podido afectar a la descodificación.

En la terminología PGP, la secuencia de líneas desde el delimitador de encapsulación de inicio hasta el del final se llama **armadura ASCII** del mensaje.

2) Mensajes PGP firmados en claro

Igual que S/MIME, PGP también define un formato para enviar mensajes firmados en claro, que permite leer el contenido a los usuarios que no disponen de PGP. Éste es un ejemplo:

```
Date: Mon, 1 Mar 2004 11:35:55 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 5
To: usuario-2@uoc.edu

-----BEGIN PGP SIGNED MESSAGE-----
Hash: MD5

Ejemplo de mensaje firmado.

-----BEGIN PGP SIGNATURE-----
Version: 2.6.3y

iQBDAwUBObNQzFDy7z4CpbtnAQF7aQFrBtyRK8bdaPFlht7KeFzO/N0lJTcnYhbS
TvlZsTwr6+iQJqHP5nKnYr0W/Q9mow==
=5TnX
-----END PGP SIGNATURE-----
```

En este caso aparecen dos submensajes encapsulados, con la siguiente estructura:

- El delimitador de inicio de la primera parte es la cadena "`BEGIN PGP SIGNED MESSAGE`", con una secuencia de cinco guiones delante y detrás.
- En el primer submensaje aparecen cero o más cabeceras `Hash`, que indican el algoritmo (o algoritmos) de *hash* utilizados para calcular la firma (o firmas), seguidos de una línea en blanco y del cuerpo del mensaje. La especificación del algoritmo al inicio permite procesar el mensaje en un solo paso. En ausencia de este campo, se entiende por defecto que la función de *hash* utilizada es MD5.
- Después del primer submensaje aparece la armadura ASCII de uno o más paquetes de firma, con un delimitador de inicio formado por la cadena "`BEGIN PGP SIGNATURE`", también con cinco guiones delante y detrás, y con un delimitador de final igual, aunque cambiando "`BEGIN`" por "`END`".

  Las firmas se calculan a partir del cuerpo del mensaje en forma canónica, es decir, representando los finales de línea con <CR><LF>. Además, PGP siempre elimina los espacios en blanco y los tabuladores que haya antes de un final de línea en el momento de obtener las firmas.

Líneas que empiezan con guión

Si en el primer submensaje hay líneas que empiezan por guión, hay que añadir la secuencia "`- `" de acuerdo con RFC 934. La firma, sin embargo, se obtiene de las líneas originales.

3) Mensajes de bloques de claves públicas

Hay otro formato de armadura PGP que sirve para enviar bloques de claves públicas y certificados. El delimitador de inicio consta de la cadena "`BEGIN PGP PUBLIC KEY BLOCK`" rodeada de dos secuencias de cinco guiones, y el de final es igual, aunque cambiando "`BEGIN`" por "`END`". Éste es un ejemplo:

```
Date: Mon, 1 Mar 2004 11:38:20 +0100
From: usuario-1@uoc.edu
Subject: Mi clave PGP
To: usuario-2@uoc.edu

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: 2.6.3y

mQA9AzmvN9AAAAEBbAw1Es5ojfSWtFCPLLOdONBOz+8u96IVp1GIYqVU2ewWQbH8
TAd0UPLvPgKlu2cAEQEAAbQhVXN1YXJpIDEgPHVzdWFyaS0xQGNhbXB1cy51b2Mu
ZXM+iQBCAwUQOa830FDy7z4CpbtnAQFdoQFox7LHdl8wdIA69f4REn14bVYxBaxw
4Y35PJwRWqI2c+8T75vqUdBhiydsZ2Fo
=Ninr
-----END PGP PUBLIC KEY BLOCK-----
```

Cualquiera de los tipos de armadura que hemos visto se puede utilizar para intercambiar información PGP con otros medios de transferencia además del correo electrónico: FTP, HTTP, etc.

4) PGP/MIME

Para incorporar PGP a MIME, inicialmente se definió el tipo de contenido MIME `application/pgp`. Más tarde, sin embargo, este tipo de contenido se abandonó en favor del método RFC 1847, que es el mismo que utiliza S/MIME para los mensajes firmados en claro. Además del tipo de contenido `multipart/signed` correspondiente a los mensajes firmados, RFC 1847 también define el tipo `multipart/encrypted` correspondiente a los mensajes cifrados.

La técnica para incluir mensajes PGP en mensajes MIME RFC 1847 se denomina PGP/MIME, y está especificada en el RFC 2015. PGP/MIME define tres tipos de contenido para las partes MIME que representen mensajes PGP: `application/pgp-encrypted`, `application/pgp-signature` y `application/pgp-keys`.

Sin embargo, actualmente es mucho más habitual el uso de las armaduras ASCII para encapsular mensajes PGP que la técnica PGP/MIME.

Resumen

En este módulo didáctico hemos presentado dos aplicaciones que utilizan los mecanismos de protección que hemos visto en el módulo anterior. La primera de ellas es el *Secure Shell* **(SSH)**, que permite establecer una conexión cifrada con un servidor. SSH define su propio protocolo para autenticar el servidor y el usuario que se quiere conectar, y para determinar las claves para el cifrado simétrico y para los códigos MAC, de forma parecida a como lo hace SSL/TLS.

Una vez establecida la comunicación segura, el protocolo SSH permite usar otras funciones, como el envío de datos protegidos por un canal, la **redirección de puertos TCP** desde el cliente o desde el servidor, etc.

La otra aplicación que hemos visto en este módulo es el **correo electrónico seguro**. Dado que en esta aplicación interesa proteger los mensajes enviados más que la comunicación en sí, se definen mecanismos para introducir la confidencialidad y la autenticación en el cuerpo de los mensajes de correo. Esto permite aprovechar la infraestructura de agentes de correo existentes, manteniendo la compatibilidad con los sistemas de correo tradicionales.

Para la confidencialidad normalmente se utiliza la técnica del **sobre digital**, que consiste en cifrar el mensaje con una clave de sesión simétrica, y añadirle esta clave de sesión cifrada con la clave pública de cada destinatario. De este modo se puede enviar un mismo mensaje a múltiples destinatarios, como se hace con el correo electrónico normal. Para la autenticación se utilizan las **firmas digitales**, que además proporcionan el servicio de no repudio.

Uno de los principales sistemas de correo electrónico seguro actualmente en uso es **S/MIME**. Este sistema incorpora estructuras de datos **PKCS #7** a los mensajes de correo utilizando la tecnología MIME. La norma PKCS #7 especifica cómo se deben representar mensajes cifrados con sobre digital y/o firmados, aplicando criptografía de clave pública y sobre una infraestructura de certificados X.509.

Otro sistema para el intercambio de datos protegidos y, en particular, mensajes de correo electrónico, es **PGP**. Este sistema utiliza un formato propio, publicado en especificaciones como la antigua “PGP versión 3” o la más moderna “OpenPGP”, para representar los datos cifrados y/o firmados. Para las claves públicas no utiliza una infraestructura de certificados X.509, sino que la certificación es descentralizada: cada usuario puede certificar las claves públicas que crea auténticas, de modo que las relaciones entre claves públicas de usuarios que confían en otros forman una “malla de confianza”.

Actividades

3-1 Una implementación libre del protocolo SSH bastante conocida es la del proyecto OpenSSH. Visitad su página web (`www.openssh.com`) y comprobad qué protocolos y qué algoritmos criptográficos soporta la última versión.

3-2 Visitad la página web del proyecto GnuPG (`www.gnupg.org`) y comprobad qué algoritmos criptográficos soporta la última versión.

3-3 Acceded a un servidor de claves públicas PGP (por ejemplo, `www.rediris.es/cert/servicios/keyserver/`). ¿Que información se tiene que dar para encontrar una clave PGP? ¿Qué tipo de información puede devolver?

Buscad las claves públicas asociadas al nombre "`Philip R. Zimmermann`". ¿Existe alguna razón para pensar que alguna de ellas pueda ser falsa?

Ejercicios de autoevaluación

3-1 Una organización quiere ofrecer un servicio web seguro, con autenticación de servidor y confidencialidad de los datos, pero no dispone de un servidor HTTPS sino de software cliente y servidor del protocolo SSH, que se puede instalar en cualquier ordenador que tenga que hacer uso de este servicio. ¿Cómo se puede configurar el software SSH para ofrecer el servicio deseado?

3-2 En los sistemas de correo electrónico seguro como S/MIME o PGP:

a) ¿Cómo se aplica la protección de confidencialidad a un mismo mensaje de forma que pueda ser leído por cinco destinatarios diferentes, y solamente por estos cinco?

b) El remitente puede usar un cliente de correo que guarde copia de cada mensaje enviado. Si el mensaje se ha enviado cifrado, ¿cómo puede el remitente consultar más adelante el contenido original del mensaje?

c) Si el remitente también quiere firmar el mensaje, ¿es necesario que lo firme antes de cifrarlo o después? ¿Por qué?

3-3 Observad los ejemplos de mensajes firmados en claro S/MIME (página 37) y PGP (página 55). ¿Sabríais explicar por qué la firma del primero es más larga que la del segundo?

3-4 Si un atacante intenta un ataque de repetición contra el correo S/MIME, reenviando, por ejemplo, un mensaje como éste con el campo "`Date`" de la cabecera canviado:

```
Date: Mon, 14 Jun 2004 11:45:20 +0100
From: profesor
Subject: Entrega de la práctica aplazada
To: estudiantes
MIME-Version: 1.0
Content-Type: multipart/signed; boundary="20040614094520";
        protocol="application/pkcs7-signature"; micalg=md5

--20040614094520
Content-Type: text/plain

La entrega de la práctica que se tenía que realizar este viernes
queda aplazada hasta el viernes de la próxima semana.

El profesor.

--20040614094520
... (la firma S/MIME del mensaje) ...
--20040614094520--
```

¿habría manera de detectar el ataque?

3-5 Si un atacante intenta un ataque de repetición contra el correo PGP, reenviando por ejemplo un mensaje como este con el campo "`Date`" de la cabecera cambiado:

```
Date: Mon, 14 Jun 2004 11:45:30 +0100
From: profesor
Subject: Entrega de la práctica aplazada
To: estudiants

-----BEGIN PGP SIGNED MESSAGE-----
Hash: MD5

La entrega de la práctica que se tenía que realizar este viernes
queda aplazada hasta el viernes de la próxima semana.

El profesor.

-----BEGIN PGP SIGNATURE-----
... (la firma PGP del mensaje) ...
-----END PGP SIGNATURE-----
```

¿habría manera de detectar el ataque?

3-6 La firma de un mensaje PGP firmado se calcula mediante el algoritmo de *hash* MD5, de 128 bits de salida. El mensaje firmado incluye los primeros 16 bits de este *hash* en claro, para comprobar que los datos sobre los cuales se quiere verificar la firma son correctos.

a)¿Hasta qué punto puede comprometer esto la seguridad del algoritmo de *hash*?

b)¿Hasta qué punto ayudan estos bits a comprobar que los datos son correctos?

Solucionario

3-1 Se puede hacer uso de la funcionalidad de redirección de puertos TCP que proporciona el protocolo SSH. En el ordenador donde haya el servidor web, que normalmente admitirá conexiones por el puerto 80, se instala también el servidor SSH, configurado de forma que permita la autenticación nula (si no es necesaria la autenticación de cliente). Entonces, en cada ordenador que se tenga que conectar al servidor, se instala el cliente SSH configurándolo de modo que las conexiones que lleguen a un puerto *P* local sean redirigidas a conexiones realizadas desde el servidor al puerto 80 del propio servidor.

Una vez realizado esto, para acceder al servidor web des de los clientes sólo es necesario dar URLs de la forma "`http://localhost:`*P*`/...`" al navegador, y automáticamente se establecerán las conexiones con el servidor web mediante un canal SSH.

3-2

a) Con la técnica del sobre digital, es decir, cifrando el mensaje con una clave de sesión simétrica, y añadiendo al mensaje cifrado el resultado de cifrar la clave de sesión con la clave pública de cada uno de los cinco destinatarios.

b) Haría falta añadir una sexta copia de la clave de sesión, cifrada con la clave pública del remitente.

c) Dependiendo del uso que se quiera hacer del mensaje en recepción, puede ser más interesante cifrar antes de firmar, firmar antes de cifrar, o puede ser indiferente.

3-3 En el mensaje S/MIME hay una estructura PKCS #7 de tipo `SignedData`, que normalmente incluirá al menos el certificado X.509 del firmante (y posiblemente también el de la CA emisora, el de la CA de nivel superior, etc.). En el mensaje PGP sólo hay un identificador de la clave pública del firmante: el valor entero de la clave pública se debe obtener por otros medios (*keyring* local del receptor, servidor de claves públicas, etc.).

3-4 Se puede detectar la repetición porque la firma se representa mediante una estructura PKCS #7 `SignedData`, que puede incluir un atributo que indica cuándo se ha generado la firma: el atributo `signingTime` (aunque este atributo no es obligatorio en PKCS #7 ni CMS, se supone que las implementaciones S/MIME lo deberían incluir en los mensajes firmados).

3-5 Se puede detectar la repetición porque el receptor puede saber cuándo se ha generado la firma: esta información se encuentra en uno de los campos del paquete de firma (en el momento de verificar la firma, las implementaciones PGP normalmente muestran la fecha en la que se creó).

3-6

a) Si el mensaje sólo está firmado no existe ningún compromiso, porque cualquiera puede calcular el *hash* de los datos firmados (si está firmado y cifrado, primero se construye el mensaje firmado y después es cifra el resultado).

b) La probabilidad de que en un mensaje incorrecto coincidan los 16 primeros bits del *hash* es 2^{-16}. Por lo tanto, estos 16 bits dan una confianza razonable de que el mensaje ha llegado correctamente.

Glosario

Agente de autenticación SSH: aplicación que, a través de conexiones SSH con otras aplicaciones, permite a estas otras aplicaciones realizar la autenticación de cliente del protocolo SSH, mediante las claves privadas correspondientes, y sin que estas claves tengan que salir del sistema en el que se está ejecutando el agente.

Armadura ASCII: representación de un mensaje PGP apta para ser incluida en el cuerpo de un mensaje de correo electrónico, sin peligro de que los agentes de correo la modifiquen haciendo irrecuperable el mensaje PGP.

***Attached* (datos firmados):** ver *Firma con datos firmados incluidos*.

Base 64: codificación que permite representar datos binarios arbitrarios como líneas de caracteres ASCII, utilizando un carácter por cada 6 bits.

Bloque de clave pública PGP: conjunto formado por una clave pública PGP, el nombre o nombres de usuario asociados y los certificados que confirman que la relación entre la clave y cada nombre es auténtica.

Canal SSH: cada uno de los distintos flujos de información que se pueden transmitir a través de una conexión SSH (un canal puede ser una sesión interactiva, una conexión a un servidor de ventanas X, una conexión TCP redirigida o una conexión a un agente de autenticación).

CMS: ver *Cryptographic Message Standard*.

Código de redundancia cíclica (CRC): valor calculado a partir de una secuencia de bits, para confirmar (dentro de un margen de probabilidad) que no se ha producido un error de transmisión cuando esta secuencia llega al destinatario.

Codificación canónica: representación de los mensajes de correo electrónico que se utiliza para su transferencia, con el objetivo de que todos los sistemas puedan convertir esta forma canónica, si es necesario, a la representación local que utilice cada uno.

Codificación de transferencia: representación del contenido de los mensajes de correo electrónico que puede ser necesaria por compatibilidad con todos los posibles agentes de correo que han de procesar los mensajes (por ejemplo, la codificación base 64).

CRC: ver *Código de redundancia cíclica*.

***Cryptographic Message Standard* (CMS):** versión del formato PKCS #7 estandarizada por el IETF (*Internet Engineering Task Force*).

***Detached* (datos firmados):** ver *Firma con datos firmados no incluidos*.

Encapsulación RFC 934: técnica para combinar varios mensajes de correo electrónico en un único cuerpo de mensaje RFC 822.

***Fingerprint*:** ver *Huella*.

Firma con datos firmados incluidos (*attached*): estructura de datos que representa una firma y que incluye los datos firmados.

Firma con datos firmados no incluidos (*detached*): estructura de datos que representa una firma pero que no incluye los datos firmados, que se encuentran en algún otro lugar (por ejemplo, en otra parte del mensaje).

Firma en claro: firma (con datos no incluidos) que se añade como segunda parte de un mensaje firmado, cuya primera parte contiene los datos firmados, y que permite leer el mensaje aunque no se disponga del software necesario para verificar la firma.

HMAC: técnica para calcular códigos de autenticación de mensaje (MAC) basada en funciones *hash*.

Huella (*fingerprint*): valor resumido de una clave pública PGP, obtenido con una función *hash*, que se utiliza en lugar de la clave entera cuando se tiene que comparar con un valor

supuestamente auténtico.

Identificador de clave PGP: número que sirve para identificar una clave pública dentro de un paquete PGP, para no tener que incluir el valor entero de la clave, y que internamente se representa con 8 bytes, aunque al usuario normalmente se le muestran solamente los 4 últimos.

***Keyring* PGP:** base de datos que contiene un conjunto de claves PGP.

Malla de confianza: modelo de confianza mutua utilizado en sistemas como PGP, donde las claves públicas se pueden autenticar mediante certificados generados por cualquier usuario, en lugar de usar una estructura jerárquica de autoridades de certificación.

MIME: Ver *Multipurpose Internet Mail Extensions*.

Mensaje MIME multiparte: mensaje MIME cuyo cuerpo está estructurado en distintas partes, cada una con su contenido, que puede ser texto, gráficos, datos, etc. u otro mensaje MIME (que, a su vez, puede ser un mensaje multiparte).

***Multipurpose Internet Mail Extensions* (MIME):** estándar para la inclusión de otro tipo de información, distinto de simples líneas de texto, en el cuerpo de los mensajes de correo electrónico, de forma compatible con el estándar RFC 822.

OpenPGP: versión del formato de los mensajes PGP estandarizada por el IETF (*Internet Engineering Task Force*).

Paquete PGP: estructura de datos utilizada para representar los distintos tipos de información que hay en un mensaje protegido con PGP.

PEM: Ver *Privacy Enhanced Mail*.

PGP: Ver *Pretty Good Privacy*.

PKCS #7: Ver *Public Key Cryptographic Standard #7*.

PKCS #10: Ver *Public Key Cryptographic Standard #10*.

***Pretty Good Privacy* (PGP):** aplicación utilizada para proteger datos, con confidencialidad (sobre digital) y/o autenticidad (firma digital), que utiliza claves públicas autenticadas según un esquema descentralizado, y que se utiliza normalmente para proteger el correo electrónico.

***Privacy Enhanced Mail* (PEM):** uno de los primeros sistemas de correo electrónico seguro que se desarrollaron, compatible directamente con el formato RFC 822.

***Public Key Cryptographic Standard #7*:** norma para representar mensajes protegidos criptográficamente (normalmente con sobre digital y/o firma digital), a partir de claves públicas autenticadas con certificados X.509.

***Public Key Cryptographic Standard #10*:** estándar para representar peticiones de certificación, que se envían a una CA para que ésta genere un certificado a partir de los datos de la petición.

***Remote Shell*:** aplicación que se incorporó al sistema Unix BSD (con el nombre `rsh`), y que actualmente está disponible en casi todas las versiones de Unix, que permite a los usuarios de un sistema ejecutar comandos en otro sistema remoto.

RFC 822: estándar para la representación de los mensajes de correo electrónico, estructurados en un conjunto de líneas de cabecera, el cuerpo del mensaje, y una línea en blanco que separa las cabeceras del cuerpo.

***Secure MIME* (S/MIME):** sistema de correo electrónico seguro que utiliza MIME para incluir datos PKCS #7 o CMS en los mensajes, y por lo tanto proporciona confidencialidad (sobre digital) y/o autenticidad (firma digital) a partir de claves públicas autenticadas con certificados X.509.

***Secure Shell*:** aplicación que proporciona un servicio análogo al del programa *Remote Shell* de los sistemas Unix, pero con la comunicación protegida mediante autenticación y cifrado, y con funcionalidades añadidas, como la redirección de puertos TCP a través de conexiones seguras, etc. También es el nombre que recibe el protocolo utilizado por esta aplicación para la comunicación segura.

***Simple Mail Transfer Protocol* (SMTP):** protocolo usado en Internet para la transmisión de mensajes de correo electrónico, especificado en el estándar RFC 821.

S/MIME: Ver *Secure MIME*.

SMTP: Ver *Simple Mail Transfer Protocol*.

Sobre digital: técnica para proporcionar confidencialidad, consistente en cifrar los datos con una clave de sesión simétrica, y añadir al mensaje esta clave de sesión cifrada con la clave pública de cada destinatario.

SSH: Ver *Secure Shell*.

Subclave PGP: clave PGP asociada a una clave principal, de forma que normalmentela clave principal se utiliza para firmar y sus subclaves para cifrar (las subclaves están definidas solamente en OpenPGP, no en el sistema PGP original).

Bibliografía

1. Barrett, D. J.; Silverman, R. (2001). *SSH, The Secure Shell: The Definitive Guide.* Sebastopol: O'Reilly.

2. Stallings, W. (2003). *Cryptography and Network Security, Principles and Practice, 3rd ed.* Upper Saddle River: Prentice Hall.